SMART CONTRACT AUDIT

zokyo

March 9th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

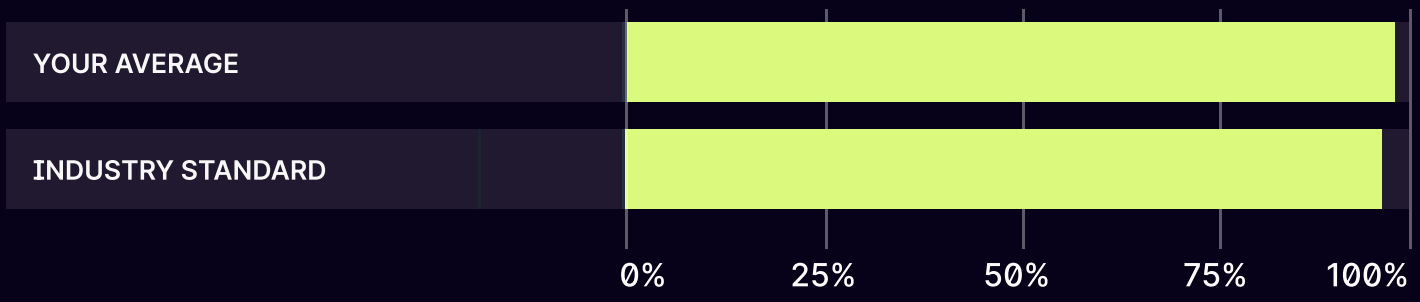
TECHNICAL SUMMARY

This document outlines the overall security of the TruFin smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the TruFin smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



97% of the code is testable, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the TruFin team put in place a bug bounty program to encourage further active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the TruFin repository:
https://github.com/TruFin-io/staker-audit

Branch: master
Initial commit: 71e300d31ad068f4118752f5f94cc571b086f6f1
Final commit: b2e4f7e882dcbac970bd9e63369f7147bb88813b

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- StakerStorage.sol
- Staker.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of TruFin smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Hardhat testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	03	Testing contract logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contracts by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

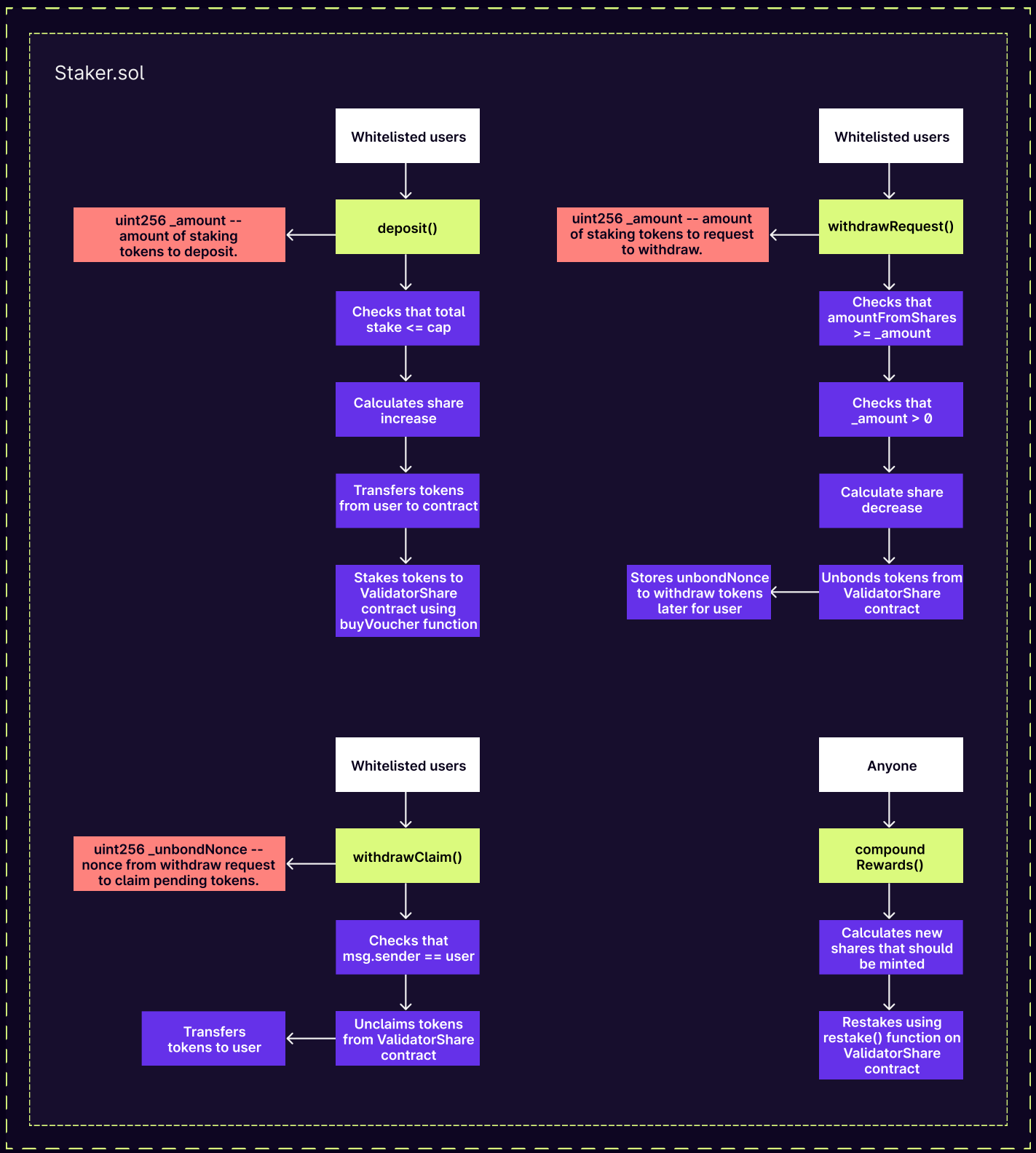
Zokyo Security received a set of contracts that represents a Trufin staker protocol. There were two contracts within the audit scope: Staker.sol and StakerStorage.sol. Staking allows users to deposit their Matic tokens on Ethereum mainnet and receive shares in exchange. Matic is then invested into Polygon PoS Staking Contract, allowing staking contract to earn rewards which are then distributed among the users of the protocol. Thus auditors needed to not only analyze smart contracts against the list of common vulnerabilities, gas optimization and detect any possible issues with the contract but also validate that Staker.sol interacts with Polygon Staking in a correct and safe way.

During the manual audit, auditors have found several informational and low issues, as well as one medium issue. The medium issue described the presence of a whitelist in the smart contract. In case users had deposited Matic tokens while they were whitelisted, they wouldn't be able to withdraw them if users were removed from whitelist. Trufin team has verified that such functionality is necessary in case users are put on the sanction list they shouldn't be able to interact with the protocol any longer. Other issues were connected to lack of parameters validation, lack of documentation and usage of custom errors. Trufin team has successfully fixed all of them. Also, one of the informational issues was originally marked as high and was connected to the unavailability of deploying a staking smart contract with any other token but Matic due to the design of stakeClaimedRewards() function, which performs a transfer of zero Matic from zero address. Since most of the ERC-20 tokens forbid direct transfers from zero address, such approach won't work with them. However it works fine in the current implementation where ERC-20 Matic on Ethereum mainnet is used. The issue was marked as info later after Trufin team has verified that only Matic would be used.

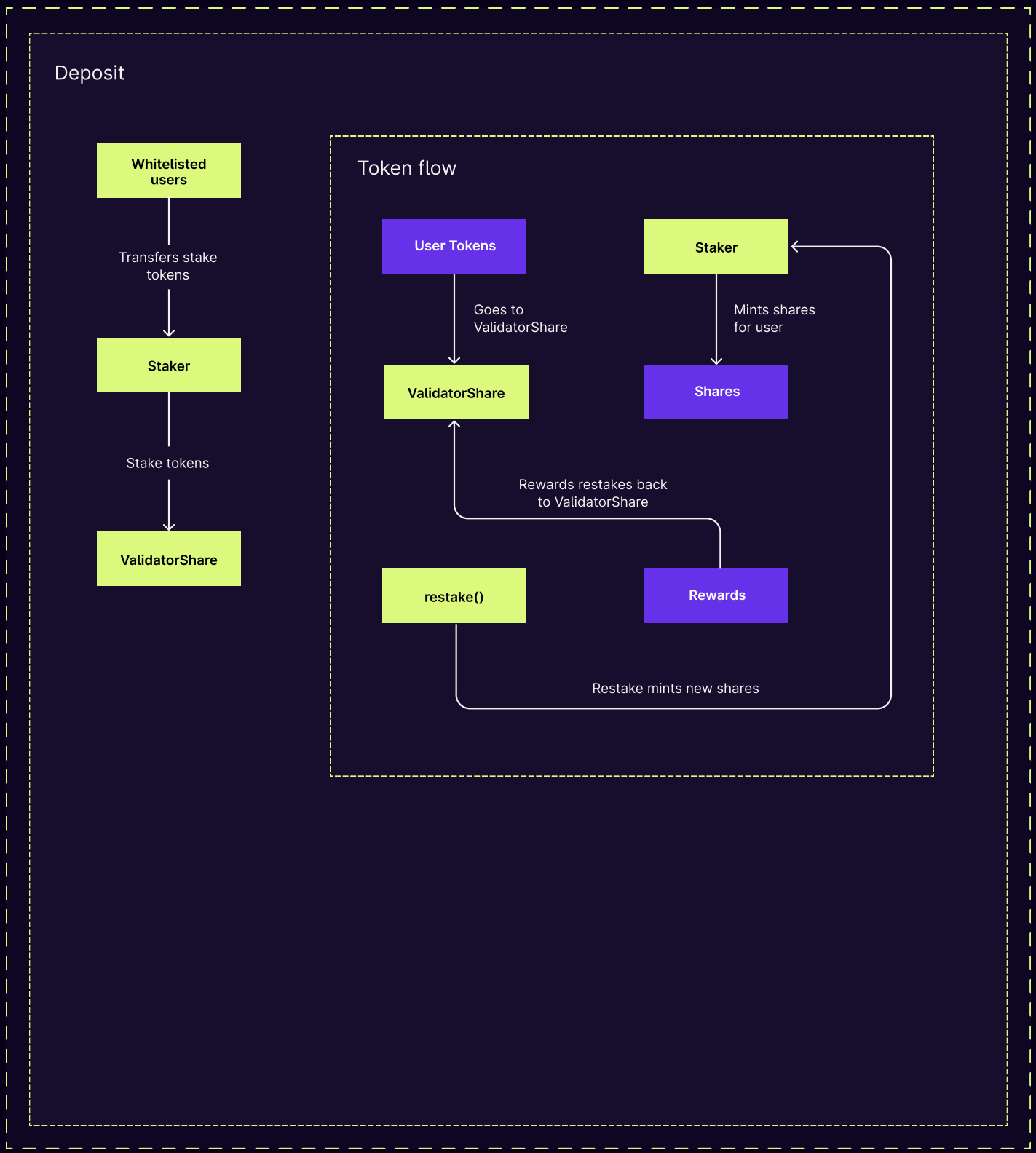
Zokyo Security team has also prepared a set of unit tests on Ethereum mainnet-fork in order to validate that staker smart contract interacts with Polygon staking correctly. Besides testing a common flow of the protocol such as deposits and withdrawals, auditors have also validated the interaction with 3rd party contract on mainnet-fork, calculation of shares price, restaking of rewards and withdrawal of profit.

The overall security of smart contracts is high enough. Contracts are well-written and tested both by Trufin team and Zokyo Security team. During the audit Trufin team has also added sufficient documentation that describes all the aspects of smart contract.

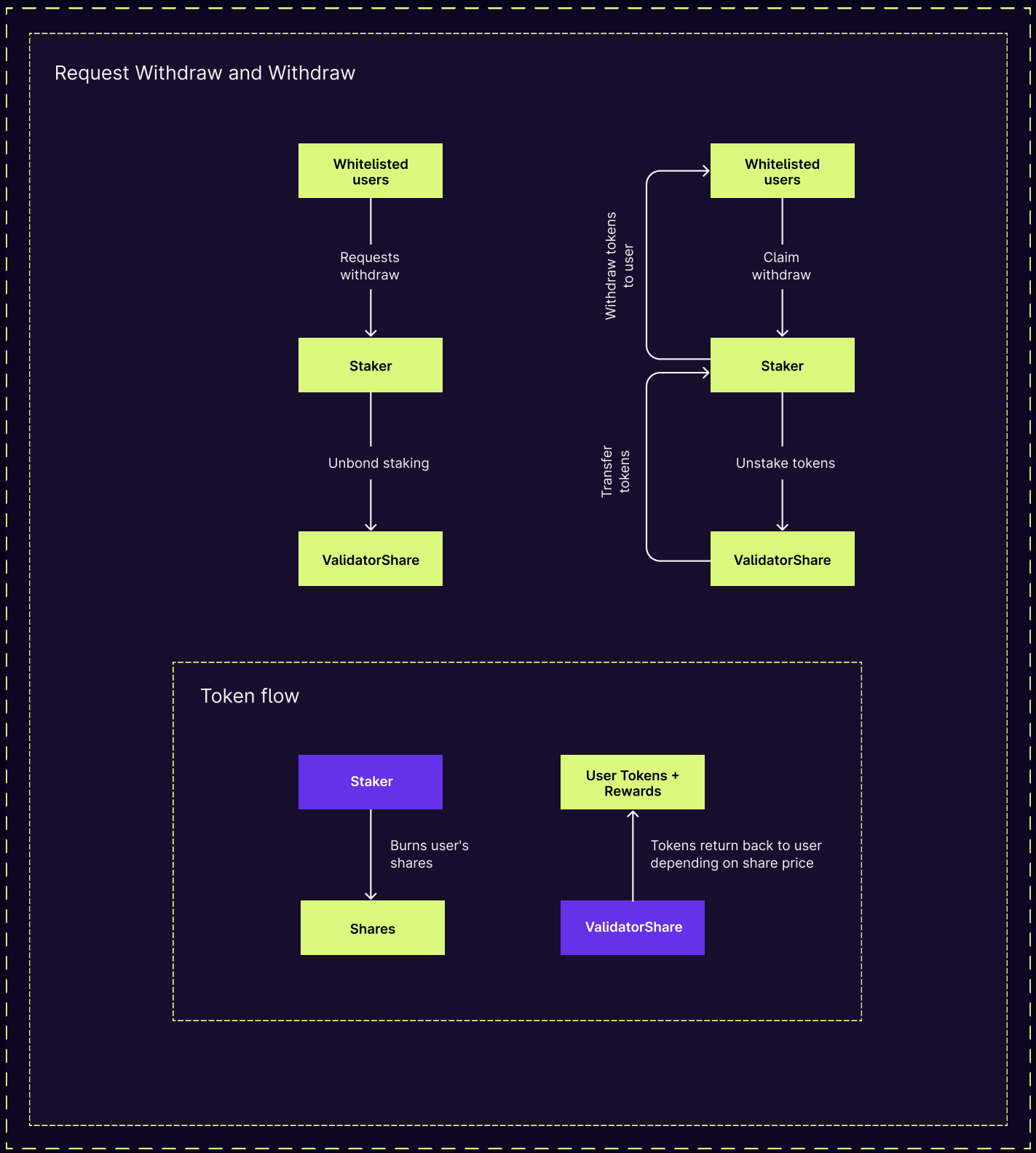
TruFin staking scheme



TruFin staking scheme



TruFin staking scheme



STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. The issues tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

MEDIUM-1 | VERIFIED

Un-whitelisted users can't withdraw their funds.

Staker.sol: withdrawRequest(), withdrawClaim().
Only whitelisted users can deposit, request or claim withdrawals. Thus in case a user has deposited while he was whitelisted and then later he was removed from whitelist, he won't have access to his funds and will withdraw them.

Recommendation:
Allow users who were whitelisted and deposited to withdraw tokens even if they were removed from the whitelist.

From client:
According to the Trufin team, un-whitelisted users should be able to withdraw funds since they can be removed from the whitelist in case they are put on OFAC sanctions list and shouldn't interact with the protocol any longer.

LOW-1 | RESOLVED

Parameters lack validation.

Staker.sol: initialize() - validate parameters before deploy.
Setter functions - setStakingToken(), setStakeManagerContract(), setValidatorShareContract(), setWhitelist(), setTreasury(), setCap().
setPhi() - validate that function argument is less or equal to phiPrecision constant.
It is recommended to validate that address parameters are not zero addresses so that contract will work without issues.

Recommendation:
Validate functions parameters.

From client:
Validation of phi was added. As for address validation, setting a zero address might be a valid case, thus validation is unnecessary.

INFO-1 | RESOLVED

Custom errors should be used.

Starting from the 0.8.4 version of Solidity it is recommended to use custom errors instead of storing error message strings in storage and use “require” statements. Using custom errors is more efficient in terms of gas spending and increases code readability.

Recommendation:
Use custom errors.

INFO-2 | RESOLVED

Lack of documentation.

Adding NatSpec to contract functions and variables will make it more understandable about functions and variables. As an example, a contract uses a specific token for staking. In this case, having documentation (NatSpec) in the contract description and function would be helpful, where a token will be used.

Recommendation:
Add NatSpec documentation.

Post-audit:
A detailed NatSpec documentation was added.

Dangerous transfer call.

Staker.sol: stakeClaimedRewards().
Function stakeClaimedRewards is invoking _deposit() function with zero address as parameter. Later in _deposit() function invokes safeTransferFrom which transfers zero amount of token from zero address. Though it works fine in fork tests, but if try to use any other token rather than MATIC with transferFrom, transaction will fail as it is not allowed to transfer from zero address. Though current implementation works fine with MATIC on Ethereum network, in case of other tokens being used, transactions fail.

Recommendation:
Validate that if _user is zero address in _deposit() function, it should skip safeTransferFrom OR validate that contract should only support MATIC.

From client:
Trufin team has verified that smart contract is supposed to interact only with MATIC on Ethereum network.

	StakerStorage.sol	Staker.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting TruFin in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Hardhat testing framework.

The tests were based on the functionality of the code, as well as a review of the TruFin contract requirements for details about issuance amounts and how the system handles these.

Staker

Deposit operations
- ✓ Should deposit (402ms)
- ✓ Should deposit twice (549ms)
- ✓ Should deposit (2 users) (537ms)
- ✓ Should deposit zero amount (232ms)
- ✓ Shoudn't deposit more than CAP (85ms)

Rewards operations
- ✓ Should simulate `SubmitCheckpoint` transaction on RootChainProxy (6250ms)
- ✓ Should compound unclaimed rewards (5890ms)
- ✓ Should withdraw after compound unclaimed rewards (6171ms)
- ✓ Should compound unclaimed rewards (2 users) (6014ms)
- ✓ Shouldn't compound 0 rewards (241ms)

Withdraw operations
- ✓ Should withdraw part (443ms)
- ✓ Should withdraw all (412ms)
- ✓ Should withdraw parts twice (597ms)
- ✓ Should withdraw + stakeClaimedRewards (6448ms)
- ✓ Shouldn't withdraw zero amount (38ms)
- ✓ Shouldn't withdraw more than deposited (217ms)

WithdrawClaim operations
- ✓ Shouldn't claim without requested withdrawal (42ms)
- ✓ Shouldn't claim with incomplete withdrawal period (48ms)
- ✓ Shouldn't claim with non-existent unbond nonce
- ✓ Shouldn't claim already claimed withdrawal (130ms)
- ✓ Should withdrawClaim (197ms)

ClaimList operations
- ✓ Shouldn't claim if one from list has not matured (148ms)
- ✓ Shouldn't claim list when one has already been claimed (193ms)
- ✓ Shouldn't claim list when one request from list was from different user (100ms)

✓ Should claimList (205ms)
✓ Should claim two of three from list (149ms)
✓ Should claim one of three from list (77ms)

Additional shares calculation check

✓ Should calculate shares correctly (1265ms)
✓ Should deposit user's funds + amount that was previously on contract
✓ Should withdraw more than deposited after share price increase

30 passing

FILE	% STMTS	% BRANCH	% FUNCS
StakerStorage.sol	100	100	100
Staker.sol	94.3	77.3	95.4
All files	**97.15**	**88.65**	**97.7**

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by the TruFin team

As a part of our work assisting TruFin in verifying the correctness of their contract code, our team has checked the complete set of tests prepared by the TruFin team.

We need to mention that the original code has a significant original coverage with testing scenarios provided by the TruFin team. All of them were also carefully checked by the team of auditors.

Staker

Owner: Initial State
✓ initialize (131ms)
Owner: Setters
✓ setStakingToken (112ms)
✓ setStakeManagerContract (95ms)
✓ setValidatorShareContract (87ms)
✓ setWhitelist (75ms)
✓ setTreasury (82ms)
✓ setCap (83ms)
✓ setPhi (77ms)
User: deposit
✓ single deposit (396ms)
✓ repeated deposits (604ms)
✓ multiple account deposits (527ms)
✓ deposit zero matic (196ms)
✓ try depositing more than the cap (68ms)
Vault: Simulate rewards accrual
✓ Simulating `SubmitCheckpoint` transaction on RootChainProxy (6659ms)
Vault: compound reward
✓ rewards compounded correctly (compoundRewards: using unclaimed rewards) (6241ms)
✓ rewards compounded correctly (stakeClaimedRewards: using claimed rewards) (6385ms)
✓ try compounding rewards with rewards equal to zero (281ms)
User: withdrawRequest
✓ initiate a partial withdrawal (469ms)
✓ initiate a complete withdrawal (443ms)
✓ initiate multiple partial withdrawals (697ms)
✓ initiate withdrawal with rewards wip (8136ms)
✓ try initiating a withdrawal of size zero (63ms)
✓ try initiating withdrawal of more than deposited (233ms)

User: withdrawClaim
✓ try claiming withdrawal requested by different user (49ms)
✓ try claiming withdrawal requested 79 epochs ago (73ms)
✓ try claiming withdrawal with unbond nonce that doesn't exist
✓ try claiming already claimed withdrawal (90ms)
✓ successfully claim withdrawal requested 80 epochs ago with expected changes in state and balances (107ms)
User: claimLis
✓ try to claim test unbonds when one has not matured (528ms)
✓ try to claim test unbonds when one has already been claimed (219ms)
✓ try to claim test unbonds when one has a different user (97ms)
✓ successfully claim three test unbonds consecutively (454ms)
✓ successfully claim two of three test unbonds inconsecutively (454ms)
✓ successfully claim just one withdrawal (325ms)

34 passing (58s)

FILE	% STMTS	% BRANCH	% FUNCS
StakerStorage.sol	100	100	100
Staker.sol	92.06	75	93.55
All files	**96.03**	**87.5**	**96.76**

We are grateful for the opportunity to work with the TruFin team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the TruFin team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo