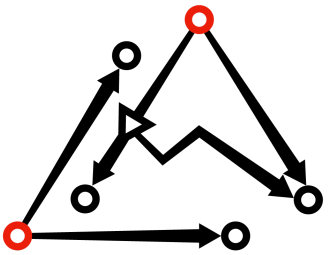

Least Authority

PRIVACY MATTERS

Coin Voting 2.0

Security Audit Report

ZCG

Final Audit Report: 25 March 2025

Table of Contents

Overview

Background

ZCG has requested that Least Authority perform a security audit of Coin Voting 2.0.

Project Dates

- **February 19, 2025 - March 11, 2025:** Initial Code Review *(Completed)*
- **March 11, 2025:** Delivery of Initial Audit Report *(Completed)*
- **March 19, 2025:** Verification Review *(Completed)*
- **March 19, 2025:** Delivery of Final Audit Report *(Completed)*
- **March 25, 2025:** Delivery of Updated Final Audit Report *(Completed)*

Review Team

- George Gkitsas, Security / Cryptography Researcher and Engineer
- Mirco Richter, Cryptography 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 the Coin Voting 2.0 followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

- Frontend:
  - https://github.com/hhanh00/zcash-vote-create
    - a51e965524bdc968a9ea11925d9b42b9d87cd8b2
  - https://github.com/hhanh00/zcash-vote-app
    - a8310693bee849f40239d44e97ff30365ea7f861
  - https://github.com/hhanh00/zcash-vote-audit
    - e2ecb5328e86c9e5b2dec0a9d868990ec23452be
  - https://github.com/hhanh00/zcash-vote-server
    - 77471084ac7472a01a0f82c4f973ac500e0dc38b
- Backend:
  - https://github.com/LeastAuthority/zcash-vote
    - 30bfba872ef4088b52b23b1b5b55415c5f634248
- Orchard Audit diff:
  - https://github.com/zcash/orchard/compare/main...hhanh00:orchard:vote

The following code repositories were cloned for the audit and used for verification with the specified versions.

- Zcash-vote-create:
  - https://github.com/LeastAuthority/zcash-vote-create
    - 1fe0c1c127303e6a3239be359eb201f3d6da0e9d
- Zcash-vote-app:
  - https://github.com/LeastAuthority/zcash-vote-app
    - 403e3780626fc034319282155c111a0dbf4ee2a2

- Zcash-vote-audit:
  - https://github.com/LeastAuthority/zcash-vote-audit
    - b3f2af7f32ee41883e4a808b9e9b5a8b024b3436

- Zcash-vote-server:
  - https://github.com/LeastAuthority/zcash-vote-server
    - fedbc6cbac4a966e3d780088ece1209d9fdaf744
- Zcash-vote
  - https://github.com/LeastAuthority/zcash-vote
    - 8b42736fdde90a8ad19619a6fad9fae7e1c16769

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.

Supporting Documentation

The following documentation was available to the review team:

- Forum: https://forum.zcashcommunity.com/t/coin-voting-2-0/49177
- Project Proposal Documentation: https://hhanh00.github.io/coin-voting-book/proposal/details.html
- Project specification https://github.com/hhanh00/coin-voting-book
- Reference circuits and gadgets https://github.com/zcash/orchard

In addition, this audit report references the following documents:

- D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash Protocol Specification | Version 2024.5.1-112-gcf7a5c [NU6]." 2024, [HBH+24]
- Path Traversal: https://owasp.org/www-community/attacks/Path_Traversal

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

Coin Voting 2.0 is designed as a voting system that aims to be privacy-preserving and secure. The voting system leverages Zcash Orchard's shielded notes to utilize Zcash balances as verifiable voting power by converting ZEC into anonymous, coin-weighted votes. Coin Voting 2.0 runs the elections using a public ledger called the "Voting Blockchain," which is a centralized ledger maintained by an Election Authority. The solution comprises four components:

1. `vote-server` maintains the Voting Blockchain and responds to requests about the data held in it;
2. `vote-app` is used by the users to register and cast votes;
3. `vote-create` is used to create a new election; and
4. `vote-audit` is used to verify the election results.

We analyzed the implemented circuits against the original Orchard circuits and the specification [HBH+24], focusing on missing constraints. Additionally, we examined the domain derivation and the associated domain nullifier, their implementation, and their usage in the Voting circuit, but found no further issues beyond the unintended permutation of arguments (Issue F). Specifically, we compared the Voting circuit to the original Orchard circuit and assessed the security implications of the deviation, without reasoning about the unchanged components.

We audited the ballot structure, vote encryption and decryption, as well as the (non) inclusion trees, and found no issues.

Our team also analyzed the election ID derivation and recommends implementing election IDs as unique domain specifiers (Issue J). Beyond this issue, we did not identify any concerns within this context.

We reviewed the high-level architecture as presented in the `coin-voting-book`, assessed the system's trust assumptions, and identified multiple issues. In particular, we observed that a malicious authority can manipulate the outcome of an election by refusing to acknowledge Voting transactions (Issue E), and a malicious server can misrepresent election details (Issue J and Issue K). In addition, a malicious creator can manipulate an election by spending candidates (Issue G).

We assessed the security of communications between the different applications and found that the `vote-server` relies solely on unauthenticated communication. Additionally, we discovered that the `vote-app` is permitted to establish unauthenticated connections to blockchain nodes. Both findings introduce the risk of man-in-the-middle (MITM) attacks (Issue A and Issue B).

We reviewed the security of sensitive values throughout their lifecycle across all applications and found no measures in place to protect against residual data leakage (Issue D).

We evaluated potential security issues stemming from untrusted inputs and identified a limited directory traversal attack vector in the `vote-app` (Issue C).

We assessed the consistency and correctness of the `vote-server` database logic and found no issues in the database. However, we identified an issue in the election `vote-create` database (Suggestion 11).

We also examined the feasibility of a denial-of-service (DoS) attack by splitting a node's voting weight into a large number of votes but concluded that this attack is not feasible.

System Design

Our team examined the design of the Coin Voting 2.0 and found that security has generally been well considered. Specifically, the circuit implementation follows Zcash Orchard best practices, introducing only minimal, carefully designed modifications, without compromising the security assumptions of the components used. However, we identified areas of improvement (Suggestion 1, Suggestion 2, Suggestion 4, Suggestion 5, Suggestion 7, Suggestion 8, and Suggestion 9). Additionally, system security could be improved by implementing `vote-create` as a multi-party computation, such that no single party has full access to the spending keys of any candidate (Suggestion 6).

Dependencies

We examined the implemented dependencies in the codebase and identified several vulnerable instances. We recommend improving dependency management (Issue H).

Code Quality

We performed a manual review of the repositories in scope and found the code to be well-organized and in adherence to Rust, TypeScript, and Zcash Orchard best practices. However, we found that the SQL code could benefit from refinements (Suggestion 2).

Tests

During our review, our team found no tests within the codebase. We recommend implementing a test suite, which helps identify implementation errors that could lead to security vulnerabilities (Suggestion 1).

Documentation and Code Comments

Although the code lacks inline documentation, the project specification provided by the Coin Voting team sufficiently describes the system's intended functionality. However, we recommend enhancing the documentation to provide more comprehensive coverage of the threat model (Suggestion 4).

Scope

The scope of this review was sufficient and included all security-critical components. However, since some identified issues require significant protocol or code extensions, we recommend a comprehensive follow-up audit once these issues are resolved and development is finalized.

Guidance for Operators and Users

Part of the security of Coin Voting depends on choices made by users and operators. Below we outline the best practices they can follow.

1. The election creator should publish the election ID. The chosen publishing method affects the security since different choices involve varying levels of trust in the servers. The strictest option is to commit the election ID to the Zcash blockchain, as it is immutable. Other methods introduce different risks. For example, a creator publishing the election ID to a website carries the risk of the website being maliciously altered.
2. Operators managing the servers should configure HTTPS for both the app-to-server and server-to-lightnode communications.
3. A voter is offered the option to add multiple blockchain nodes. We recommend users add multiple nodes to minimize the trust in a single node. Ideally the user should utilize an honest majority of nodes.

4. Voters should also check the election ID presented in their application against the election ID published by the creators.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. 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	STATUS
Issue A: Communication Between the Voting and Audit Applications and the Voting Server Is Susceptible to Man-In-The-Middle Attacks	Unresolved
Issue B: Connection Between the Voting and Election Creation Applications and Blockchain Nodes Is Susceptible to Man-In-The-Middle Attacks	Unresolved
Issue C: Malicious Server Can Launch Directory Traversal Attack Against Clients	Resolved
Issue D: Leakage Through Residual Data	Unresolved
Issue E: Authority Can Manipulate Election Outcome	Partially Resolved
Issue F: Domain Nullifier Constraints Violate the Specification	Resolved
Issue G: Missing Unspent Candidates Check	Resolved
Issue H: Usage of Vulnerable Dependencies	Unresolved
Issue I: Trust on Fetched Blockchain Node Data	Resolved
Issue J: Non-Unique Election Identifiers	Resolved
Issue K: Missing On-Chain Election Commitment	Partially Resolved
Issue L: Reusing of Candidate Addresses	Resolved
Suggestion 1: Implement a Test Suite	Not Implemented
Suggestion 2: Improve Code Quality	Not Implemented
Suggestion 3: Improve Constraint Descriptor	Not Implemented
Suggestion 4: Improve Documentation	Not Implemented
Suggestion 5: Handle Chain Reorgs/Forks	Not Implemented
Suggestion 6: Implement Multi-Party Creator	Not Implemented
Suggestion 7: Specify Minimum Amount of Voting Weight for Valid Elections	Not Implemented

Suggestion 8: Specify How Ties Are Broken	Not Implemented
Suggestion 9: Validate Election Start and End Times	Not Implemented
Suggestion 10: Restrict Auditor Access To Viewing Keys Only	Not Implemented
Suggestion 11: Remove Database Collisions in vote-create	Not Implemented
Suggestion 12: Introduce Election-Specific Filtering in Database Queries	Not Implemented

Issue A: Communication Between the Voting and Audit Applications and the Voting Server Is Susceptible to Man-In-The-Middle Attacks

Location

`zcash-vote-app/src/Election.tsx#L39`

`src-tauri/src/lib.rs#L24-L25`

`hhanh00/zcash-vote-server`

Synopsis

The communication between the Voting application and a Voting server uses plain HTTP, which enables man-in-the-middle (MITM) attacks.

Impact

An attacker can modify the contents of communications. Specifically, it can alter the name of the election, the text describing the voting candidates, and the candidate addresses. This can mislead the target into casting votes for candidates other than their intended choice. It can also add new candidates that are not part of the actual election, and any votes cast for them will not be counted. Additionally, it can hide candidates from the user by omitting them.

Preconditions

The attacker would need to utilize MITM techniques, such as ARP or DNS spoofing.

Feasibility

This attack is moderately feasible.

Severity

High.

Technical Details

A MITM attacker can intercept the communication between the Voting application and the Voting server and can also alter or conceal any information that is not cryptographically secured, as well as hide cryptographically secured information. Examples include the Election definition shared by the server when the application first connects and the reporting information that the server provides to the Audit application.

Remediation

We recommend implementing HTTPS for all communications on the server side, disabling `HTTP-only`, and using an audited library, such as `rustls`. Preferably, this should be complemented by the use of the formally verified TLS `v1.3` protocol.

On the client side (Voting and Auditing applications), we recommend that the application prohibit `HTTP-only` connections. For the HTTPS configuration, we suggest utilizing certificate pinning to future-proof against PKI compromises. We additionally recommend configuring the application to reject self-signed certificates and employ HSTS to protect against downgrading attacks.

Status

The Coin Voting team has not implemented any code changes but stated that the operator has the option to configure TLS. We recommend setting TLS as the default to adhere to security-by-default best practices.

Verification

Unresolved.

Issue B: Connection Between the Voting and Election Creation Applications and Blockchain Nodes Is Susceptible to Man-In-The-Middle Attacks

Location

`src-tauri/src/lib.rs#L85`

`src-tauri/src/download.rs#L38`

Synopsis

Both the Voting application and the election creation application rely on a blockchain node to obtain blockchain information. Consequently, the applications could accept `HTTP-only` node URLs, which allows for man-in-the-middle (MITM) attacks. Additionally, the connection is unauthenticated, which does not protect against malicious nodes.

Impact

In combination with Issue I, an attacker can modify or selectively hide the contents of communications.

Additionally, even if Issue I were resolved, an attacker could still hijack the communication channel and provide a chain of blocks that passes verification but is not the chain of blocks that appears in the blockchain.

Preconditions

The attacker would need to utilize MITM techniques, such as ARP or DNS spoofing.

The attacker could deploy their own malicious node server by convincing application users to connect to it.

Feasibility

This attack is difficult to execute.

Severity

Low.

Remediation

For all affected applications, we recommend only accepting HTTPS URLs for blockchain nodes.

Status

Although the blockchain node default currently uses HTTPS and is specified in the applications' code, there is an option to bypass this selection. Enforcing HTTPS-only options as a proactive measure will increase user security and reduce opportunities for attackers. Alternatively, we recommend at a minimum notifying users when a non-HTTPS link is used.

Verification

Unresolved.

Issue C: Malicious Server Can Launch Directory Traversal Attack Against Clients

Location

`zcash-vote-app/src/Election.tsx#L40`

Synopsis

The `name` field in the Election application is controlled by the server and used to delete a `.db` file on the host of the Voting application. Since the application does not fully sanitize this input, a directory traversal attack is possible.

Impact

A malicious server can delete an arbitrary `.db` file on the host of the Voting application.

Preconditions

An attacker would need to run a malicious Voting server.

Feasibility

Straightforward.

Severity

Medium.

Technical Details

The value of the Election `path` field is controlled by the server. It is then passed to the `save_db` function unsanitized. The `save_db` function proceeds with deleting from the file system a file named according to the value of `path`. Since `path` is unsanitized, a malicious server could exploit directory/path traversal attacks to trigger the deletion of any file that the application has access to.

The attack is limited exclusively to files with `.db` extensions due to a filter implemented in the frontend.

There are multitudes of ways to execute this type of attack, such as introducing unsafe characters, employing parent directory operators (for example: `../`), and utilizing symbolic links.

Remediation

We recommend properly sanitizing the `name` field in the Election definition controlled by the server.

We additionally recommend:

- Using `sanitize_filename` to clean up file names from unsafe characters;
- Using `path_clean` to normalize paths;
- Using `std::fs::canonicalize` to obtain an absolute path; and
- Verifying that the absolute path falls within the intended directory.

Status

While the file path is controlled by the server, it is set as the default, and the user always has the option to review and change it. Additionally, the user is prompted before any file overwrite occurs. Given these safeguards, we have determined this to be a non-issue, as the user is involved at two stages.

Verification

Resolved.

Issue D: Leakage Through Residual Data

Location

`src-tauri/src/lib.rs#L41`

`zcash-vote-app/src/Election.tsx#L142-L143`

Synopsis

Sensitive data remains resident in memory or storage after the application's execution has terminated, potentially exposing it to unauthorized access or compromise.

Impact

An attacker could exploit this vulnerability to gain unauthorized access to the wallet seed phrase.

Preconditions

The attacker would have to be logically or physically local to the target device.

Feasibility

Low.

Severity

Medium.

Technical Details

The Voting application does not manage data at rest securely. It stores the user's seed phrase and some wallet addresses in a local, unencrypted `SQlite` database file. The database file is also retained even after an election has concluded.

Additionally, the Voting application does not implement security measures to protect data in use. As a result, sensitive information, including the user's seed phrase, signing key, and viewing keys, is vulnerable to residual memory leakage. Attackers can exploit various methods to retrieve residual data, such as reallocating recently freed memory, triggering core dumps, or accessing swap memory.

The Election creation application also does not implement security measures to protect data in use for the seed phrase.

Remediation

For data-at-rest security, we recommend implementing one of the following measures:

- using encrypted storage for the `.db` file or utilizing database encryption; or
- using the host's keyring feature for storing sensitive data.

Irrespective of the remediation path chosen, we further advise that the `.db` files be deleted upon the conclusion of the election.

For data-in-use security, we recommend utilizing libraries for the secure handling of sensitive values in memory, such as `zeroize`, `secrecy`, or `secrets`.

Status

The Coin Voting team has not implemented any code changes. The team stated that they plan to incorporate memory zeroization to minimize exposure of residual data during use. Given that the Zcash threat model assigns users the responsibility for securing their seed phrase at rest, we find that Coin Voting aligns with this approach and, therefore, do not consider encryption of data at rest for the seed phrase a requirement. However, we still encourage implementing data-at-rest encryption through encrypted storage, database encryption, a keystore, or other secure methods.

Additionally, we recommend informing users of the risks through a UI message. The message should indicate that the seed phrase will be stored unencrypted and advise users to delete the .db file upon completing their voting actions.

Verification

Unresolved.

Issue E: Authority Can Manipulate Election Outcome

Location

`coin-voting-book/tree/main`

Synopsis

The authority runs a private blockchain that must include all transactions that clients send to it in order to run a proper election. However, if the authority is malicious, it could refuse to acknowledge and include some transactions into its blockchain.

Impact

Refraining from including certain transactions manipulates the outcome of an election. Moreover, if the authority and the creator collude, they can strategically refuse transactions to enforce any desired election outcome. This is possible because the creator has access to each candidate's spending and viewing key, enabling it to compute the election outcome for any subset of `send` transactions in order to include only those that yield the intended result.

Preconditions

A malicious authority would have to rewrite the software. For strategic manipulation, the creator and the authority would have to collude.

Feasibility

Straightforward.

Severity

Critical.

Technical Details

The authority organizes Voting transactions in a self-hosted blockchain, maintaining full control over which transactions are included. As a result, no consensus can be reached about the Voting Blockchain state among all participating parties. Consider the following two scenarios:

1. Scenario A: Client A is honest and sends a Voting transaction to the authority. The authority is malicious and refuses to acknowledge and include the transactions. Client A claims it has sent the transaction, and the authority claims that client A has not sent the transaction.
2. Scenario B: Client A is malicious and does not send a transaction to the authority. Client A then claims it has sent the transaction, and the authority claims that client A has not sent the transaction.

Both scenarios are indistinguishable for all other participants in the network.

Remediation

We recommend implementing a consensus mechanism to ensure that all participants reach consensus about the set of voting transactions. Since BFT consensus mechanisms require honest supermajorities, which are difficult to guarantee in permissionless, fully untrusted networks, we suggest leveraging Zcash's main chain in combination with a system such as IPFS, as the source of truth. Voting transactions are then written to IPFS, with the IPFS address committed on the Zcash main chain.
The auditor (public verifier) must then run a check to verify that all publicly agreed-upon transactions have been included into the Voting chain by the authority.

Status

The Coin Voting team has implemented a mitigation that utilizes `cometBFT` to establish consensus on the set of `send` transactions. However, since `cometBFT` requires an honest supermajority of validators ($> 2/3 \ n$), this mitigation only partially resolves the issue because it shifts honesty from a single entity (the authority) to a group of entities ($2/3$ of the validator set chosen by the creator). This provides a significantly weaker security guarantee than that of the Zcash blockchain: In Zcash, consensus is strongly favored by economic incentivization and penalties, whereas the implemented `cometBFT` validator set lacks such mechanisms. Moreover, in most scenarios, the validator set is expected to be much smaller than the number of Zcash miners.

We therefore still recommend implementing a proper remediation, such as writing ballots into memos on the Zcash main chain, which becomes possible after ZIP-0231. This approach allows the Voting protocol to inherit the same security properties as Zcash.

In addition, we observed that the `zcash-vote-app` still sends its votes exclusively to the authority, and the set of validators is not included in the election description. Instead, the validator set should be committed in the same manner as any other part of an election description (Issue J). Furthermore, votes should be transmitted (e.g., via a P2P Gossip protocol) to all validators or at least to an honest majority ($> 1/3 \ n$) of validators.

The team has implemented an option that enables the user to add multiple nodes. The responsibility of adding validators that are at least the number of honest majority is left to the user.

Verification

Partially Resolved.

Issue F: Domain Nullifier Constraints Violate the Specification

Location

orchard/src/keys.rs#L234

src/circuit/gadget.rs#L228

src/note/nullifier.rs#L75

src/implementation/circuit.md

Synopsis

According to the specification (src/implementation/circuit.md), the domain nullifier should be computed/constrained as `DeriveDomainNullifier=(HASH(nk||HASH(edh||rho))+psi)*Nk+cm`. However, the circuit deviates from this definition and incorrectly constrains `DeriveDomainNullifier=(HASH(nk||HASH(rho||edh))+psi)*Nk+cm`.

While the function `prf_nf_domain` adheres to the specification and computes the domain nullifier witness as expected, the issue remains undetected in tests because the function call permutes the inputs in a way that satisfies the incorrect constraint.

Impact

Although the code works as intended, this is only because the permuted constraint is canceled by the permuted arguments in the `prf_nf_domain` function—a situation that may confuse auditors.

Severity

Low.

Remediation

We recommend implementing the `DeriveDomainNullifier` constraints as defined in the specification and using the arguments in `prf_nf_domain` in the correct order.

Status

The Coin Voting team has implemented the proper constraint.

Verification

Resolved.

Issue G: Missing Unspent Candidates Check

Location

coin-voting-book/tree/main

Synopsis

Candidates' notes can be spent to cast votes for other candidates.

Impact

Any process with access to the candidate's spending keys can use that candidate's voting weight to vote for any other candidate, effectively enforcing arbitrary election outcomes.

Preconditions

This issue necessitates access to the candidate's spending key.

Feasibility

This exploit is straightforward for the creator, as they already own the candidate's spending keys.

Severity

Critical.

Remediation

In order to prevent this attack, the auditors (public verifiers) must check that all candidates remain unspent at the end of an election. Although the creator can use this check to invalidate an election (by spending a candidate), we consider this acceptable since the candidate owns the election and hence has the right to invalidate it. If elections are required that cannot be invalidated in this manner, we recommend implementing the creator as a multi-party process (Suggestion 6).

Status

The Coin Voting team has implemented the remediation as recommended.

Verification

Resolved.

Issue H: Usage of Vulnerable Dependencies

Location

`zcash-vote-server/Cargo.toml`

`zcash-vote-app/src-tauri/Cargo.toml`

`zcash-vote-audit/src-tauri/Cargo.toml`

`zcash-vote/Cargo.toml`

`zcash-vote-create/src-tauri/Cargo.toml`

`zcash-vote-app/package.json`

`zcash-vote-audit/package.json`

`zcash-vote-create/package.json`

Synopsis

Running `cargo audit` and `yarn audit` revealed several dependency issues across all applications.

Severity

Undetermined.

Technical Details

We identified the following vulnerabilities in `zcash-vote-app`:

- https://rustsec.org/advisories/RUSTSEC-2025-0004
- https://rustsec.org/advisories/RUSTSEC-2024-0336
- https://rustsec.org/advisories/RUSTSEC-2024-0413
- https://rustsec.org/advisories/RUSTSEC-2024-0416
- https://rustsec.org/advisories/RUSTSEC-2024-0388
- https://rustsec.org/advisories/RUSTSEC-2021-0141
- https://rustsec.org/advisories/RUSTSEC-2024-0412
- https://rustsec.org/advisories/RUSTSEC-2024-0418
- https://rustsec.org/advisories/RUSTSEC-2024-0370
- https://rustsec.org/advisories/RUSTSEC-2021-0127
- https://rustsec.org/advisories/RUSTSEC-2024-0429
- https://rustsec.org/advisories/RUSTSEC-2022-0070
- https://www.npmjs.com/advisories/1094698
- https://www.npmjs.com/advisories/1102341

We identified the following vulnerabilities in `zcash-vote-audit` and in `zcash-vote-create`:

- https://rustsec.org/advisories/RUSTSEC-2024-0336
- https://rustsec.org/advisories/RUSTSEC-2024-0413
- https://rustsec.org/advisories/RUSTSEC-2024-0416
- https://rustsec.org/advisories/RUSTSEC-2021-0141
- https://rustsec.org/advisories/RUSTSEC-2024-0412
- https://rustsec.org/advisories/RUSTSEC-2024-0418
- https://rustsec.org/advisories/RUSTSEC-2024-0411
- https://rustsec.org/advisories/RUSTSEC-2024-0417
- https://rustsec.org/advisories/RUSTSEC-2024-0414
- https://rustsec.org/advisories/RUSTSEC-2024-0415
- https://rustsec.org/advisories/RUSTSEC-2024-0420
- https://rustsec.org/advisories/RUSTSEC-2024-0419
- https://rustsec.org/advisories/RUSTSEC-2024-0370
- https://rustsec.org/advisories/RUSTSEC-2021-0127
- https://rustsec.org/advisories/RUSTSEC-2024-0429
- https://rustsec.org/advisories/RUSTSEC-2022-0070
- https://www.npmjs.com/advisories/1094698
- https://www.npmjs.com/advisories/1102341

We identified the following vulnerabilities in `zcash-vote-server`:

- https://rustsec.org/advisories/RUSTSEC-2024-0336
- https://rustsec.org/advisories/RUSTSEC-2021-0141
- https://rustsec.org/advisories/RUSTSEC-2021-0127
- https://rustsec.org/advisories/RUSTSEC-2022-0070

Remediation

We recommend following a process that emphasizes secure dependency usage to avoid introducing vulnerabilities and to mitigate supply-chain attacks, which includes:

- Manually reviewing and assessing currently used dependencies;
- Upgrading dependencies with known vulnerabilities to patched versions with fixes;

- Replacing unmaintained dependencies with secure and battle-tested alternatives, if possible;
- Pinning dependencies to specific versions, including pinning build-level dependencies in the `package.json` and `Cargo.toml` files to a specific version;
- Only upgrading dependencies upon careful internal review for potential backward compatibility issues and vulnerabilities; and
- Including automated dependency auditing reports in the project's CI/CD workflow.

Status

The Coin Voting team stated that they plan to implement the recommended remediation for their immediate dependencies in a future release.

Verification

Unresolved.

Issue I: Trust on Fetched Blockchain Node Data

Location

`src-tauri/src/download.rs#L27`

Synopsis

The Voting application does not verify fetched blocks and their order in the blockchain. A malicious blockchain node can omit or replace certain blocks, thus misleading the Voting and Create applications.

Impact

A malicious blockchain node can modify the view of the blockchain that the Voting and Create applications have.

Preconditions

The attacker would need to be able to operate a malicious blockchain node and convince users to use this node.

Alternatively, when combined with Issue B, an attacker would need to be able to launch a man-in-the-middle attack against an honest HTTP blockchain node.

Feasibility

This attack is moderately feasible.

Severity

Low.

Technical Details

In order to ensure that the blockchain data is not tampered with, the applications need to check each block's integrity and validity and verify that the chain of blocks is consistent. Omitting these checks puts full trust in the blockchain node, which contradicts the fundamental purpose of a blockchain.

Remediation

We recommend implementing all the necessary block checks. We also recommend fetching blocks before the starting height to further increase confidence in the data.

Optionally, to further increase security and resilience, we recommend using multiple blockchain nodes, significantly increasing the effort required for an attack.

Status

The Coin Voting team argued convincingly that this is not an issue, as the election file contains the `cmx_root`. Any reordered or missing transaction will change the resulting `cmx_root` calculated by the Voting application.

Verification

Resolved.

Issue J: Non-Unique Election Identifiers

Location

`coin-voting-book/tree/main`

Synopsis

Elections are uniquely characterized by their creators, their candidate questions and addresses, the election name and its ID, as well as the chosen start, end, and close height. However, the election ID does not include all this data into a unique election identifier.

Impact

Since election IDs are not unique, they allow for potential replay attacks, such as running the same election on different registration or voting windows or by different creators. Once the candidates' spending keys are published, an attacker can reuse the same addresses.

Severity

High

Remediation

We recommend that election IDs uniquely identify any election by committing to all data that is publicly known, and include a signature of the creator. In addition, auditors (public verifiers) should verify that the commitment is computed correctly and that the signature is valid.

Status

The Coin Voting team has implemented a unique election identifier. Additionally, the serialization of the election fields implicitly uses delimiters from `protobuf`, which safeguards against deserialization confusion attacks.

Verification

Resolved.

Issue K: Missing On-Chain Election Commitment

Location

`coin-voting-book/tree/main`

Synopsis

The creator trusts the server to represent the election as intended. However, a malicious server could change election details, such as associations between candidate descriptions and candidate addresses, or add / remove candidates.

Impact

A malicious server can misrepresent an election in arbitrary ways, potentially altering its outcome.

Preconditions

The server must be malicious and operated by a party other than the creator.

Feasibility

Straightforward.

Severity

High.

Remediation

We recommend that the creator commit to the structure of an election on the Zcash blockchain, such that every auditor (public verifier) can independently check that the election is correctly represented by the server.

Status

The Coin Voting team has acknowledged that the creator should publish an election domain commitment, and that voters should verify that the election they are voting on has the correct commitment. However, the team has decided to delegate to the creator the method of committing to the election ID. Consequently, the user is responsible for checking the election ID in their application against the published one. For fully resolving this issue, our team recommends including the commit in the Zcash blockchain instead of allowing mutable publishing methods such as webpages.

Verification

Partially Resolved.

Issue L: Reusing of Candidate Addresses

Location

`src-tauri/src/lib.rs#L42`

Synopsis

The address for each election candidate is randomly generated and thus not bound to the candidate's purpose. This allows for scenarios where the candidate address is reused in different elections and for different purposes.

Impact

This issue could enable replay attacks, allowing candidate addresses to be reused for different purposes and across different elections.

Preconditions

Executing this exploit requires either running multiple malicious voting servers or compromising an existing one.

Feasibility

This attack is moderately feasible.

Severity

High.

Remediation

We recommend using domain separation when deriving the candidate addresses. The domain separation must uniquely bind the election ID, the starting and ending height, the candidate's purpose, and other relevant information to the candidate address.

Note that this remediation introduces an interdependency between candidate address generation and election ID generation, as discussed in Issue J. Careful design decisions are required to ensure both conditions are met simultaneously.

Status

This issue is a broader implication of Issue J. Hence, the Coin Voting team's implementation of a unique election identifier that effectively resolves Issue J also addresses this issue.

Verification

Resolved.

Suggestions

Suggestion 1: Implement a Test Suite

Location

`zcash-vote-create`

`zcash-vote-app`

`zcash-vote-audit`

`zcash-vote-server`

`zcash-vote`

`zcash-orchard/pull/1`

Synopsis

Our team found no tests in the repository in scope. Sufficient test coverage should include tests for success and failure cases, which helps identify potential edge cases, and protect against errors and bugs that may lead to vulnerabilities or exploits. A test suite that includes a minimum of unit tests and integration tests adheres to development best practices. In addition, end-to-end testing is also recommended to assess if the implementation behaves as intended.

Mitigation

We recommend creating comprehensive test suites for all components of the codebase to facilitate the identification of implementation errors and potential security vulnerabilities by developers and security researchers.

Status

The Coin Voting team has not addressed this suggestion. As a result, it remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 2: Improve Code Quality

Location

`zcash-orchard diff`

`zcash-vote/src/db.rs#L17`

`main/src/db.rs#L39`

Synopsis

During our extensive review of the codebase, our team identified practices that impact the quality, readability, and maintainability of the codebase. Below, we share a non-exhaustive list of remediation measures addressing the code quality issues we identified:

1. Utilize names instead of literal constants;
2. Use explicit row names instead of name-value pairs in the database; and
3. Explicitly specify the trigger in UPSERT SQL statements.

Mitigation

We recommend improving code quality by addressing the items listed above, as well as any similar instances.

Status

The Coin Voting team has not addressed this suggestion. As a result, it remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 3: Improve Constraint Descriptor

Location

`src/vote/circuit.rs#L315`

Synopsis

The constraints descriptor currently specifies an XOR constraint but correctly implements an OR constraint, which is potentially misleading for future audits.

Mitigation

We observed that the same error exists in the original Orchard circuit, and our team has submitted a PR to address it. We recommend handling the situation in the same manner as Zcash.

Status

The Coin Voting team has not addressed this suggestion. As a result, it remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 4: Improve Documentation

Location

coin-voting-book/tree/main

Synopsis

Currently, no trust assumptions are specified, and availability assumptions are made for the various parties.

Mitigation

We recommend improving the threat model in the documentation.

Status

The Coin Voting team has not addressed this suggestion. As a result, it remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 5: Handle Chain Reorgs/Forks

Location

coin-voting-book/tree/main

Synopsis

The Zcash blockchain can experience forks/chain reorgs during the registration window, which are currently not handled.

Mitigation

We recommend implementing a mechanism that can flexibly handle reorgs during the registration period.

Status

The Coin Voting team has not addressed this suggestion. As a result, it remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 6: Implement Multi-Party Creator

Location

`zcash-vote-create`

`coin-voting-book/tree/main`

Synopsis

The creator owns both the spending keys as well as the viewing keys of all candidates in any election it creates. As explained in Issue G, the creator can use the spending keys to invalidate an election. Moreover, it may selectively disclose the viewing keys to certain participants while withholding them from others, providing an informational advantage. As a result, for some elections, it may be beneficial to distribute the spending key generation among multiple parties.

Mitigation

We recommend implementing the creator as a multi-party key generation schema. This prevents single party ownership of spending keys and allows for coordinated distribution of viewing keys in case de-shielded elections are required.

Status

The Coin Voting team has not addressed this suggestion. As a result, it remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 7: Specify Minimum Amount of Voting Weight for Valid Elections

Location

`coin-voting-book/tree/main`

Synopsis

Auditors (public verifiers) do not verify whether any candidate received any voting weight, which can result in elections with no participation.

Mitigation

We recommend that the creator specify a minimum threshold of voting weight required for an election to be considered valid, excluding the edge case where no votes are cast, and all candidates have an equal voting weight of zero.

Status

The Coin Voting team has not addressed this suggestion. As a result, it remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 8: Specify How Ties Are Broken

Location

coin-voting-book/tree/main

Synopsis

The creator does not specify how ties should be broken in an election, potentially allowing for non-deterministic/unexpected outcomes.

Mitigation

We recommend that the creator specify a field that describes how ties are broken.

Status

The Coin Voting team has not addressed this suggestion. As a result, it remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 9: Validate Election Start and End Times

Location

src-tauri/src/lib.rs#L89

Synopsis

If the user provides the same election start and end values, a division by zero occurs. Additionally, the case where `end < start` is logically invalid and should be also checked against.

Mitigation

We recommend implementing a check that only allows for the case `end > start`.

Status

The Coin Voting team has not addressed this suggestion. As a result, it remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 10: Restrict Auditor Access To Viewing Keys Only

Location

coin-voting-book/tree/main

Synopsis

The Audit application only requires the incoming viewing keys of the election candidates' addresses in order to publicly verify an election. Instead, the seed phrases of those candidates are shared. This results in unnecessary access to the candidate addresses.

Mitigation

We recommend only sharing the incoming viewing keys of the candidate addresses with the Voting auditors.

Status

The Coin Voting team has not addressed this suggestion. As a result, it remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 11: Remove Database Collisions in vote-create

Location

src-tauri/src/lib.rs#L88

Synopsis

The election ID under which a new election is stored in the database is hardcoded to `0`. If multiple elections are created, their data will collide in the database under the `0` ID. As a result, this would lead to data corruption and can cause unintended consequences.

Mitigation

We recommend using the actual election ID for each created election.

Status

The Coin Voting team has not addressed this suggestion. As a result, it remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 12: Introduce Election-Specific Filtering in Database Queries

Location

main/src/trees.rs#L7

main/src/trees.rs#L27

Synopsis

The queries do not currently filter their results based on the election ID. When the database holds data for multiple elections, the queries retrieve the nullifiers and commitments for all elections instead of only the relevant one.

Mitigation

We recommend fetching only the information relevant to the specific election by using a `WHERE election = ?1` with the election ID as a parameter.

Status

The Coin Voting team has not addressed this suggestion. As a result, it remains unresolved at the time of verification.

Verification

Unresolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

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

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up 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.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.