Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Tendermintx

Veridise Inc.

April 5, 2024

► **Prepared For:**

Succinct
https://succinct.xyz/

► **Prepared By:**

Shankara Pailoor
Timothy Hoffman

► **Contact Us:** contact@veridise.com

► **Version History:**

Mar. 28, 2024 Initial Draft

Contents

1 Executive Summary

From Mar. 12, 2024 to Mar. 19, 2024, Succinct engaged Veridise to review the security of Tendermintx, their Tendermint ZK light client. The review covered their ZK circuits and smart contracts which serve to verify Tendermint headers according to the Tendermint light client protocols for skip and sequential verification. Veridise conducted the assessment over 2 person-week, with 2 engineers performing an extensive manual review of the code over 1 week on commits 30fa25a72 -a6b05bb97.

Code assessment. The Tendermintx developers provided the source code of the Tendermintx contracts and circuits for review which contained extensive documentation in the form of READMEs and comments within the code. To facilitate the Veridise auditors' understanding of the code, the Tendermintx developers provided a Notion document describing the intended behavior of the circuits along with specific types of bugs they wanted the auditors to be aware of. The source code contained a test suite, which the Veridise auditors noted included end-to-end functional tests along with unit tests. The codebase also was subject to two previous audits, both of which are publicly available. All issues from the previous audits had been resolved and, the Veridise auditors found the code to be of very high quality overall.

Summary of issues detected. The audit uncovered 4 issues, 1 of which are assessed to be of high or critical severity by the Veridise auditors. Specifically, V-TEND-VUL-001 was a logical error which allowed a malicious validator to forge headers. The Veridise auditors also identified 3 informational findings, and all the above issues were fixed immediately after disclosure.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve Tendermintx. While the code was generally well tested, the auditors felt that the testing could be improved by adding so-called "negative" tests which ensure that undesirable behavior is not permitted. Such tests are more important for ZK protocols as they can help detect underconstrained bugs if/when the circuit is refactored. The auditors also recommend that the Tendermintx developers add cargo-audit checks to their CI-workflow to ensure vulnerable libraries are not used.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Tendermintx	`30fa25a-a6b05bb`	Rust and Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Mar. 12 - Mar. 19, 2024	Manual	2	2 person-week

Table 2.3: Vulnerability Summary.

Name	Number	Fixed	Acknowledged
Critical-Severity Issues	1	1	1
High-Severity Issues	0	0	0
Medium-Severity Issues	0	0	0
Low-Severity Issues	0	0	0
Warning-Severity Issues	0	0	0
Informational-Severity Issues	3	3	3
TOTAL	4	4	4

Table 2.4: Category Breakdown.

Name	Number
Maintainability	3
Logic Error	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Tendermintx's smart contracts and ZK circuits. In our audit, we sought to answer high-level questions such as:

- Does Tendermintx obey the verification protocol for Tendermint light clients as specified in the original whitepaper?
- Do the circuits contain any under-constrained or over-constrained vulnerabilities which are common to ZK circuits?
- Does the Tendermintx smart contract include the required checks on the public inputs of the circuit?

Answering those questions required investigating several other low-level questions and attack scenarios some of which are listed below:

- Does the circuit prevent malicious validators from double voting to inflate their voting power?
- The circuit includes many merkle proof checks to validate the fields of the header. Are the proofs susceptible to second pre-image attacks? Moreover, are all the leaves of the proof properly constrained?
- Are the messages signed by the headers properly deserialized? This is especially important as the messages from Tendermint validators can be of varying length.
- Does skip verification ensure the untrusted header's distance from the trusted header is larger than 1 and within the maximum skip distance? Likewise, does the sequential header verification ensure that the untrusted header is the successor of the trusted one?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, we performed an extensive manual audit of the code base.

Scope. The scope of the audit included all code under `circuits/`, which contains the circuits to generate ZK proofs for `skip` and `sequential` verification as well as `contracts/src/TendermintX.sol`, which exposes methods to allow users to actually rotate the trusted header using ZK proofs.

Methodology. Veridise auditors reviewed the reports of previous audits for Tendermintx, inspected the provided tests, and read the Tendermintx documentation. They then began an extensive manual audit of the code. During the audit, the Veridise auditors regularly met with the Tendermintx developers over Zoom and communicated over Slack to ask questions about the code.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-TEND-VUL-001	A Trusted Validator Can Forge Headers in ...	Critical	Fixed
V-TEND-VUL-002	Outdated Comments/Variable Names	Info	Fixed
V-TEND-VUL-003	Missing Array-Out-of-Bounds Checks	Info	Fixed
V-TEND-VUL-004	Unused VALIDATOR_SET_MAX Parameter	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-TEND-VUL-001: A Trusted Validator Can Forge Headers During Skip Verification

Severity	Critical	**Commit**	30fa25a
Type	Logic Error	**Status**	Fixed
File(s)	`circuits/builder/verifier.rs`		
Location(s)	verify_skip, verify_trusted_validators		
Confirmed Fix At	a6b05bb		

The *skip verification* phase of the tendermint light client protocol allows an untrusted, non-adjacent header to be verified if:

1. The validators which signed the *untrusted header* comprise more than 1/3 of the total voting power of the *trusted header.*
2. The combined voting power of the validators that signed the untrusted header exceeds 2/3 of the voting power for that header.

The `tendermintx` circuit incorrectly checked (1) as it checked that the validators of the *trusted header* comprised more than 1/3 of the voting power of the *untrusted header*. This can be seen in the code snippet below:

```
fn verify_skip<const VALIDATOR_SET_SIZE_MAX: usize, const CHAIN_ID_SIZE_BYTES: usize
    >(
        &mut self,
        expected_chain_id_bytes: &[u8],
        skip_max: usize,
        trusted_block: U64Variable,
        trusted_header_hash: Bytes32Variable,
        target_block: U64Variable,
        skip: &VerifySkipVariable<VALIDATOR_SET_SIZE_MAX>,
    ) {
        // Verify the target block is non-sequential with the trusted block and
    within maximum
        // skip distance.
        self.verify_skip_distance(skip_max, &trusted_block, &target_block);

        // Verify the validators from the target block marked
    present_on_trusted_header
        // are present on the trusted header, and comprise at least 1/3 of the total
    voting power
        // on the target block.
        self.verify_trusted_validators(
            &skip.target_block_validators,
            skip.target_block_nb_validators,
            trusted_header_hash,
            &skip.trusted_header_validator_hash_proof,
            &skip.trusted_header_validator_hash_fields,
            skip.trusted_block_nb_validators,
        );
    }
```

As a consequence, a trusted validator could forge headers by marking themselves as a validator on the new header and setting their voting power to be more than 1/3 of the total voting power on the new header.

Impact The following exploit scenario is possible:

1. Alice is a trusted validator.
2. Alice observes that the TVL in the bridge using the `tendermintx` light client exceeds the amount they staked.
3. She forges a new header, creating new keys to impersonate the other validators, sets their voting power to +1/3 of the total voting power.
4. She generates a ZK proof, and gets the new header verified.

Recommendation We recommend updating the check for (1) to check that the validators of the new header comprise +1/3 of the voting power on the trusted header.

Developer Response The `tendermintx` developers implemented the fix in this PR: `https://github.com/succinctlabs/tendermintx/pull/73`.

4.1.2 V-TEND-VUL-002: Outdated Comments/Variable Names

Severity	Info	**Commit**	30fa25a
Type	Maintainability	**Status**	Fixed
File(s)	`circuits/builder/verify.rs`		
Location(s)	multiple		
Confirmed Fix At	2f51e5f		

There are a couple of locations where the variable names and surrounding comments should be updated to be consistent with the code:

1. In `verify_skip()`, the comment (on lines 540-541) indicates that the voting power of the trusted validators should be $\geq \frac{1}{3} * TOTAL\_VP$ whereas it should be strictly greater.

```
// Verify the validators from the target block marked present_on_trusted_header
// are present on the trusted header, and comprise at least 1/3 of the total voting
    power
```

Snippet 4.1: Excerpt from `verify_skip()`

2. In `verify_voting_threshold()` the variable name `gte_threshold` indicates that the result will be true if the value provided was greater than or equal to the threshold. However, the computation checks if it is strictly larger and the surrounding comments agree with that computation.

Impact Code readability.

Recommendation

1. The phrase "at least" should be changed to "more than"
2. The variable should be renamed `gt_threshold`

Developer Response The `tendermintx` developers implemented the fix in this PR: `https://github.com/succinctlabs/tendermintx/pull/72`.

4.1.3 V-TEND-VUL-003: Missing Array-Out-of-Bounds Checks

Severity	Info	**Commit**	30fa25a
Type	Maintainability	**Status**	Fixed
File(s)	`circuits/builder/verify.rs`		
Location(s)	is_voting_power_greater_than_threshold		
Confirmed Fix At	2f51e5f		

The function `is_voting_power_greater_than_threshold()` iterates over a `validator_voting_power` array and sums their voting power. The loop which accumulates the voting power uses the bound `VALIDATOR_SET_SIZE_MAX` but there is no check that the length of the arrays are equal to `VALIDATOR_SET_SIZE_MAX`. Based on current usage of the function, it can be inferred that `validator_voting_power.len() == in_group.len() == VALIDATOR_SET_SIZE_MAX` but that is not made explicit in this function.

```
fn is_voting_power_greater_than_threshold<const VALIDATOR_SET_SIZE_MAX: usize>(
        &mut self,
        validator_voting_power: &[U64Variable],
        in_group: &[BoolVariable],
        total_voting_power: &U64Variable,
        threshold_numerator: &U64Variable,
        threshold_denominator: &U64Variable,
    ) -> BoolVariable {
        let zero = self.constant::<U64Variable>(0);

        let mut accumulated_voting_power = self.constant::<U64Variable>(0);
        // Accumulate the voting power from the enabled validators.
        for i in 0..VALIDATOR_SET_SIZE_MAX {
            let select_voting_power = self.select(in_group[i], validator_voting_power
    [i], zero);
            accumulated_voting_power = self.add(accumulated_voting_power,
    select_voting_power);
        }

        let scaled_accumulated = self.mul(accumulated_voting_power, *
    threshold_denominator);
        let scaled_threshold = self.mul(*total_voting_power, *threshold_numerator);

        // Return accumulated_voting_power > total_vp * (threshold_numerator /
    threshold_denominator).
        self.gt(scaled_accumulated, scaled_threshold)
    }
```

Snippet 4.2: Definition of `is_voting_power_greater_than_threshold()`

Impact The current usage of this function implies an array-out-of-bounds error but those checks are located outside of this function rather than inside of it. It would be more clear to have those assertions checked inside of this function since this is the location where those equality properties are important. This can help to avoid/resolve issues that may occur if another usage of the function is added in the future.

Recommendation We recommend adding/moving the runtime assertion that the lengths of `validator_voting_power` and `in_group` are equal to `VALIDATOR_SET_MAX`.

Developer Response The `tendermintx` developers implemented the fix in this PR: `https://github.com/succinctlabs/tendermintx/pull/72`.

4.1.4 V-TEND-VUL-004: Unused VALIDATOR_SET_MAX Parameter

Severity	Info	**Commit**	30fa25a
Type	Maintainability	**Status**	Fixed
File(s)	`circuits/builder/verify.rs`		
Location(s)	get_total_voting_power		
Confirmed Fix At	2f51e5f		

In the function `get_total_voting_power()`, the template parameter `VALIDATOR_SET_SIZE_MAX` is not used. Based on current usage of the function, it can be inferred that `validator_voting_power.len() == VALIDATOR_SET_SIZE_MAX`. Similar blocks of code within the project use `VALIDATOR_SET_SIZE_MAX` as the loop iteration count instead of using `validator_voting_power.len()`.

```
fn get_total_voting_power<const VALIDATOR_SET_SIZE_MAX: usize>(
        &mut self,
        validator_voting_power: &[U64Variable],
        nb_enabled_validators: Variable,
    ) -> U64Variable {
        // Note: This can be made more efficient by implementing the add_many_u32
    gate in plonky2x.
        let zero = self.zero();
        let mut total = self.zero();

        let mut is_enabled = self._true();
        for i in 0..validator_voting_power.len() {
            let idx = self.constant::<Variable>(L::Field::from_canonical_usize(i));

            // If at_end, then the rest of the leaves (including this one) are
    disabled.
            let at_end = self.is_equal(idx, nb_enabled_validators);
            let not_at_end = self.not(at_end);
            is_enabled = self.and(not_at_end, is_enabled);

            // If enabled, add the voting power to the total.
            let val = self.select(is_enabled, validator_voting_power[i], zero);
            total = self.add(total, val)
        }
        total
    }
```

Snippet 4.3: Definition of `get_total_voting_power()`

Impact This code is inconsistent with the way similar blocks of code are written in the project. Furthermore, using the `len()` function instead of the `const` value may prevent the Rust compiler from performing some optimizations on the loop.

Recommendation We recommend changing the loop bound to be over `VALIDATOR_SET_SIZE_MAX` and adding the assertion `validator_voting_power.len() == VALIDATOR_SET_SIZE_MAX` to make the relationship between these values explicit within the `get_total_voting_power()` function.

Developer Response The `tendermintx` developers implemented the fix in this PR: `https://github.com/succinctlabs/tendermintx/pull/72`.