zkVerify Foundation, zkVerify Blockchain Review

Security Assessment

February 6, 2025

Prepared for:
Rosario Pabst
zkVerify Foundation

Prepared by: **Gustavo Grieco and Simone Monica**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Gustavo Grieco, Consultant
gustavo.grieco@trailofbits.com

Simone Monica, Consultant
simone.monica@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
January 7, 2025	Pre-project kickoff call
January 13, 2025	Status update meeting #1
January 17, 2025	Delivery of report draft
January 17, 2025	Report readout meeting
February 6, 2025	Delivery of final comprehensive report

Executive Summary

Engagement Overview

zkVerify Foundation engaged Trail of Bits to review the blockchain-related security of zkVerify, a blockchain that enables decentralized verification of zero-knowledge proofs issued from different proof systems.

A team of two consultants conducted the review from January 6 to January 15, 2025, for a total of three engineer-weeks of effort. Our testing efforts focused on the Substrate node procedure, the `aggregate` and `hyperbridge_aggregations` pallets, and the general runtime parameter usage. With full access to source code and documentation, we performed static analysis and manual code review of the codebase.

Observations and Impact

Only informational-severity issues were found during this engagement, one related to event emission (TOB-HRZ-1) and the other related to vulnerable Substrate node dependencies (TOB-HRZ-2). The audited code is properly organized and follows best practices in terms of data validation and state transition handling.

Recommendations

Based on the findings identified during the security review as well as discussions with the client, Trail of Bits recommends that zkVerify Foundation take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or any refactor that may occur when addressing other recommendations. Also consider addressing the code quality recommendations provided in appendix B.
- **Review the recommendations for the Substrate node upgrade process provided in appendix C.** The zkVerify Substrate node was upgraded to 1.15. While the upgrade was not critical (since it involved only the testnet, which will eventually be reinitialized), future upgrades should be performed with extra care to avoid breaking the chain.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	0
Low	0
Informational	2
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Auditing and Logging	1
Patching	1

Project Goals

The engagement was scoped to provide a security assessment of zkVerify Foundation's zkVerify. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are best practices followed for upgrading a Substrate-based blockchain?
- Are there incorrect state transitions for the state machine in the `aggregate` pallet?
- Are there appropriate access controls for the pallets' extrinsics?
- Are the weights correctly set for the pallets' extrinsics?
- Are the pallets configured in the runtime correctly and securely?
- Is the value used by the pallets properly secured and accounted for?

Project Targets

The engagement involved reviewing and testing the following target.

zkVerify

Repository	https://github.com/zkVerify/zkVerify
Version	`00332a52d562e763ed8bd55bdb584d4c366cc11e`, PR #168
Type	Rust
Platform	Substrate

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **Substrate node upgrade:** We reviewed the changes made to upgrade the zkVerify Substrate node from 1.10 to 1.15, including the use of new parameters, the removal of deprecated or obsolete parameters, storage changes, and specific migrations. We also investigated vulnerable Cargo dependencies in the upgraded code.

- **Runtime configuration of pallets:** The system uses standard pallets such as the frame, staking, and balances pallets. They require extensive configuration and are critical for the correct execution of the node. We reviewed their configuration, paying particular attention to consensus parameters and parameters that can introduce unexpected or undesired economic incentives for validators and users in general.

- **Two custom pallets:**

    - **`aggregate`:** This pallet provides a mechanism for tracking and aggregating statements (i.e., proof verification submissions) from users. It is possible to define different aggregation sizes and thresholds for different domains. We checked whether operations are performed in a permissionless way, how their inputs are validated, and whether the system transitions between states according to the provided inline documentation.

    - **`hyperbridge_aggregations`:** This pallet provides a way to send aggregation data to another chain. We checked that the ISMP API is correctly used and the input data is correctly encoded.

    For both pallets, we reviewed their weights to check whether they are correctly set and reviewed their code for general Rust-specific issues such as arithmetic overflows, unexpected errors (or panics), and code prone to denial of service. We used static analysis tools such as Clippy and Semgrep to find common patterns associated with issues.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- Some parameters from the runtime were still under consideration so they were not audited in depth.

- No cryptographic code was reviewed during this engagement
- No testing nor deployment scripts were reviewed.
- Standard Substrate pallet code was not reviewed.

Summary of Findings

The table below summarizes the findings of the review, including details on type and severity.

ID	Title	Type	Severity
1	NewDomain event emitted before state changes are applied	Auditing and Logging	**Informational**
2	Vulnerable, unmaintained, or deprecated dependencies in the Substrate node	Patching	**Informational**

Detailed Findings

1. NewDomain event emitted before state changes are applied

Severity: **Informational**	Difficulty: **Low**
Type: Auditing and Logging	Finding ID: TOB-HRZ-1
Target: `pallets/aggregate/src/lib.rs`	

Description

In the `register_domain` function, the `NewDomain` event is emitted at the start of the function before the arguments are validated and state changes are made. This does not cause any problems, because if the transaction reverts the state changes also revert; nonetheless, the checks-effects-interactions pattern should be followed in this function, as is done in other functions of the codebase.

```
pub fn register_domain(
     origin: OriginFor<T>,
     aggregation_size: AggregationSize,
     queue_size: Option<u32>,
 ) -> DispatchResultWithPostInfo {
     ...
     Self::deposit_event(Event::NewDomain { id });
     ...
     let domain = Domain::<T>::try_create(
         id,
         owner.clone(),
         1,
         aggregation_size,
         queue_size,
         ticket,
     )?;
     Domains::<T>::insert(id, domain);
     NextDomainId::<T>::put(id + 1);


     Ok(owner.post_info(None))
 }
```

Figure 1.1: Snippet of the `register_domain` function (`lib.rs#L644–L680`)

Recommendations

Short term, move the emission of the event after the state changes are made.

Long term, try to always follow the checks-effects-interactions pattern; where it is not followed, add documentation explaining why it is not necessary and why the code is safe.

2. Vulnerable, unmaintained, or deprecated dependencies in the Substrate node

Severity: **Informational**	Difficulty: **High**
Type: Patching	Finding ID: TOB-HRZ-2
Target: Substrate node	

Description

The Substrate node codebase uses the following vulnerable or unmaintained Rust dependencies.

Dependency	Version	ID	Description
`idna`	0.2.3, 0.4.0	RUSTSEC-2024-0421	`idna` accepts Punycode labels that do not produce any non-ASCII when decoded
`rustls`	0.20.9	RUSTSEC-2024-0336	`rustls::ConnectionCommon::complete_io` could fall into an infinite loop based on network input
`rustls`	0.23.17	RUSTSEC-2024-0399	`rustls` network-reachable panic in `Acceptor::accept`
`atty`	0.2.14	RUSTSEC-2021-0145	Potential unaligned read
`crossbeam-utils`	0.7.2	RUSTSEC-2022-0041	Unsoundness of `AtomicCell<64>` arithmetics on 32-bit targets that support Atomic64
`memoryoffset`	0.5.6	RUSTSEC-2023-0045	`memoffset` allows reading uninitialized memory
`pprof`	0.12.1	RUSTSEC-2024-0408	Unsound usages of `std::slice::from_raw_parts`

`ansi_term`, `atty`, `derivative`, `instant`, `json`, `match`, `proc-macro-error`	N/A	N/A	All these crates are unmaintained
`parity-wasm`	0.4.19	RUSTSEC-2022-0061	Deprecated by author

Except for the unmaintained and deprecated crates, all these dependencies need to be updated to their newest versions to fix the vulnerabilities.

Recommendations

Short term, update all dependencies to their newest versions. Review the unmaintained and deprecated crates to look for alternatives.

Long term, integrate `cargo-audit` into the development process to detect issues with dependencies.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Non-Security-Related Recommendations

The following recommendations are not associated with specific vulnerabilities. However, implementing them may enhance code readability and prevent the introduction of vulnerabilities in the future.

- In the `reserve_currency_for_publication` function, replace the `saturating_add` function with `defensive_saturating_add`, which will provide better logging.

```
fn reserve_currency_for_publication(
     &self,
     account: &AccountOf<T>,
 ) -> Result<BalanceOf<T>, DispatchError> {
     let estimated =
estimate_publish_aggregation_fee::<T>(self.max_aggregation_size);
     let hold = (estimated.saturating_add(
         <T as
Config>::ComputePublisherTip::compute_tip(estimated).unwrap_or_default(),
     )) / self.next.size.into();
     T::Hold::hold(&HoldReason::Aggregation.into(), account, hold).map(|_| hold)
 }
```

Figure B.1: The `reserve_currency_for_publication` function (pallets/aggregate/src/lib.rs#L371–L380)

- Remove the assignment of `DomainState::Hold` to `domain.state`; it is unnecessary since it will be overwritten (possibly with the same `DomainState::Hold`) in the subsequent call to `update_hold_state`, which does not depend on the current state.

```
pub fn hold_domain(origin: OriginFor<T>, domain_id: u32) ->
DispatchResultWithPostInfo {
     let owner = User::<T::AccountId>::from_origin::<T>(origin)?;
     Domains::<T>::try_mutate_exists(domain_id, |domain| {
         match domain {
             Some(domain) if owner.can_handle_domain::<T>(domain) => {
                 if domain.state == DomainState::Ready {
                     domain.state = DomainState::Hold;
                     domain.update_hold_state();
                     domain.emit_state_changed_event();
     ...
```

Figure B.2: Snippet of the `hold_domain` function (pallets/aggregate/src/lib.rs#L703–L723)

- Fix the typo in the following log message; “founds” should be “funds.”

```
if remain > 0_u32.into() {
     log::warn!(
         "Cannot refund all founds from {account:?} to {origin:?}: missed
{remain:?}"
     )
 }
```

Figure B.3: Log message with a typo (pallets/aggregate/src/lib.rs#L617–L621)

- Properly document the `hyperbridge_aggregations` pallet; it does not provide any information about the data that is sent or its constraints.
- Remove the unused `AggregationData` structure.

```
struct AggregationData {
     aggregation_id: u64,
     aggregation: sp_core::H256,
 }
```

Figure B.4: The `AggregationData` structure (pallets/hyperbridge_aggregations/src/lib.rs#L108–L111)

C. Migration Recommendations

The following are recommendations for the Substrate node upgrade process. They are partially based on the Open Polkadot Bootcamp documentation on node migrations.

- Clearly document the version bumps needed to reach the target from the current code state (e.g., migration starts in 1.10.0 and goes to 1.15.0), including the migrations needed for each version (e.g., from 1.10 to 1.11, 1.11 to 1.12, and so on).
- Clearly document deprecated functions, behavior changes, and new features. If a new feature needs to be properly configured after a migration, make sure to add a "TODO/FIXME" note and keep a centralized document to track these tasks.
- Consider using tools such as the Polkadot SDK Version Manager and Zepter to manage Cargo upgrades.

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries and government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

Trail of Bits performed all activities associated with this project in accordance with a statement of work and an agreed-upon project plan.

Security assessment projects are time-boxed and often rely on information provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test software controls and security properties. These techniques augment our manual security review work, but each has its limitations. For example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. A project's time and resource constraints also limit their use.