Audit Report

Namada Ethereum Bridge

v1.0

March 28, 2023

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Heliax AG to perform a security audit of the Namada Ethereum Bridge.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following GitHub repositories:

https://github.com/anoma/namada
The audit focussed on the code in the `apps/src/lib/node/ledger` directory and relevant type definitions in `shared/src/ledger/eth_bridge` and `shared/src/types`.
Commit hash: `57fc202ea33df5072c9286f5ccb339537b1be885`

https://github.com/anoma/ethereum-bridge
Commit hash: `fe93d2e95ddb193a759811a79c8464ad4d709c12`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Namada is a sovereign proof-of-stake blockchain, using Tendermint BFT consensus, that enables multi-asset private transfers for any native or non-native asset using a multi-asset shielded pool derived from the Sapling circuit. Namada is a fractal instance in the Anoma ecosystem. A custom Ethereum bridge is implemented to support transferring any fungible or non-fungible asset from Ethereum to Namada.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Ethereum events are not correctly confirmed	Critical	Resolved
2	Missing replay protection for signatures	Critical	Acknowledged
3	Bridging tokens from Ethereum to Namada with an invalid amount of receiver addresses will freeze escrowed funds	Major	Resolved
4	Oracle uses an unbounded queue which might lead to out-of-memory issues	Major	Resolved
5	Incorrect accounting of voting power for Ethereum events	Major	Resolved
6	The minimum number of block confirmations required to bridge ERC20 tokens from Ethereum to Namada is not enforced	Major	Resolved
7	Slashable offense for voting on an unconfirmed event is not accounted as misbehavior	Major	Acknowledged
8	Validator set update protocol transactions are not validated in the process proposal phase	Minor	Resolved
9	Unsafe use of `selfdestruct`	Minor	Resolved
10	The total size in bytes of the transactions in a block proposal can exceed the block byte size limit	Minor	Resolved
11	Smart contract hub can upgrade contracts to an already used contract address	Minor	Acknowledged
12	The minimum number of block confirmations required to bridge ERC20 tokens from Ethereum to Namada is not enforced	Minor	Resolved
13	Less than half of the available wrapped transactions from the mempool are potentially included in a proposal	Informational	Resolved
14	Include a check to ensure that a transfer is sent only if the `slashed_amount` is greater than 0	Informational	Acknowledged

15	Instance of potential integer overflow on arithmetic operations	Informational	Acknowledged
16	Typographical error	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Low-Medium	Some parts of the codebase have low-medium readability because of confusing variable names and missing code comments. Other parts of the codebase could be simplified.
Level of documentation	Low-Medium	High-level documentation was provided, while in-depth code-level documentation was not present.
Test coverage	Medium-High	-

Detailed Findings

1. Ethereum events are not correctly confirmed

Severity: Critical

Events emitted by the Ethereum smart contracts must reach a certain amount of block confirmations before validators vote to include them on Namada. An event is confirmed if the difference between the heights of the current block and the block containing the event is greater than or equal to the required amount of confirmations.

However, the `PendingEvent::is_confirmed` function in `apps/src/lib/node/ledger/ethereum_node/events.rs:206` considers a block confirmed if the required amount of `confirmations` is greater than or equal to the block height difference. Thus, if there are more block confirmations, the block is considered unconfirmed and discarded from further processing. This leads to frozen funds in case of the `TransferToNamada` event.

Recommendation

We recommend changing the implementation to `self.confirmations <= height.clone() - self.block_height.clone()` to correctly identify Ethereum events as confirmed.

Status: Resolved

2. Missing replay protection for signatures

Severity: Critical

Signatures of the current validators set are provided in the execution of the `upgradeContract` function in `contracts/contract/Governance.sol:42`, `upgradeBridgeContract` function in `contracts/contract/Governance.sol:60` and `addContract` function in `contracts/contract/Governance.sol:77`. In all three functions, `messageHash` gets calculated to validate the given signatures. There is no replay protection for these functions though, which implies that an attacker can use previously provided signatures to again upgrade/update the state of the contract given that the validator set during the signature creation time is the same as the attack time. For example, an attacker may replay a previous upgrade and then execute a message, which may lead to irrecoverable state inconsistencies in the contract.

Recommendation

We recommend implementing replay protection for messages, such as adding a nonce to the pre-image of `messageHash`.

Status: Acknowledged

The client has assessed that this is not an urgent issue and will be addressed at a later date, as proposals need to be voted on via Namada governance.

3. Bridging tokens from Ethereum to Namada with an invalid amount of receiver addresses will freeze escrowed funds

Severity: Major

ERC-20 tokens are transferred from Ethereum to Namada using the `transferToNamada` function in `contracts/Bridge.sol`. The appropriate Namada receiver addresses are provided with the function parameter `address[] calldata _tos`. After transferring the specified tokens to the `Bridge` contract for escrow, an event `TransferToNamada` is emitted and processed by Namada validators. However, if the amount of `_tos` receiver addresses does not match the length of the other parameters `_froms` and `_amounts`, the event is discarded. Escrowed ERC-20 tokens for this specific token transfer transaction are locked and the user can not withdraw funds.

Recommendation

We recommend validating the function parameters `_froms`, `_amounts`, and `_tos` to assert the same lengths of the arrays.

Status: Resolved

4. Oracle uses an unbounded queue which might lead to out-of-memory issues

Severity: Major

In `apps/src/lib/node/ledger/ethereum_node/oracle.rs:33` in the `Oracle` struct the `sender` attribute is of type `UnboundedSender`. Its unbounded nature might cause an out-of-memory termination of the process in case the reader side of the channel cannot consume events as fast as the producer creates them. There might be a possibility of an attack in which a large number of events on the Ethereum side are crafted with the intention of exhausting the memory of the node.

Recommendation

We recommend replacing `UnboundedSender` with a bounded alternative.

Status: Resolved

5. Incorrect accounting of voting power for Ethereum events

Severity: Major

In `apps/src/lib/node/ledger/shell/vote_extensions/ethereum_events.rs:186`, the voting power for events gets accounted for cumulatively rather than per event to check whether sufficient votes are provided for Ethereum events by Namada validators.

This allows malicious validators to successfully add invalid events. For example, an event crafted without actually locking funds in the Ethereum smart contract could be used to mint new funds on the Namada chain.

Recommendation

We recommend checking that the voting power of every seen Ethereum event is greater than ⅔ of the total votes.

Status: Resolved

6. The minimum number of block confirmations required to bridge ERC20 tokens from Ethereum to Namada is not enforced

Severity: Major

Transferring ERC20 tokens to Namada by using the `transferToERC` function in the `contracts/Bridge.sol` contract emits the `TransferToNamada` event. This event includes the number of block confirmations that must be reached on the Ethereum chain before validators vote to include them on Namada. This number is provided by the function parameter `uint256 confirmations` and must be at least the protocol-specified minimum number of confirmations, initially set to 100. However, this aforementioned invariant is not enforced in the `RawTransfersToNamada::decode` function in `apps/src/lib/node/ledger/ethereum_node/events.rs:288`. As a result, a user can circumvent the protocol-specified minimum number of confirmations, which might be exploited to double-spend funds.

Recommendation

We recommend validating the number of confirmations required for the `TransferToNamada` event to assert the required minimum.

Status: Resolved

7. Slashable offense for voting on an unconfirmed event is not accounted as misbehavior

Severity: Major

When applying slashing validators in `apps/src/lib/node/ledger/shell/mod.rs:525`, `EvidenceType` gets matched with the `SlashType`. Voting on unconfirmed events is considered a slashable offense according to the provided documentation, but it is not accounted for as one of the variants of `EvidenceType`. This allows validators to vote on unconfirmed events. This issue may be abused by malicious validators to double-spend funds.

Recommendation

We recommend adding the logic for slashing for the voting on unconfirmed events.

Status: Acknowledged

The client has assessed that it is not possible to determine if a validator has voted on an unconfirmed event, and will be adjusting the specs accordingly.

8. Validator set update protocol transactions are not validated in the process proposal phase

Severity: Minor

Transactions in newly proposed blocks are validated by all validator nodes in the `Shell::process_proposal` in `apps/src/lib/node/ledger/shell/process_proposal.rs` to ensure valid transactions. However, contrary to properly validating protocol transactions of type `ProtocolTxType::EthereumEvents`, transactions of type `ProtocolTxType::ValidatorSetUpdate` are not currently validated.

Recommendation

We recommend adding appropriate validation logic for transactions of the type `ProtocolTxType::ValidatorSetUpdate`.

Status: Resolved

9. Unsafe use of `selfdestruct`

Severity: Minor

Using `selfdestruct` in `contracts/contract/Bridge.sol:160` is not safe because ERC20 funds would be lost if those funds were not withdrawn before executing the `selfdestruct` function. There is no inherent protection that makes sure all funds are withdrawn from the contract before `selfdestruct` so there is a risk of human error of misplacing an ERC20 token address within the `_tokens` param of `withdraw` function.

Recommendation

We recommend removing the `selfdestruct` function and using a `reclaimETH` function which explicitly withdraws all native funds and transfers them to the `_to` address.

Status: Resolved

10. The total size in bytes of the transactions in a block proposal can exceed the block byte size limit

Severity: Minor

Tendermint expects the application to check whether the size of all transactions exceeds the byte limit `RequestPrepareProposal.max_tx_bytes`, which by default is set to `1048576` bytes. If so, the application must remove transactions at the end of the list until the total byte size is at or below the limit. However, the current implementation of `Shell::prepare_proposal` in `apps/src/lib/node/ledger/shell/prepare_proposal.rs:48-57` does not enforce this limit. Therefore, if the transaction byte size limit surpasses the limit, Tendermint will reject the block.

For more details, see requirement number 2 in Tendermints' formal requirements.

Recommendation

We recommend limiting the transactions within a proposed block accordingly to prevent surpassing the byte limit `RequestPrepareProposal.max_tx_bytes`.

Status: Resolved

11. Smart contract hub can upgrade contracts to an already used contract address

Severity: Minor

The `contracts/Hub.sol` contract holds references to the addresses of the latest contract version. Contracts are upgraded to a newer address with the `upgradeContract` function.

However, the current implementation does not verify if the `Hub` contract already uses the new contract address. Even though there is no immediate security implication, upgrading to an already used contract address should be prevented, similar to how it is implemented in the `addContract` function.

Recommendation

We recommend checking for duplicate addresses and preventing such contract upgrades.

Status: Acknowledged

12. The minimum number of confirmations needed to trust an Ethereum branch should be set to 100

Severity: Minor

According to the specs, `TransferToNamada` events may include a custom minimum number of confirmations, which must be at least the protocol-specified minimum number of confirmations but is initially set to `100`. However, `MIN_CONFIRMATIONS` in `apps/src/lib/node/ledger/ethereum_node/oracle.rs:19` is currently set to `50`, deviating from the specs.

Recommendation

We recommend changing the value of `MIN_CONFIRMATIONS` to `100` to comply with the specs.

Status: Resolved

13. Less than half of the available wrapped transactions from the mempool are potentially included in a proposal

Severity: Informational

A block proposer anticipates including half of the available wrapped transactions from the mempool in a new block proposal. However, the `Shell::build_mempool_txs` function in `apps/src/lib/node/ledger/shell/prepare_proposal.rs:136-148` first takes half of the transactions in `txs: Vec<Vec<u8>>` and then filters the transactions to only include `TxType::Wrapper` transactions. In case there are non-wrapper transactions in this batch of taken transactions, the block will include less than half of the anticipated wrapped transactions from the mempool.

Recommendation

We recommend filtering `TxType::Wrapper` transactions first and then taking half of the transactions to include in the block proposal.

Status: Resolved

14. Include a check to ensure that a transfer is sent only if the `slashed_amount` is greater than 0

Severity: Informational

In `apps/proof_of_stake/src/lib.rs:881` the `transfer` method is called regardless of the amount being slashed.

Recommendation

We recommend checking if the `slashed_amount` is greater than 0 before calling `transfer` in `apps/proof_of_stake/src/lib.rs:881`.

Status: Acknowledged

15. Instance of potential integer overflow on arithmetic operations

Severity: Informational

When calculating the `validator_voting_power` in `apps/src/lib/node/ledger/shell/vote_extensions/ethereum_events.rs:186` there is no check for arithmetic overflow. In the event of an overflow, events that should be part of `ethereum_events::VextDigest` will otherwise be ignored. However, this event seems unlikely to ever happen, since the total voting power is greater than the sum of the validator voting powers.

Recommendation

We recommend adding checks for arithmetic overflow in the `AddAssign` implementation to prevent `FractionalVotingPower` from overflowing.

Status: Acknowledged

The client mentioned that this specific code path is not active anymore.

16. Typographical error

Severity: Informational

In `shared/src/ledger/gas.rs:13`, the enum variant `TransactionGasExceededededError` has a spelling mistake.

Recommendation

We recommend replacing the variant with `TransactionGasExceededError`.

Status: Resolved