Three Sigma Labs

Code Audit

Trestle Protocol

Ethereum's Gateway to Celestia

Disclaimer

Code Audit

Trestle Protocol Ethereum's Gateway to Celestia

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Trestle Protocol Ethereum's Gateway to Celestia

Table of Contents

Summary

Code Audit

Trestle Protocol Ethereum's Gateway to Celestia

Summary

Three Sigma Labs audited Trestle Protocol in a 3 days engagement. The audit was conducted from 7/2/2024 to 9/2/2024.

Protocol Description

Trestle's mission is to connect the power of Celestia's modular blockchain network with the decentralized protocols and user base of Ethereum.

Trestle not only serves as a seamless bridge between the two chains, but also brings Wrapped Tia (wTIA) to Ethereum, creating a frictionless trading and development experience for a large audience of users.

Scope

Code Audit

Trestle Protocol Ethereum's Gateway to Celestia

Scope

wTIA

Trestle

Assumptions

OpenZeppelin is considered safe.

Methodology

Code Audit

Trestle Protocol Ethereum's Gateway to Celestia

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Trestle Protocol Ethereum's Gateway to Celestia

Project Dashboard

Application Summary

Name	Trestle Protocol
Commit	ab48e180b65275c71b7a6e778a4c80072c4e72e4
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	7/2/2024 to 9/2/2024
№ of Auditors	2
Review Time	3 days

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	1	1	0
Medium	7	5	2
Low	5	1	4

None	1	0	1

Category Breakdown

Suggestion	3
Documentation	0
Bug	10
Optimization	1
Good Code Practices	0

Code Maturity Evaluation

Code Audit

Trestle Protocol Ethereum's Gateway to Celestia

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. All functions had proper access control.
Arithmetic	Satisfactory. No rounding errors were found.
Centralization	Weak. The owner of wTIA can freely mint/burn tokens.
Code Stability	Satisfactory. Code was deployed.
Upgradeability	Weak. The contracts are not upgradeable.
Function Composition	Satisfactory. Functionality was well split into helpers.
Front-Running	Moderate. Frontrunning issues were found.
Monitoring	Moderate. Several events were missing
Specification	Satisfactory. Most of the code followed the specification.
Testing and Verification	Moderate. Some bugs should have been found by tests.

Findings

Code Audit

Trestle Protocol Ethereum's Gateway to Celestia

Findings

3S-TR-H01

Distribute is permissionless, allowing malicious users to specify 0 slippage and sandwich the swap

Id	3S-TR-H01
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Addressed by disabling taxes.

Description

`distribute()` uses the argument amountOutMinimum as slippage control, but the function is permissionless, so malicious users can trigger swaps with 0 minimum amount out.

Recommendation

Set up a keeper role to distribute rewards.

3S-TR-M01

Stuck tokens due to not using SafeTransfer

Id	3S-TR-M01
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug
Status	Acknowledged.

Description

reclaimToken() does not use SafeERC20.safeTransfer(), which could lead to stuck tokens.

3S-TR-M02

Swaps should use the deadline argument on top of **`minimumAmountOut`**

Id	3S-TR-M02
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Status	Addressed by disabling taxes.

Description

Swapping with block.timestamp as **`deadline`** means validators may do MEV up to the **`minimumAmountOut`**.

https://github.com/TrestleProtocol/Audit-Contracts/blob/main/src/Trestle.sol#L441

https://github.com/TrestleProtocol/Audit-Contracts/blob/main/src/Trestle.sol#L709

https://github.com/TrestleProtocol/Audit-Contracts/blob/main/src/Trestle.sol#L727

3S-TR-M03

On tax sells, swaps are vulnerable to MEV

Id	3S-TR-M03
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Status	Addressed by disabling taxes.

Description

Tax sells specify a **`minimumAmountOut`** of 0, which is vulnerable to MEV. Keep in mind that this is usually a design choice of tax tokens to avoid other issues.

3S-TR-M04

`createAmmPairWith()` in `initialize()` will revert if the pair already exists

Id	3S-TR-M04
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Protocol already deployed.

Description

createAmmPairWith() in `initialize()` reverts if the pair exists, which means that the contract could be DoSed.

Recommendation

Check if the pair already exists and skip creating if it does. However, this would make addLiquidityETH() vulnerable to slippage, so introduce slippage control arguments.

3S-TR-M05

Amounts to distribute are added directly to **`tokenPerShare`**, skipping using **`rewardPerToken`**

Id	3S-TR-M05
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed by disabling taxes.

Description

Rewards are emitted at a rate of **`rewardTokenPerBlock`**. However, when distributing, the received reward tokens are added directly to **`rewardPerToken`**, instead of being left in the contract and letting **`rewardTokenPerBlock`** distribute the amounts over time.

Recommendation

Don't add the amount to swap to **`amountToDistribute`**, but let the tokens in the contract so rewards can be emitted according to **`rewardTokenPerBlock`**. In fact, the swap should be performed before calculating **`emittedRewards()`**, so it does not happen that the tokens to distribute are cropped due to not having enough balance and then due to the swap there is now balance.

3S-TR-M06

Possible to mint or burn infinite shares, stealing all rewards or making transfers fail

Id	3S-TR-M06
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug
Status	Infinite shares addressed by disabling taxes. 0 shares is acknowledged.

Description

Increasing shares infinitely allows an attacker to steal all rewards. It's possible by doing:

1. In _isExcludedFromRewards(), the condition **`addr.code.length > 0 && !isOptin[addr]`** will return false if the contract is in the constructor, enabling - 1. transfer Trestle no a new contract, 2. after the constructor, move the funds to another address, which won't reduce shares as the contract has not opted in.

2. Calling **`rewardOptIn()`** with a blacklisted address will increase shares infinitely as **`_isExcludedFromRewards()`** will always return true, increasing shares every time (**not possible** anymore as addresses can not be blacklisted post dead block period).

3. Removing from the blacklist a smart contract that has not opted in will keep increasing shares every time, as **`_isExcludedFromRewards()`** will always return true.

Decreasing shares to 0 will mean reward eligible accounts (EOAs or smart contracts that have opted in) with previous non 0 balance will always revert when transferring shares (either **`from`** or **`to`**). It's possible to do this in the 2 following aways:

1. **`rewardOptout()`** will keep decreasing shares if called by an EOA or in the constructor of a smart contract, as **`_isExcludedFromRewards()`** will always return false.

2. **`_addToBlacklist()`** will keep decreasing shares if **`addr`** is an EOA or a smart contract that has opted in, as will always return false (not possible anymore as the dead block period is over).

3S-TR-M07

`address(0)` can be added as a pair, triggering taxes on burn

Id	3S-TR-M07
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Status	Addressed by disabling taxes.

Description

A pair with zero address may be added to the **`pairs`** mapping which means burns will be taxed.

Recommendation

Revert if the pair does not exist in **`recordAmmPairWith()`**.

3S-TR-L01

`totalShares` update should add before reducing to avoid underflows

Id	3S-TR-L01
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Acknowledged.

Description

`totalShares` update subtracts before summing, which could lead to underflow in edge cases (when something unexpected happens).

3S-TR-L02

Users may send tokens before the dead block period to an address to blacklist it

Id	3S-TR-L02
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Dead block period is over.

Description

During the dead block period, before trading has started, users may send tokens to an address to maliciously blacklist it.

Recommendation

The admin can remove addresses from the blacklist, but still, the dead block approach to blacklist could be thought over.

3S-TR-L03

Native transfers should use Openzeppelin's **`.sendValue()`**

Id	3S-TR-L03
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Acknowledged.

Description

Paying fees to the marketing operator uses .transfer(), which only forwards 2300 gas, which could lead to reverts if the operator has any logic in the **`fallback`** or **`receive`** functions.

Recommendation

Use .sendValue() from Openzeppelin's **`Address.sol`**.

3S-TR-L04

Missing events

Id	3S-TR-L04
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Acknowledged.

Occurrences

https://github.com/TrestleProtocol/Audit-Contracts/blob/main/src/Trestle.sol#L222

https://github.com/TrestleProtocol/Audit-Contracts/blob/main/src/Trestle.sol#L400-L402

https://github.com/TrestleProtocol/Audit-Contracts/blob/main/src/Trestle.sol#L469-L470

https://github.com/TrestleProtocol/Audit-Contracts/blob/main/src/Trestle.sol#L478

https://github.com/TrestleProtocol/Audit-Contracts/blob/main/src/Trestle.sol#L613-L614

https://github.com/TrestleProtocol/Audit-Contracts/blob/main/src/Trestle.sol#L613-L614

3S-TR-L05

`Ownable2Step` is preferred over `Ownable`

Id	3S-TR-L05
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Acknowledged.

Description

Openzeppelin Ownable2Step changes **`transferOwnership()`** to a 2 step procedure, which means that to change an owner, first a pending owner is set and only then this account must accept ownership. This means that it's impossible to set the wrong address as the owner by mistake.

Recommendation

Use **`Ownable2Step`**.

3S-TR-N01

Storage variables can be cached

Id	3S-TR-N01
Classification	None
Category	Optimization
Status	Acknowledged.

Occurrences

https://github.com/TrestleProtocol/Audit-Contracts/blob/main/src/Trestle.sol#L185-L186

https://github.com/TrestleProtocol/Audit-Contracts/blob/main/src/wTIA.sol#L48-L50

https://github.com/TrestleProtocol/Audit-Contracts/blob/main/src/wTIA.sol#L75