trufin.io

SMART CONTRACT AUDIT

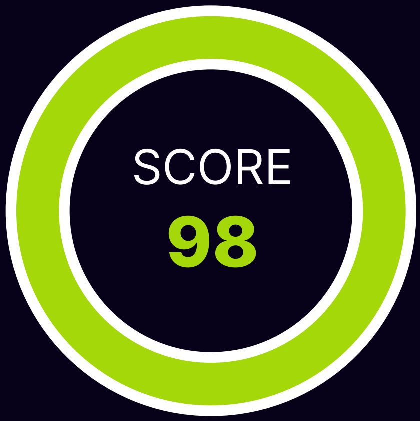
zokyo

May 22nd 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.

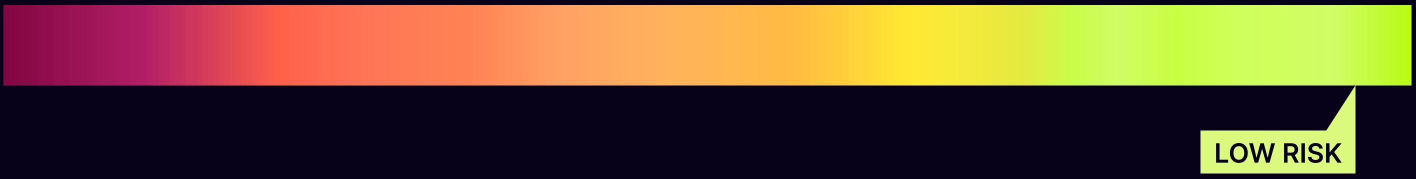


TECHNICAL SUMMARY

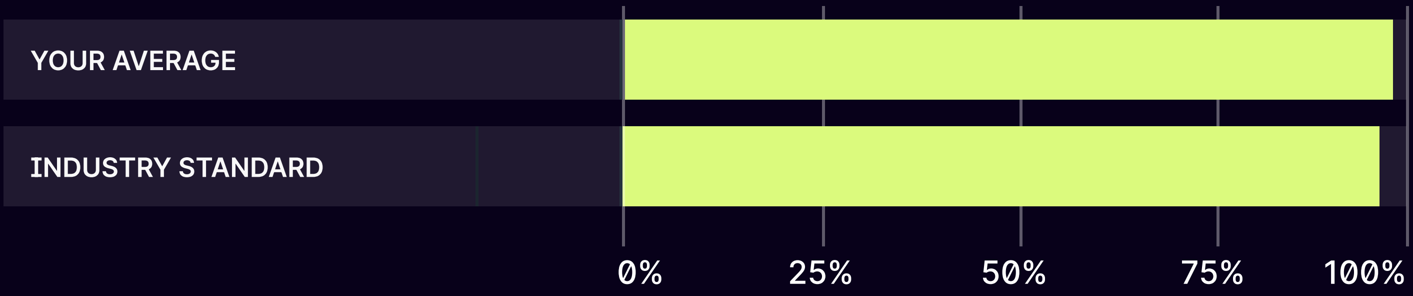
This document outlines the overall security of the TruFin smart contracts evaluated by the Zokyo Security team.

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. 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 for 1st audit iteration 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 the 1st audit iteration, the team of auditors reviewed the following contract(s):

- StakerStorage.sol
- Staker.sol

The source code of the smart contract for 2nd audit iteration was taken from the TruFin repository:
https://github.com/TruFin-io/staker-audit-april

Branch: master
Initial commit: 07edcd1dee454ece2a7f0377c92f74d713c8503f
Final commit: e5065cb2e3c7c49e28d0644b1128eae0fa4d5a75

Within the scope of the 2nd audit iteration, the team of auditors reviewed the following contract(s):

- TruStakeMATICv2Storage.sol
- TruStakeMATICv2.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.

02 Cross-comparison with other, similar smart contracts by industry leaders.

03 Testing contract logic against common and uncommon attack vectors.

04 Thorough manual review of the codebase line by line.

Executive Summary

Zokyo Security received a set of contracts that comprise the Trufin staker protocol. Two contracts were within the audit scope: Staker.sol and StakerStorage.sol. The staking function allows users to deposit their Matic tokens on the Ethereum mainnet and receive shares in exchange. Matic is then invested into Polygon PoS Staking Contract, allowing staking contracts to earn rewards, which are distributed among the protocol users. Thus, the 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 correctly and safely.

The manual audit 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. If users had deposited Matic tokens while they were allowlisted, they wouldn't be able to withdraw them if they were removed from the whitelist. Trufin verified that such functionality is necessary: if users are put on the sanction list, they can no longer interact with the protocol. The other issues were connected to a lack of parameter validation, documentation, and usage of custom errors. The Trufin team has successfully fixed all of them. Also, one of the informational issues was initially 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 the stakeClaimedRewards() function, which performs a transfer of zero Matic from zero address. Since most ERC-20 tokens forbid direct transfers from zero addresses, such an 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 verified that only Matic would be used.

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

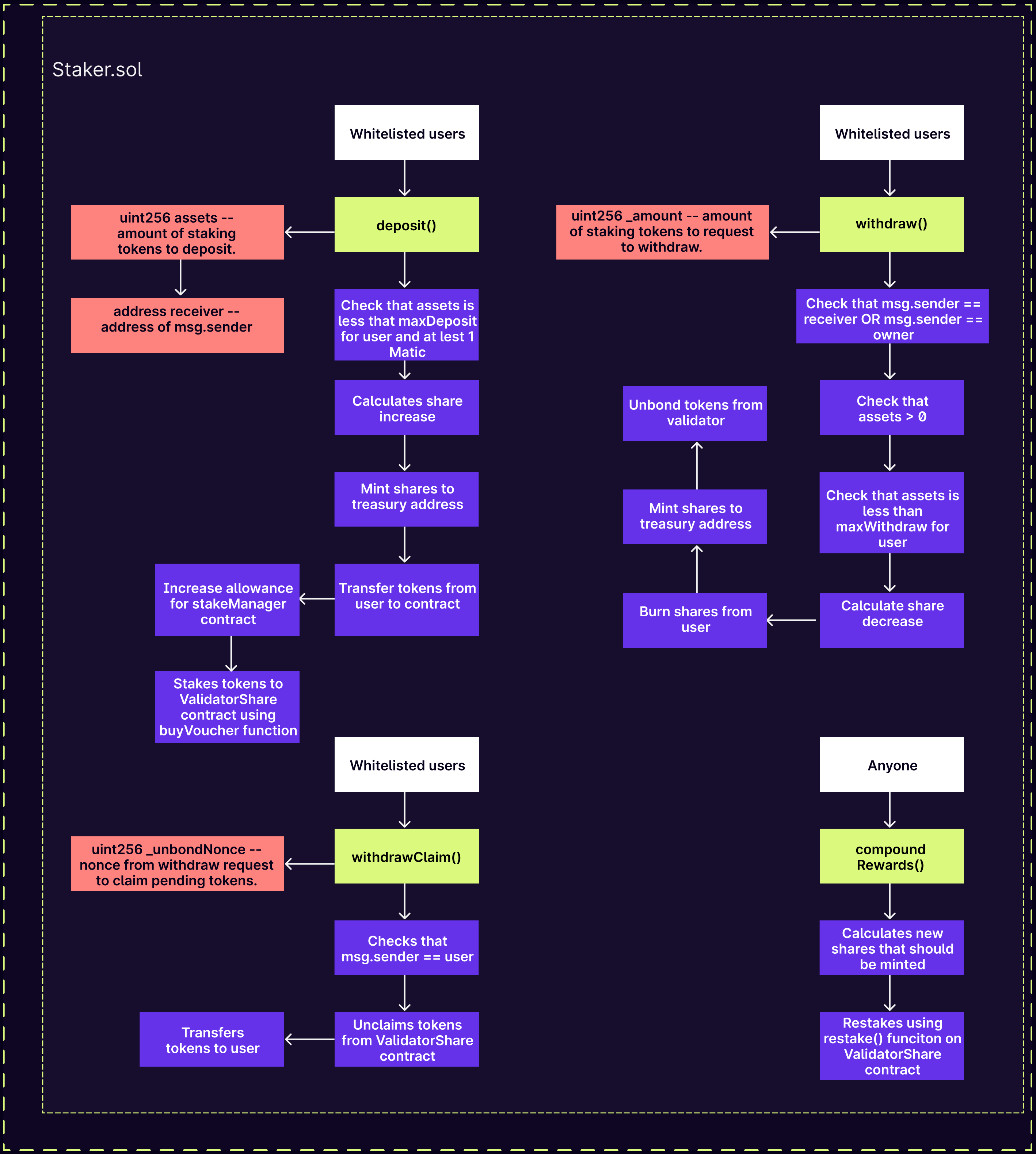
The overall security of smart contracts is high enough. Contracts are well-written and tested both by the Trufin and Zokyo Security teams. Moreover, Trufin added sufficient documentation describing all the smart contract aspects during the audit.

During the second audit iteration, Zokyo Security carefully reviewed all the changes made in smart contracts. Trufin added support for ERC-4626, a tokenized vault standard, and allocation functionality. Therefore, the second audit iteration aimed to check the correctness of ERC-4626 implementation as well as validate the standard vulnerabilities and attacks on this standard, check all the changes in the logic of smart contracts, validate the flow of new functionality, check smart contracts against the list of common vulnerabilities and verify that they are optimized in terms of gas consumptions.

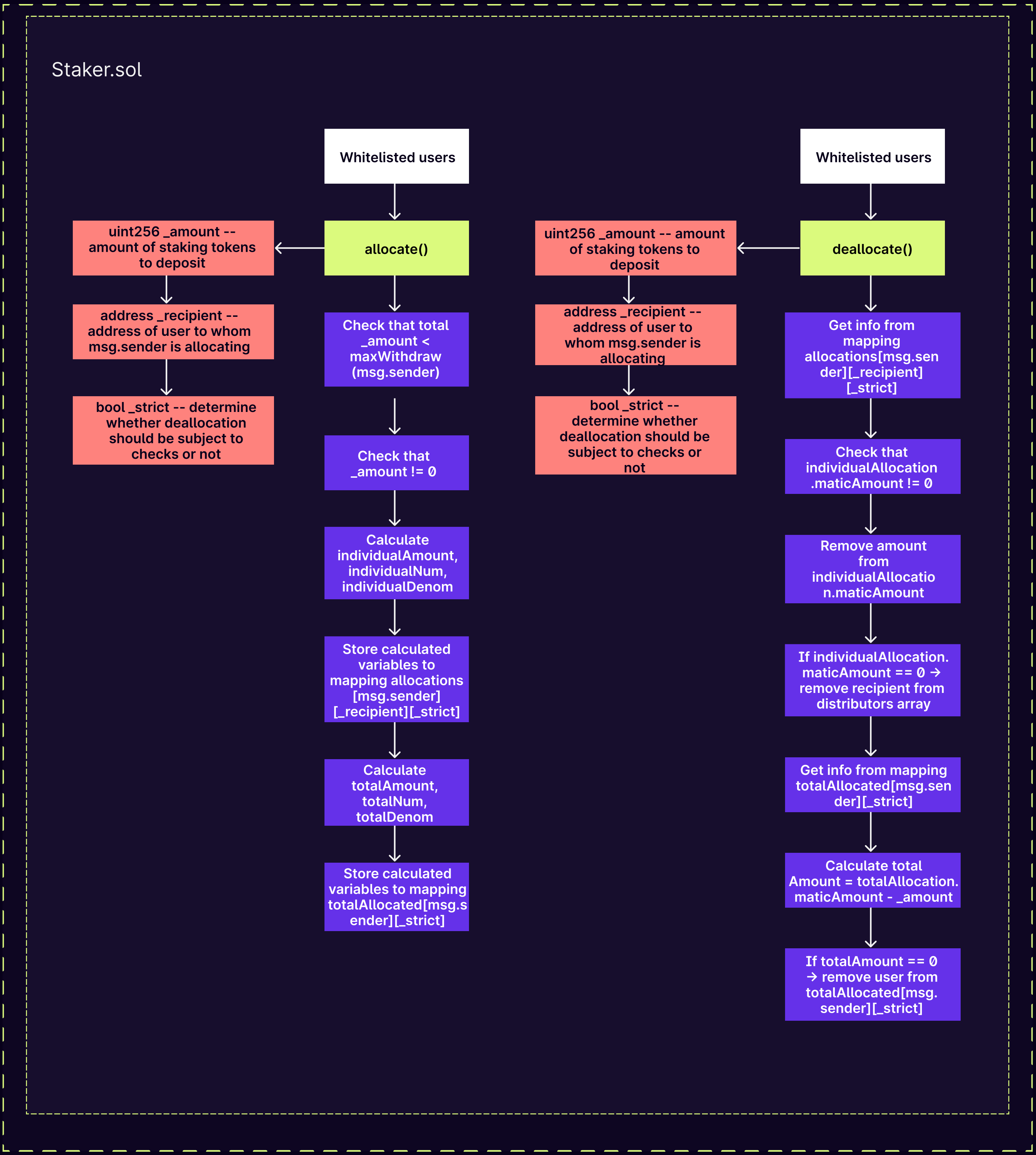
Auditors discovered one critical and several informational issues. The critical issue was connected to the inflation attack on ERC-4626. Auditors proposed several approaches to mitigate this issue. Trufin has successfully fixed it by introducing a minimum amount of Matic that users must deposit. The other issues were related to file naming, commented code, and business logic validation. Trufin has fixed or verified all of these issues as well.

The overall security of smart contracts is still high enough. Auditors have prepared a set of unit tests to validate logic and additional scenarios to validate the security of smart contracts and implementation of the ERC-4626 standard. Contracts are well-written, contain sufficient natSpec documentation, and have passed all the security tests.

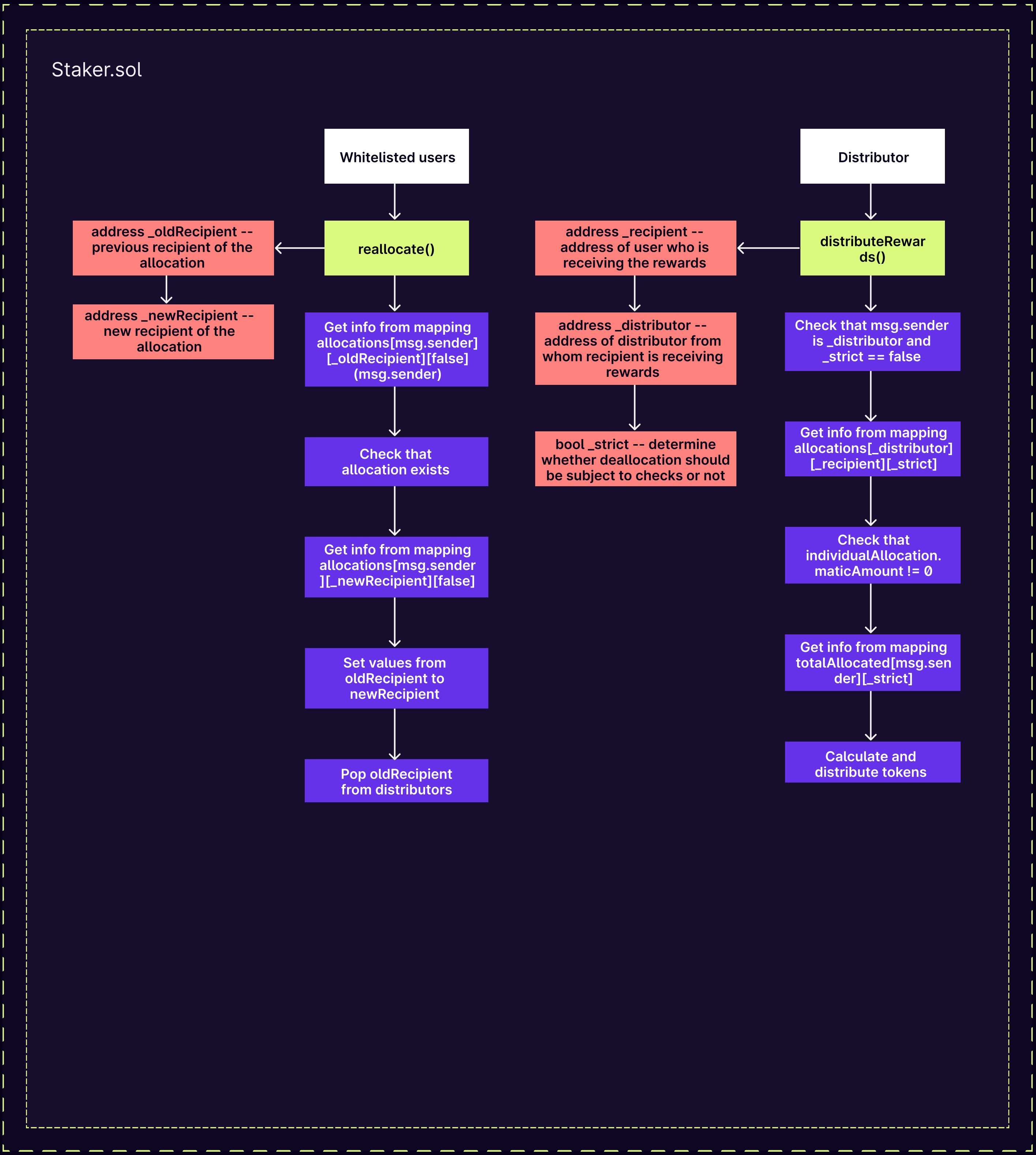
TruFin staking scheme


Staker.sol
Whitelisted users
deposit()
uint256 assets -- amount of staking tokens to deposit.
address receiver -- address of msg.sender
Check that assets is less that maxDeposit for user and at lest 1 Matic
Calculates share increase
Mint shares to treasury address
Transfer tokens from user to contract
Increase allowance for stakeManager contract
Stakes tokens to ValidatorShare contract using buyVoucher function
Whitelisted users
withdraw()
uint256 _amount -- amount of staking tokens to request to withdraw.
Check that msg.sender == receiver OR msg.sender == owner
Check that assets > 0
Check that assets is less than maxWithdraw for user
Calculate share decrease
Burn shares from user
Mint shares to treasury address
Unbond tokens from validator
Whitelisted users
withdrawClaim()
uint256 _unbondNonce -- nonce from withdraw request to claim pending tokens.
Checks that msg.sender == user
Unclaims tokens from ValidatorShare contract
Transfers tokens to user
Anyone
compound Rewards()
Calculates new shares that should be minted
Restakes using restake() funciton on ValidatorShare contract

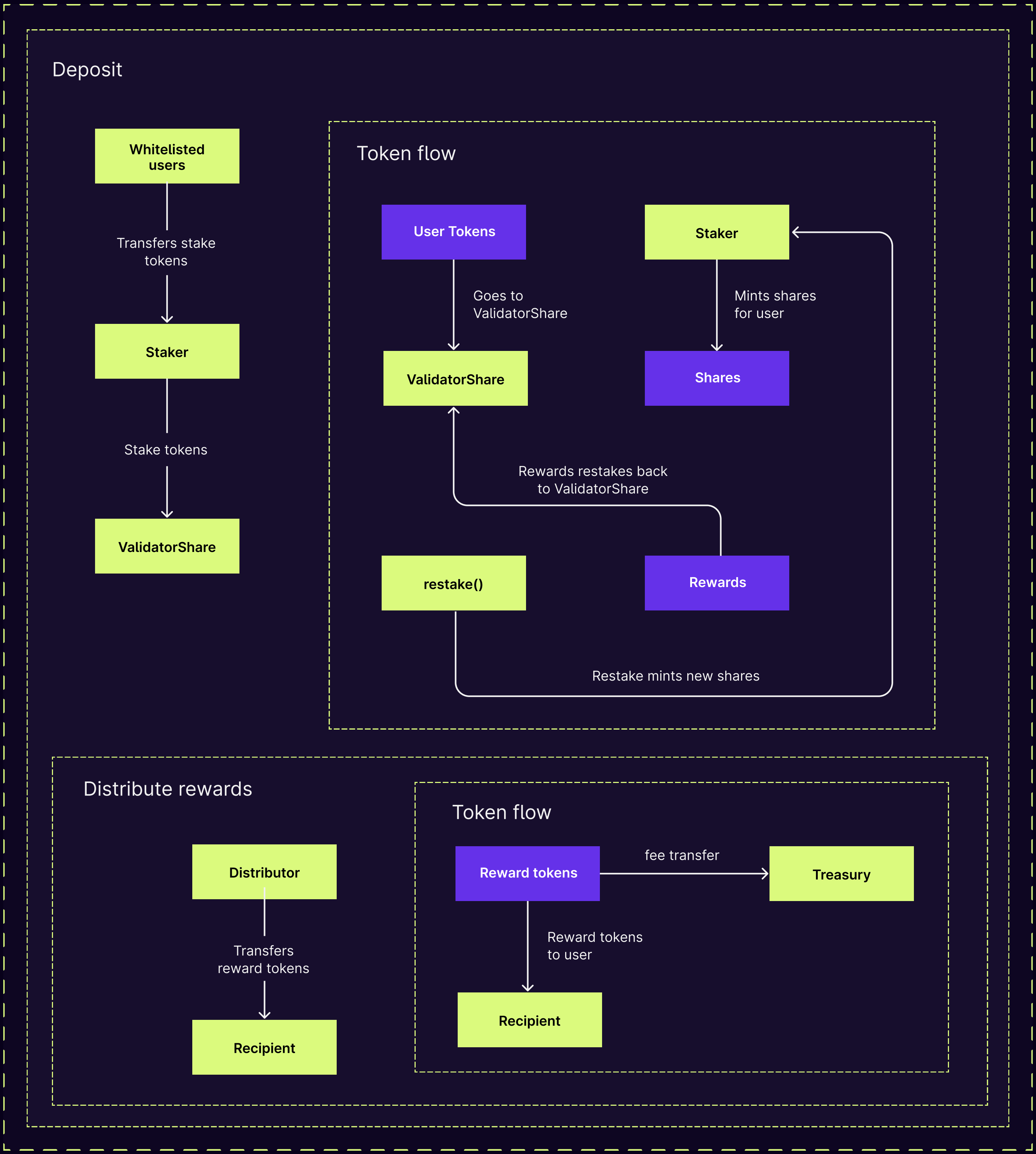
TruFin staking scheme


Staker.sol
Whitelisted users
allocate()
uint256 _amount -- amount of staking tokens to deposit
address _recipient -- address of user to whom msg.sender is allocating
bool _strict -- determine whether deallocation should be subject to checks or not
Check that total _amount < maxWithdraw (msg.sender)
Check that _amount != 0
Calculate individualAmount, individualNum, individualDenom
Store calculated variables to mapping allocations [msg.sender] [_recipient][_strict]
Calculate totalAmount, totalNum, totalDenom
Store calculated variables to mapping totalAllocated[msg.sender][_strict]
Whitelisted users
deallocate()
uint256 _amount -- amount of staking tokens to deposit
address _recipient -- address of user to whom msg.sender is allocating
bool _strict -- determine whether deallocation should be subject to checks or not
Get info from mapping allocations[msg.sender][_recipient][_strict]
Check that individualAllocation.maticAmount != 0
Remove amount from individualAllocation.maticAmount
If individualAllocation.maticAmount == 0 → remove recipient from distributors array
Get info from mapping totalAllocated[msg.sender][_strict]
Calculate total Amount = totalAllocation.maticAmount - _amount
If totalAmount == 0 → remove user from totalAllocated[msg.sender][_strict]

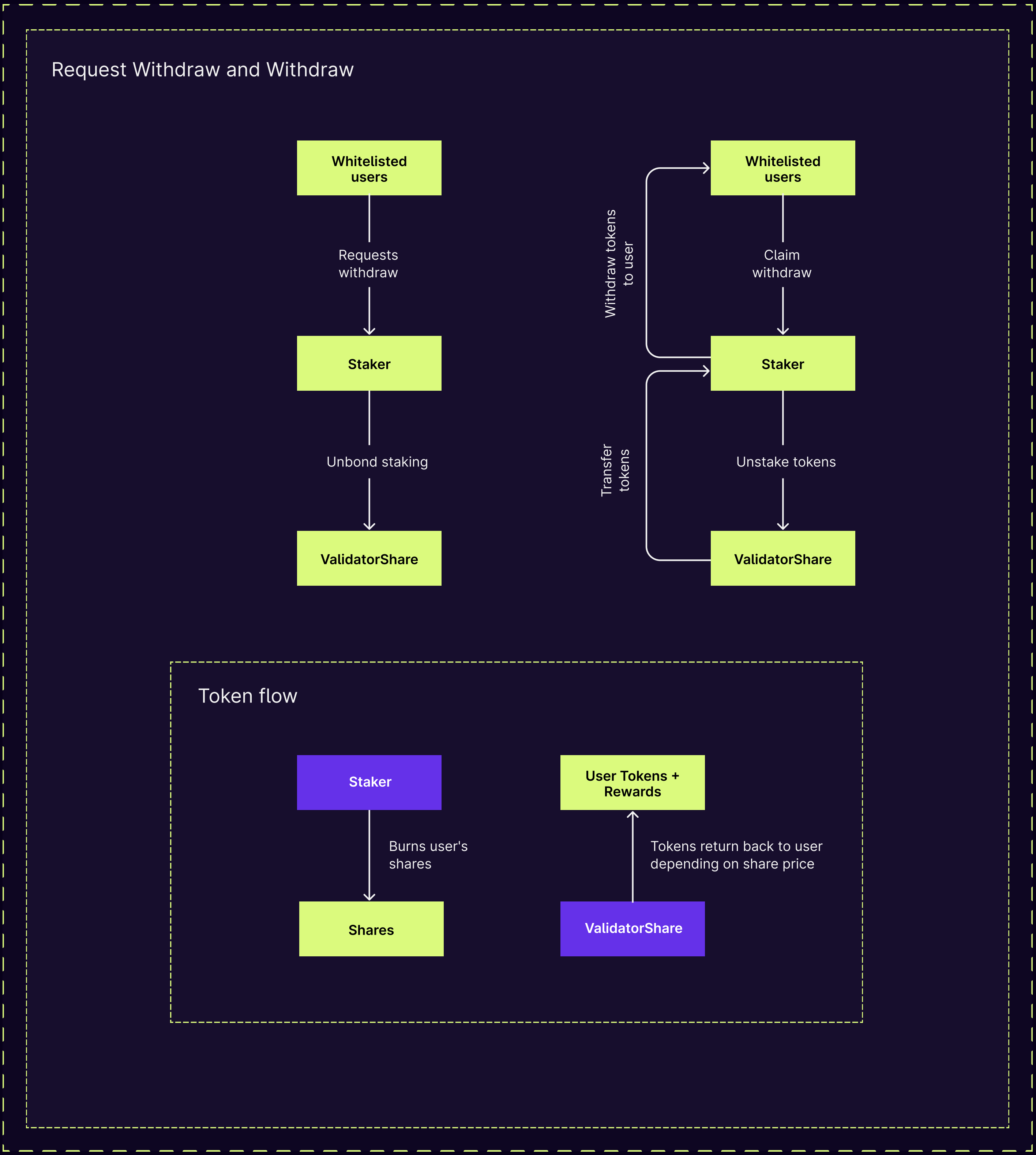
TruFin staking scheme


Staker.sol
Whitelisted users
reallocate()
address _oldRecipient -- previous recipient of the allocation
address _newRecipient -- new recipient of the allocation
Get info from mapping allocations[msg.sender][_oldRecipient][false](msg.sender)
Check that allocation exists
Get info from mapping allocations[msg.sender][_newRecipient][false]
Set values from oldRecipient to newRecipient
Pop oldRecipient from distributors
Distributor
distributeRewards()
address _recipient -- address of user who is receiving the rewards
address _distributor -- address of distributor from whom recipient is receiving rewards
bool _strict -- determine whether deallocation should be subject to checks or not
Check that msg.sender is _distributor and _strict == false
Get info from mapping allocations[_distributor][_recipient][_strict]
Check that individualAllocation.maticAmount != 0
Get info from mapping totalAllocated[msg.sender][_strict]
Calculate and distribute tokens

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

1ST AUDIT ITERATION 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, if a user has deposited while he was allowlisted and then later removed from the whitelist, he won't have access to his funds and will withdraw them.

Recommendation:

Allow users who were allowlisted to withdraw their deposited tokens even if they were removed from the whitelist.

From client:
According to the Trufin team, users who are not whitelisted should be able to withdraw funds. This is because once they are added to the OFAC sanctions list, they should no longer be able to participate in the protocol.

LOW-1 | RESOLVED

Parameters lack validation.

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

Recommendation:
Validate functions parameters.

From client:
The 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 regarding 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.

INFO-3 | VERIFIED

Dangerous transfer call.

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

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 has verified that smart contract is supposed to interact only with MATIC on the Ethereum network.

2ND AUDIT ITERATION COMPLETE ANALYSIS

CRITICAL-1 | VERIFIED

ERC4626 inflation attack is possible in the old ERC4626 version.

Staker.sol: line 10
The current implementation of the ERC4626 contract is susceptible to an underlying asset balance manipulation attack. You can check here the OpenZeppelin discussion. In the newer version of the ERC4626 it was fixed.

Recommendation:

Check the solution OpenZeppelin implemented and update the library version if it suits. Or implement one of the fixes discussed in the linked issue if the default fix doesn't suit your needs.

Post-audit.
The minimal deposit was set to 1 MATIC to prevent an inflation attack and correctly calculate deposits.

INFO-2 | RESOLVED

Hardhat console in pre-production code.

Staker.sol: line 14. It is recommended to remove any tools necessary for testing from pre-production code. Hardhat console is a tool for testing contract behavior. Thus, hardhat import should be removed from the contract when preparing for deployment.

Recommendation:
Remove import of Hardhat console from contract.

Post-audit.
The Hardhat import was removed.

INFO-3 | RESOLVED

Errors lack comments.

Staker.sol: lines 325-331.
The code contains sufficient natSpec documentation. In order to keep consistent and good readability of the code, it is recommended to add natSpec to add new features, including new custom errors. This would help inspect any errors when a contract is deployed.

Recommendation:
Add comments to errors.

Post audit.
The comments were added.

INFO-4 | RESOLVED

Commented code.

Staker.sol: _distributeRewardsUpdateTotal(), lines 1084-1086.
Staker.sol: _distributeRewards(), lines 1178-1181.
Pre-production smart contracts should not contain commented code, as it looks like an unfinished logic.

Recommendation:
Remove or uncomment any commented code.

Post-audit.
The commented code was removed.

INFO-5 | RESOLVED

Unnecessary subtraction/adding.

Staker.sol: allocate(), line 1018.
Staker.sol: deallocate(), line 1128.
The logic of adding 2 to sharePriceNum when adding allocation and subtracting 1 during deallocation is confusing. Thus its correctness should be verified (for example, in case it is necessary for precision accuracy).

Recommendation:
Verify that this logic is intended.

Post audit.
Trufin has changed the logic for calculating sharePrice.

INFO-6 | RESOLVED

Remove todo comment.

Staker.sol: _distributeRewardsUpdateTotal(), line 940
Pre-production code should not contain “to-do” comments.

Recommendation:
Remove the “// needs fixing” comment. Also, check if fixing is done and if this comment was mistakenly left.

INFO-7 | VERIFIED

Potential revert in distributeRewards() when depositor transfers all shares.

Staker.sol: _distributeRewards(), lines 1213, 1219.
When a depositor transfers all their shares to another account, and then the receiver claims rewards, the _distributeRewards() function may revert. If the depositor retains enough shares, the function will execute successfully. For example, if the depositor has 1000 shares and the reward is 2.5, the function will revert if the depositor has less than 2.5 shares. If the depositor has 2.5 or more shares, the function will execute, and the receiver will receive the reward. This behavior should be verified to ensure it is the intended functionality or if transferring allocated shares should be restricted.

Recommendation:
Verify that this behavior is intended and acceptable **OR** consider restricting allocated share transfers.

	TruStakeMATICv2Storage.sol	TruStakeMATICv2.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 THE 1ST AUDIT REVISION

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 THE 2ND AUDIT REVISION

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
    Inflation attack check
    ✓ Basic inflation attack
Staker
    Scenario 1
    ✓ Transferring allocated tokens (1474ms)
Scenario -- Check storage after allocate/deallocate/reallocate
    Flow
    ✓ Deposit as user1, deployer (207ms)
    ✓ Allocate user1 -> user2 (95ms)
    ✓ Allocate user1 -> deployer (93ms)
    ✓ Deallocate half user1 -> user2 (57ms)
    ✓ Deallocate last half user1 -> user2 (49ms)
    ✓ Reallocate user1 (deployer -> user2) (48ms)
    ✓ Allocate user1 -> deployer again (87ms)
    ✓ Reallocate user1 (user2 -> deployer) (70ms)
Staker
    Setters
    ✓ Set staking token
    ✓ Set staker manager contact
    ✓ Set validator share contract
    ✓ Set whitelist
    ✓ Set treasury
    ✓ Set cap
    ✓ Set phi
    ✓ Set dist phi
    Main
    ✓ Deposit (862ms)
    ✓ Mint (87ms)
✓ Withdraw (759ms)
✓ Redeem (194ms)
Allocations
✓ Allocate to user (336ms)
✓ Deallocate from user (154ms)
✓ Reallocate to another user (147ms)
✓ Distribute rewards (380ms)
Revert
✓ When try to initialize contract again
✓ When not owner tries to set staking token
✓ When not owner tries to set stake manager contract
✓ When not owner tries to set validator share contract
✓ When not owner tries to set whitelist
✓ When not owner tries to set treasury
✓ When not owner tries to set cap
✓ When not owner tries to set phi
✓ When not owner tries to set dist phi
✓ When try to set phi if it is too large
✓ When try to set dist phi if it is too large
✓ When not whitelisted user tries to allocate (38ms)
✓ When not whitelisted user tries to deallocate
✓ When not whitelisted user tries to reallocate
✓ When try to allocate if insufficient balance (39ms)
✓ When try to allocate 0 amount
✓ When try to deallocate
✓ When try to reallocate if allocation not exists
✓ When not whitelisted user tries to deposit
✓ When not whitelisted user tries to mint
✓ When not whitelisted user tries to withdraw
✓ When not whitelisted user tries to redeem
```

FILE	% STMTS	% BRANCH	% FUNCS
Staker.sol	81.78	65.75	84.31
StakerStorage.sol	100	100	100

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