PUBLIC

Code Assessment

of the Threshold Network Smart Contracts

November 09, 2021

Produced for

by

Contents

1 Executive Summary

Dear Sir or Madam,

First and foremost we would like to thank Threshold Network for giving us the opportunity to assess the current state of their Threshold Network system. This document outlines the findings, limitations, and methodology of our assessment.

The Threshold team was always responsive and professional. We did not uncover critical issues. Two medium issues were found. The remaining issues are low severity issues. All issues have been fixed or acknowledged.

We hope that this assessment provides more insight into the current implementation and provides valuable findings. We are happy to receive questions and feedback to improve our service and are highly committed to further support your project.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	2
• Code Corrected	1
• Risk Accepted	1
Low-Severity Findings	8
• Code Corrected	5
• Specification Changed	1
• Risk Accepted	1
• Acknowledged	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Threshold Network repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	14 October 2021	913c04debc1b5a733ea402bf5dacd5e85da0db8a	Initial Version Token and VendingMachine
2	20 October 2021	aeba9733038e0c36d3572f7bbe6e68a8302c10c5	Initial Version Staking
3	09 November 2021	024b98cc657bec19289676223d5d69a481e17525	Third Version

For the solidity smart contracts (Version 1), the compiler version `0.8.4` was chosen. The compiler was updated in (Version 3) to `0.8.9`.

Following files from repository contracts folder were part of the assessment scope:

Token and vending machine :

```
/governance/Checkpoints.sol
/token/T.sol
/vending/VendingMachine.sol
```

Staking :

```
/staking/IApplication.sol
/staking/IStaking.sol
/staking/StakingProviders.sol
/staking/TokenStaking.sol
/utils/PercentUtils.sol
```

2.1.1 Excluded from scope

Any contracts not mentioned above. Mock and testing contract that might rely on scope contracts. Imported libraries are assumed to behave according to their specification and are not part of the assessment scope. The connected contract like the legacy staking contracts are considered to behave correctly and are assumed to be safe. The Nu staking contract is unfinished and WIP. We do not know the final implementation. Interactions with this contract might be vulnerable and, hence, could not be fully audited. We also do not know the connected applications and assume all applications as trusted and the interaction as correct and secure.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section we have added a version icon to each of the findings to increase the readability of the report.

The planned merge of Keep Network (KEEP token) and NuCypher (NU token), will form a new network called Threshold network with a new native token called T. The contracts under review offer KEEP and NU holders to wrap and unwarp their tokens into the newly created T tokens and a staking contract to stake them but simultaneously supports the legacy staking contracts.

2.2.1 T token

T is an ERC20 token. On top of the usual ERC20 properties there are :

- Delegation : holders can delegate their tokens to a delegatee
- Checkpoints :
    1. records delegations weights for each delegatee, new checkpoint at each delegation and token transfer
    2. records state of T's total supply, new checkpoint at each mint or burn
- EIP-2612 Permit : allows holders to approve a spender for an amount by signing a message with a validity deadline
- Misfund recovery : owner of T's contract can transfer back ERC20 and ERC721 tokens that have been mistakenly sent to the contract

2.2.2 Vending Machine

This contract wraps KEEP and NU tokens to T, and unwraps T into KEEP and NU. The conversion is done with a fixed and immutable rate.

One contract will be deployed for each one of the tokens. The exchange rate will be the ratio between the allocated amount of T tokens assigned to the wrapped token and the supply amount of token to be wrapped. Plan is to equally split the supply of T between KEEP and NU token holder.

2.2.3 Staking contract

Threshold Network also offers a staking contract that allows KEEP, NU and T tokens to be staked. Although T staking is simply the transfer of tokens to the contract, legacy KEEP and NU staking is more sophisticated. The staker can use the stake on the legacy staking contract of his choice and, sync the new staking contract with it. The legacy stake will be accounted as it is staked in T, so that all staked tokens have their amounts accounted in T.

A staker needs to define the following roles to stake :

- owner : the original staker/delegator, owner of the tokens
- operator : address approved by the owner to manage its stake, each operator can have at most one owner
- authorizer : address approved by the owner to increase and decrease the stake amount allocated to each application
- beneficiary : address where rewards can be paid

The contract's logic can be split in five subsections :

- Stake delegation : owner can either start staking with KEEP, NU or T. Operator, authorizer and beneficiary are set at this point. If owner stakes T, a simple transfer to the contract will occur. If owner stakes KEEP or NU, the staking contract will query the legacy staking contracts for the staked amount and will account it in converted T amount.
- Stake top up : once owner did start staking on this contract, he can stake more. This can be done in KEEP, NU or T. If owner tops up in T, a token transfer will occur. If owner wants to top up with KEEP or NU, he must top up in the legacy staking contracts in the first place, and then call top up on this contract to sync the amounts. The operator can also sync KEEP and NU amounts for the owner.
- Authorize application : a set of applications will be authorized access to the staking contract, they will be chosen by governance. An authorizer can allow and disallow stake amounts to different applications. The stake amount is shared across applications. Applications can also slash misbehaving operators, by doing so, a slashing event will be added to a queue. There are two ways for an application to ask for slashing, either by calling `seize`, or by calling `slash`. On the first one, a notifier can be specified and will receive `notificationReward * _rewardMultiplier%` amount of T per notified misbehaving operator as a reward. `notificationReward` can be tuned by governance and `_rewardMultiplier` is given by the calling application. There is an incentive for processing the queue of pending slashes since the processor will receive a reward of 5% of the total slashed amount. The tokens will be slashed in this order : T, KEEP, NU.
- Undelegating stake : owner and its operator can lower their stake from the contract. Upon undelegation, the contract will check that the remaining stake amount can cover the maximum amount allowed to an application. If the check succeeds, T tokens will be transferred back to their owner, KEEP and NU amounts will simply get removed from the accounting of this contract but stay in legacy staking contracts.
- Incentive to synchronization : to mitigate stake discrepancies, i.e. this contract accounts more legacy tokens than legacy staking contracts have, the system relies on notifiers. Their role is to watch for such discrepancies and notify the staking contract, which will validate the discrepancy, punish the owner and reward the notifier. The `stakeDiscrepancyPenalty` is a fixed amount that can be changed by governance, the notifier's reward is `5% * stakeDiscrepancyRewardMultiplier% * stakeDiscrepancyPenalty`, where `stakeDiscrepancyRewardMultiplier` is also a parameter adjustable by governance.

The stake contract has three privileged roles:

1. The owner of the stake - can stake, allocate the stake and top up
2. The Threshold governance - can call administrative setters
3. A panic account per application - can disable an application

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts associated with the items defined in the engagement letter on whether it is used in accordance with its specifications by the user meeting the criteria predefined in the business specification. We draw attention to the fact that due to inherent limitations in any software development process and software product an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third party infrastructure which adds further inherent risks as we rely on the correct execution of the included third party technology stack itself. Report readers should also take into account the facts that over the life cycle of any software product changes to the product itself or to its environment, in which it is operated, can have an impact leading to operational behaviours other than initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severities. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved, have been moved to the Resolved Findings section. All of the findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	1

- Ineffective Try Catch Statement Risk Accepted

Low-Severity Findings	2

- Missing Sanity Checks Acknowledged
- Possibly Uninitialized Penalties Risk Accepted

5.1 Ineffective Try Catch Statement

Design Medium Version 1 Risk Accepted

Try catch statements should handle critical code parts that might fail and their respective exertions correctly. The used try catch statement in `authorizationDecrease` simple fails silently if not successful. Resulting in potential incorrect authorization decrease.

Risk accepted :

Threshold Network accepts that a decrease fails silently. The event `AuthorizationInvoluntaryDecreased` has been added to track involuntary decreases, it contains a field to indicate whether the call to the application succeeded or not.

5.2 Missing Sanity Checks

Design Low Version 1 Acknowledged

For security reasons stakers use different roles to manage the stake. If different roles exist, it seems consistent to enforce the use of different keys. `stake` and `stakeNu` do not check if the addresses (operator, beneficiary, owner, authorizer) are the same. In a more limited way this is also the case for `stakeKeep`.

Acknowledged :

Threshold Network does not consider that roles having different addresses must be enforced. In their modelling, they always assumed that some stakers will reuse addresses for different roles.

5.3 Possibly Uninitialized Penalties

Design | Low | Version 1 | Risk Accepted

The constructor function initializes important variables for the staking contract. However, `takeDiscrepancyPenalty` and `stakeDiscrepancyRewardMultiplier` are not initialized and need to be set separately in `setStakeDiscrepancyPenalty`. The `onlyGovernance` modifier ensures that only the community controlled governance contract can call this function. Calls from community driven governance contracts usually have a long reaction time due to voting and other collective decisions that need to be taken before. Hence, the variables might be uninitialized and result in no penalties for misbehaving.

Risk accepted :

These parameters need to be set by governance, Threshold Network believes that in the interim, zero penalty is an acceptable behavior.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	1

- Unauthorized Top Ups Code Corrected

Low-Severity Findings	6

- Compiler Version Not Fixed and Outdated Code Corrected
- Inefficient Struct Packing Code Corrected
- Inefficient processSlashing Loop Code Corrected
- Interface File Name Convention Code Corrected
- Misleading Variable count Code Corrected
- Specifications Mismatch Specification Changed

6.1 Unauthorized Top Ups

Design Medium Version 1 Code Corrected

With the current design, anybody can call topUp on any operator (in Kepp, Nu and T). This could lead to KEEP or NU staked in legacy contract, that the owner may not want to be staked on the new staking contract, ending staked on the new contract.

On Nu this might lead to trolling by calling `topUpNu` after a user send an `unstakeNu` transaction and blocking the Nu withdraw through:

1. X calls `unstakeNu`
2. Y calls `topUpNu` on X
3. X tries to withdraw from NU legacy staking contract, but it fails because there is still an amount of NU accounted in the new staking contract

With Keep the issue is more severe as non-malicious behavior could be slashed with a sandwich attack like follows:

1. Someone wants to unstake keep and calls "unstakeKeep"
2. The user sends the tx for the Keep legacy contract to "undelegate"
3. This tx lands in the men pool and someone front runs it by calling "topUpKeep"
4. The undelegate is mined after the top up
5. The attacker calls the notify keep discrepancy function to slash

Code corrected :

A modifier has been added to all three top up functions to restrict the access only to owner and operator.

6.2 Compiler Version Not Fixed and Outdated

Design | Low | Version 1 | Code Corrected

The solidity compiler is not fixed in the `Checkpoints.sol`. The version, however, is defined in the `hardhat.config.js` to be `0.8.4`.

In the code the following pragma directives are used:

```
pragma solidity ^0.8.0;
```

Known bugs in version `0.8.4` are:

https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json#L1562

More information about these bugs can be found here:

https://docs.soliditylang.org/en/latest/bugs.html

At the time of writing the most recent Solidity release is version `0.8.9` which contains some bugfixes.

Code corrected :

Compiler version is now `0.8.9` and fixed all files.

6.3 Inefficient Struct Packing

Design | Low | Version 1 | Code Corrected

The variable order inside structs is not optimized by the compiler. Hence, tight variable packing needs to be done manually. The struct `OperatorInfo` could be packed differently, to save two storage slots.

Code corrected :

Struct has been optimized.

6.4 Inefficient `processSlashing` Loop

Design | Low | Version 1 | Code Corrected

State operations are expensive. Additionally, Threshold Network told that the `processSlashing` is gas critical. The function `processSlashing` reads and writes the state variable `slashingQueueIndex` multiple times. The loop even does operations in each iteration.

Additionally, a sanity check for `count` parameter in `processSlashing` instead of a check at every iteration of the for loop could save gas.

Code corrected :

`slashingQueueIndex` is now updated only once after the for loop with an internal counter in memory.

The stopping condition of the for loop has been optimized, maxIndex is now capped at max queue's length and an event is emitted with the effective number of slashes.

6.5 Interface File Name Convention

Design Low Version 1 Code Corrected

The file `StakingProviders.sol` is an interface definition. To be consistent with the naming, the file should be renamed with a leading `I`.

Code corrected :

Filename updated.

6.6 Misleading Variable `count`

Design Low Version 1 Code Corrected

One of the stop conditions of the for loop in `processSlashing` allows the function to process one more pending slash than initially intended. `slashingQueueIndex <= maxIndex` should be `slashingQueueIndex < maxIndex` if it should match the passed in `count` argument. One more unintended iteration also would cost the processor more gas than they may have wanted to spend in the first place.

Code corrected :

Loop's stopping condition has been updated.

6.7 Specifications Mismatch

Design Low Version 1 Specification Changed

Specs of `processSlashing` say that processor can get either 4% or 5% of the slashed amount, depending on the type of call the application did, but in practice processor always gets 5%, no matter the application called `seize` or `slash`.

Version 3 `getStartTStakingTimestamp` specs say that result is zero when operator has no stake or when they was topped-up, but top up functions do not update the staking timestamp

---

Specification partially changed :

For both mentioned issues the specifications were updated accordingly.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Applications Share the Same Stake

Note Version 1

Threshold Network informed that the same stake can be used in different applications. Sharing the same stake practically means that if one application slashes or seizes all stake, all other applications that shared the stake will have no stake left to seize or slash.