Blaize.Security

May 2nd, 2023 / V. 1.0

EVERSTAKE
SMART CONTRACT AUDIT

TABLE OF CONTENTS

Audit Rating 2
Technical Summary 3
The Graph of Vulnerabilities Distribution 4
Severity Definition 5
Auditing strategy and Techniques applied/Procedure 6
Executive Summary 7
Protocol Overview 9
Complete Analysis 20
Code Coverage and Test Results for All Files (Everstake) 34
Code Coverage and Test Results for All Files (Blaize Security) 37
Test Coverage Results (Blaize Security) 49
Disclaimer 51

AUDIT RATING

SCORE **9.8**/10

The scope of the project includes Everstake's set of contracts:

AutocompoundAccounting.sol
CommonAccounting.sol
Pool.sol
Accounting.sol
RewardsTreasury.sol
TreasuryBase.sol
Withdrawer.sol
WithdrawTreasury.sol
utils\OwnableWithSuperAdmin.sol
utils\Math.sol
structs\ValidatorList.sol
structs\StakerAccount.sol
structs\WithdrawRequests.sol

Repository:

https://github.com/everstake/ETH-Staking-B2C-SC

Branch: main

Initial commit:

- 939cde35aeda3910a4edb5c469ebb6556c8ac775

Final commit:

- 2e0235308a55cc3b496e6afc590971b71378a7cd

TECHNICAL SUMMARY

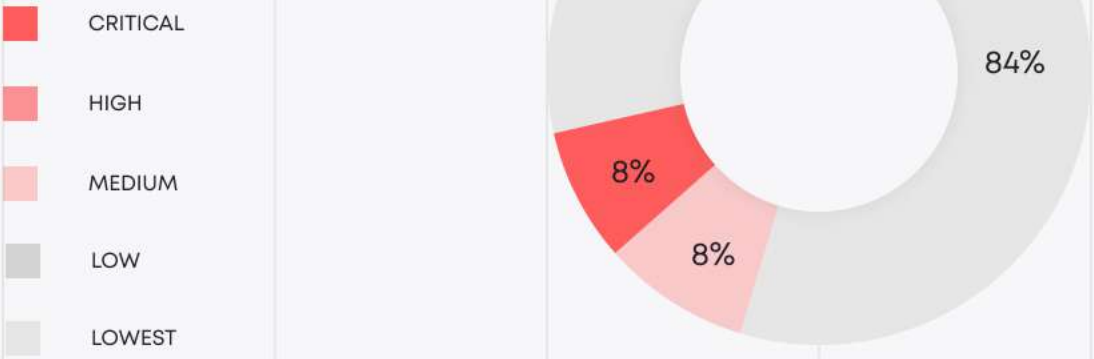
During the audit, we examined the security of smart contracts for the Everstake protocol. Our task was to find and describe any security issues in the smart contracts of the platform. This report presents the findings of the security audit of the **Everstake** smart contracts conducted between **April 10th, 2023** and **Apri 27th, 2023.**

Testable code

The code is 94% testable, which almost corresponds to the industry standard of 95%.

The scope of the audit includes the unit test coverage, which is based on the smart contract code, documentation and requirements presented by the Everstake team. The coverage is calculated based on the set of Hardhat framework tests and scripts from additional testing strategies. However, to ensure the security of the contract, the Blaize.Security team suggests that the Everstake team launch a bug bounty program to encourage further active analysis of the smart contracts.

THE GRAPH OF VULNERABILITIES DISTRIBUTION:



The table below shows the number of the detected issues and their severity. A total of 12 problems were found. 12 issues were fixed or verified by the Everstake team.

	FOUND	FIXED/VERIFIED
Critical	1	1
High	0	0
Medium	1	1
Low	0	0
Lowest	10	10

SEVERITY DEFINITION

Critical

The system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Requires immediate fixes and a further check.

High

The system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge data or financial leak. Requires immediate fixes and a further check.

Medium

The system contains issues that may lead to medium financial loss or users' private information leak. Requires immediate fixes and a further check.

Low

The system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Requires fixes.

Lowest

The system does not contain any issues critical to the secure work of the system, yet it is relevant for best practices.

AUDITING STRATEGY AND TECHNIQUES APPLIED/PROCEDURE

We have scanned this smart contracts for commonly known and more specific vulnerabilities:

- Unsafe type inference;
- Timestamp Dependence;
- Reentrancy;
- Implicit visibility level;
- Gas Limit and Loops;
- Transaction-Ordering Dependence;
- Unchecked external call - Unchecked math;
- DoS with Block Gas Limit;
- DoS with (unexpected) Throw;
- Byte array vulnerabilities;
- Malicious libraries;
- Style guide violation;
- ERC20 API violation;
- Uninitialized state/storage/ local variables;
- Compile version not fixed.

Procedure

We checked the contracts for the following parameters:

- Whether the contract is secure;
- Whether the contract corresponds to the documentation;
- Whether the contract meets the best practices in the efficient use of gas, code readability.

Automated analysis:

Scanning contracts by several publicly available automated analysis tools such as Mythril, Solhint, Slither, and Smartdec. Manual verification of all the issues found with tools.

Manual audit:

Manual analysis of smart contracts for security vulnerabilities. We checked smart contract logic and compared it with the one described in the documentation.

EXECUTIVE SUMMARY

The Blaize Security team has conducted an audit of the Everstake B2C protocol. Everstake is a protocol that allows users to stake their ETH to receive rewards. It uses a list of validators, and each is registered as an Ethereum validator by staking 32 ETH under him. For example, if 100 users have stake of 32 ETH in total, the protocol sends 32 ETH to the Beacon deposit smart contract to register one validator on the list.

The audit's goal was to verify the security of staking, withdrawal, and collection of rewards mechanisms and validate the security of users' funds. Also, audit smart contracts against the list of common vulnerabilities as well as our internal checklist, and check that contracts are optimized in terms of gas usage.

During the audit, a critical issue was identified regarding access control. Thus, it was pointed out that anyone could set the super admin account while the super admin was equal to zero address. In this case, a malicious actor could have been monitoring the mempool to search for deployment or setting super admin to zero address and using a front-run attack to set his address. The Everstake team has successfully fixed the issue.
The other issues were connected to iteration through the whole storage array, lack of events, and several gas optimizations. All of them were successfully fixed as well.

The overall security of smart contracts is high enough: they have passed all the security checks. However, the implementation of smart contracts is complex and lacks documentation. Nevertheless, the Blaize Security team has thoroughly tested the whole set.

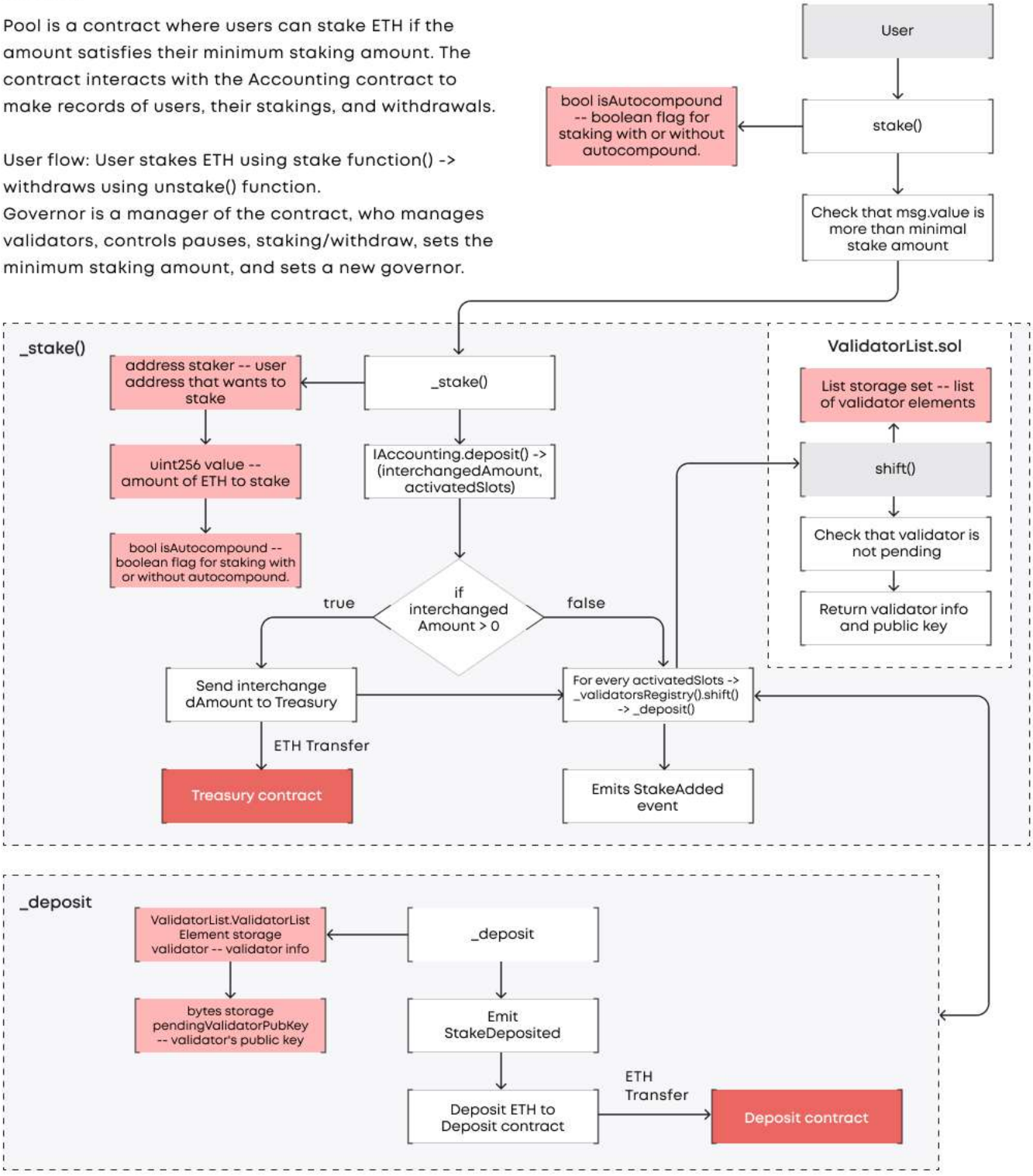
	RATING
Security	9.8
Gas usage and logic optimization	10
Code quality	9.5
Test coverage	9.9
Total	9.8

EVERSTAKE

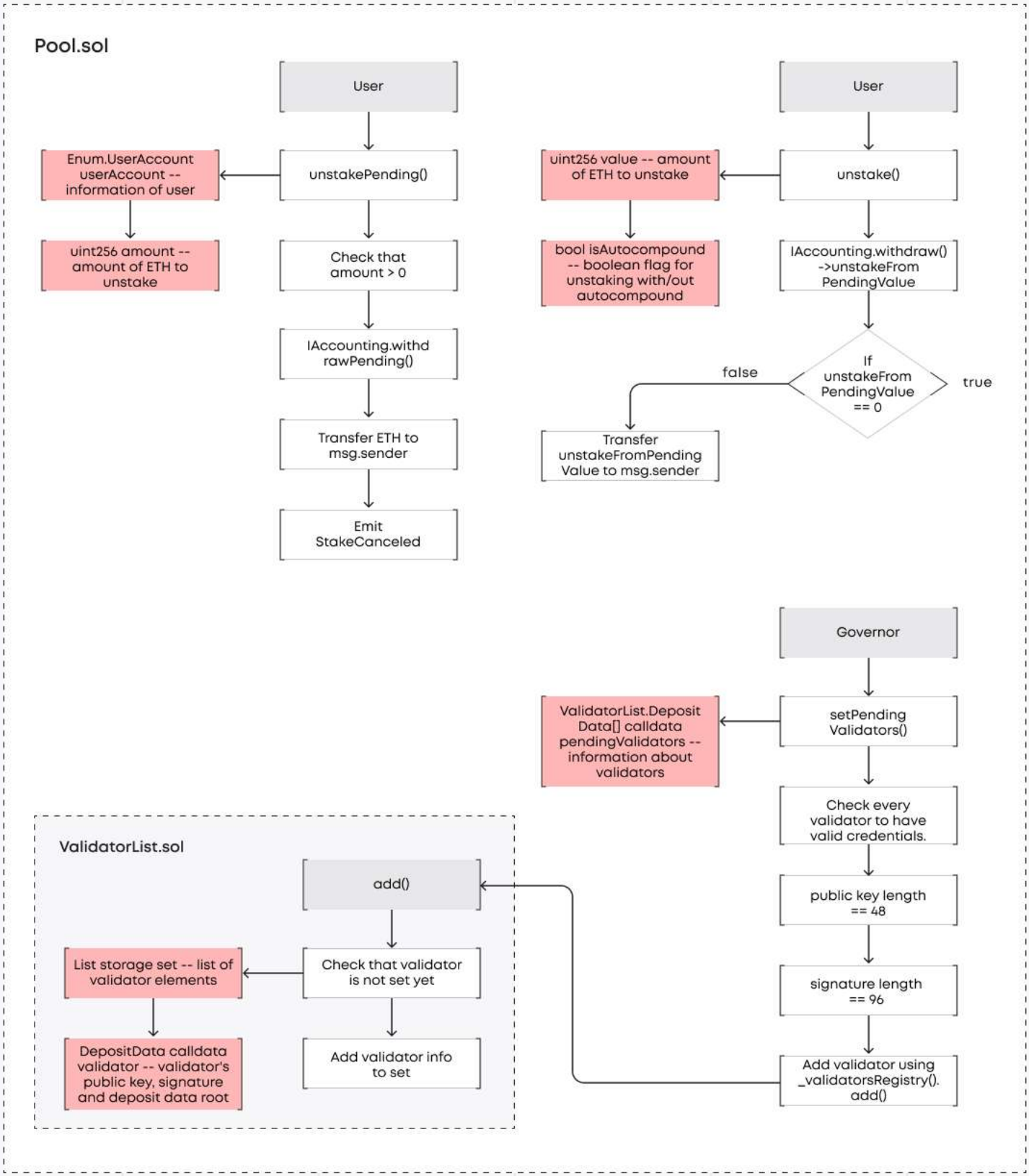
Pool.sol

Pool is a contract where users can stake ETH if the amount satisfies their minimum staking amount. The contract interacts with the Accounting contract to make records of users, their stakings, and withdrawals.

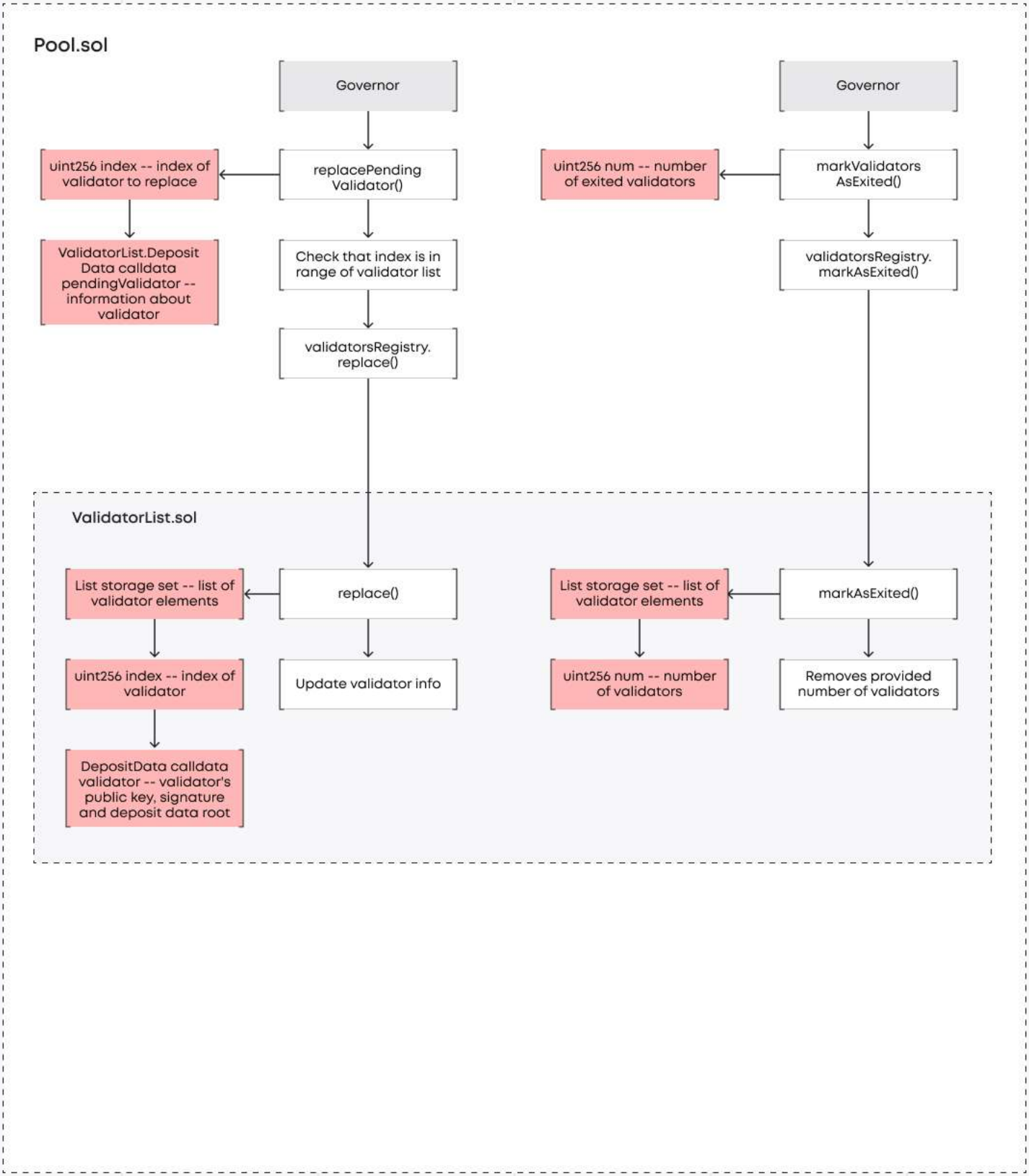
User flow: User stakes ETH using stake function() -> withdraws using unstake() function.
Governor is a manager of the contract, who manages validators, controls pauses, staking/withdraw, sets the minimum staking amount, and sets a new governor.



EVERSTAKE

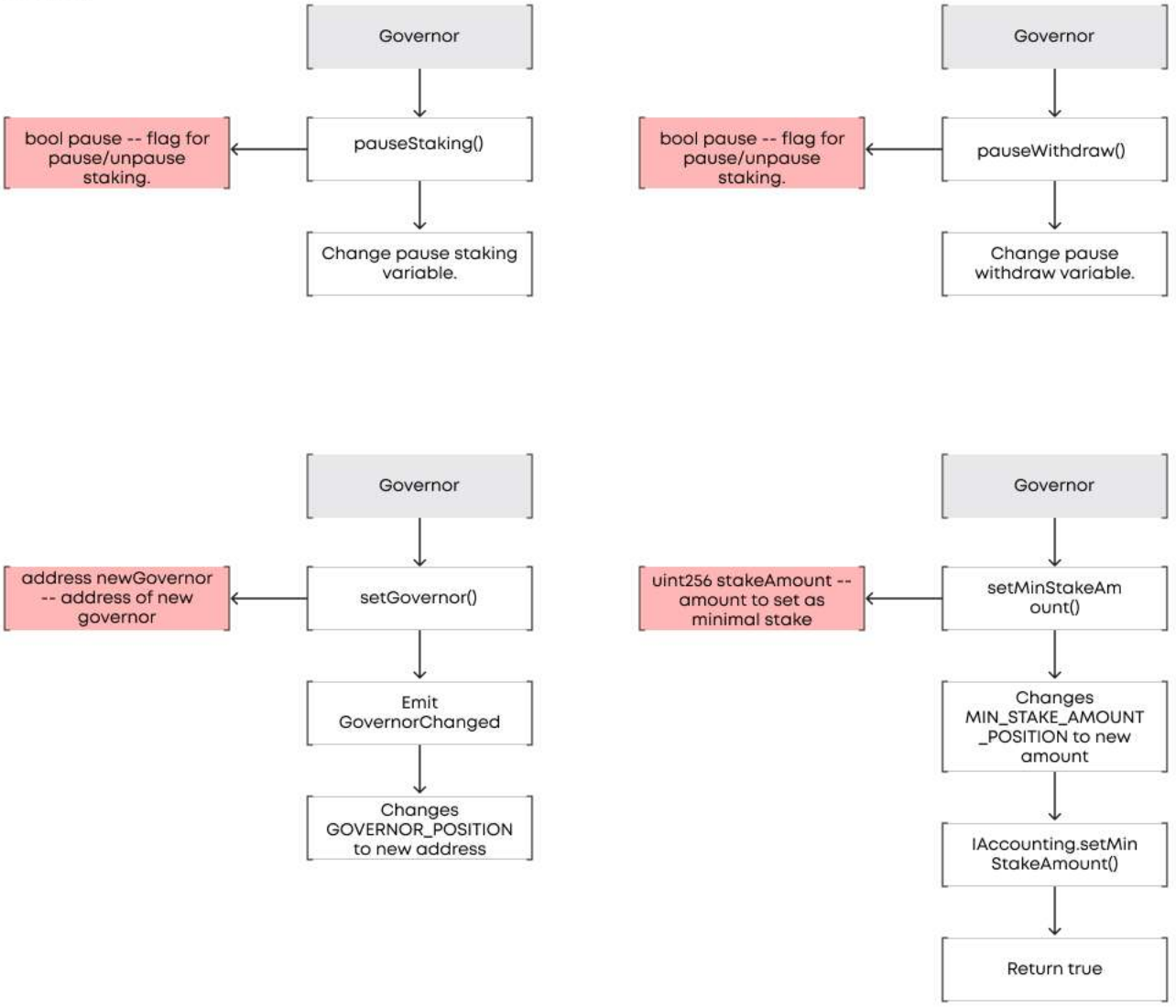

Pool.sol
User
unstakePending()
Enum.UserAccount userAccount -- information of user
uint256 amount -- amount of ETH to unstake
Check that amount > 0
IAccounting.withdrawPending()
Transfer ETH to msg.sender
Emit StakeCanceled
User
unstake()
uint256 value -- amount of ETH to unstake
bool isAutocompound -- boolean flag for unstaking with/out autocompound
IAccounting.withdraw() ->unstakeFrom PendingValue
If unstakeFrom PendingValue == 0
false
true
Transfer unstakeFromPending Value to msg.sender
Governor
setPending Validators()
ValidatorList.Deposit Data[] calldata pendingValidators -- information about validators
Check every validator to have valid credentials.
public key length == 48
signature length == 96
Add validator using _validatorsRegistry(). add()
ValidatorList.sol
add()
Check that validator is not set yet
List storage set -- list of validator elements
DepositData calldata validator -- validator's public key, signature and deposit data root
Add validator info to set

EVERSTAKE


Pool.sol
Governor
replacePending Validator()
uint256 index -- index of validator to replace
ValidatorList.Deposit Data calldata pendingValidator -- information about validator
Check that index is in range of validator list
validatorsRegistry. replace()
Governor
markValidators AsExited()
uint256 num -- number of exited validators
validatorsRegistry. markAsExited()
ValidatorList.sol
replace()
List storage set -- list of validator elements
uint256 index -- index of validator
DepositData calldata validator -- validator's public key, signature and deposit data root
Update validator info
markAsExited()
List storage set -- list of validator elements
uint256 num -- number of validators
Removes provided number of validators

EVERSTAKE

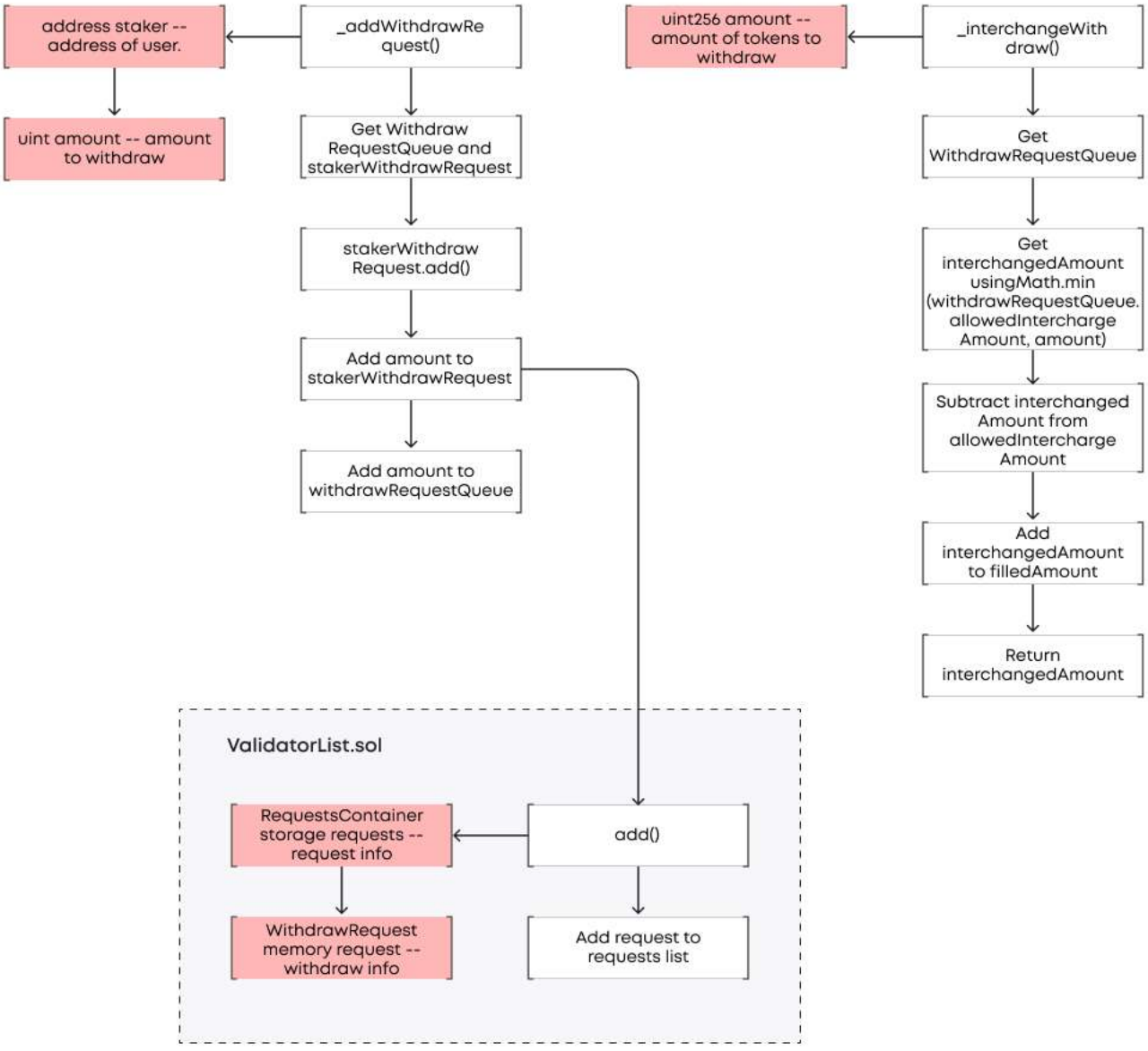
Pool.sol



EVERSTAKE

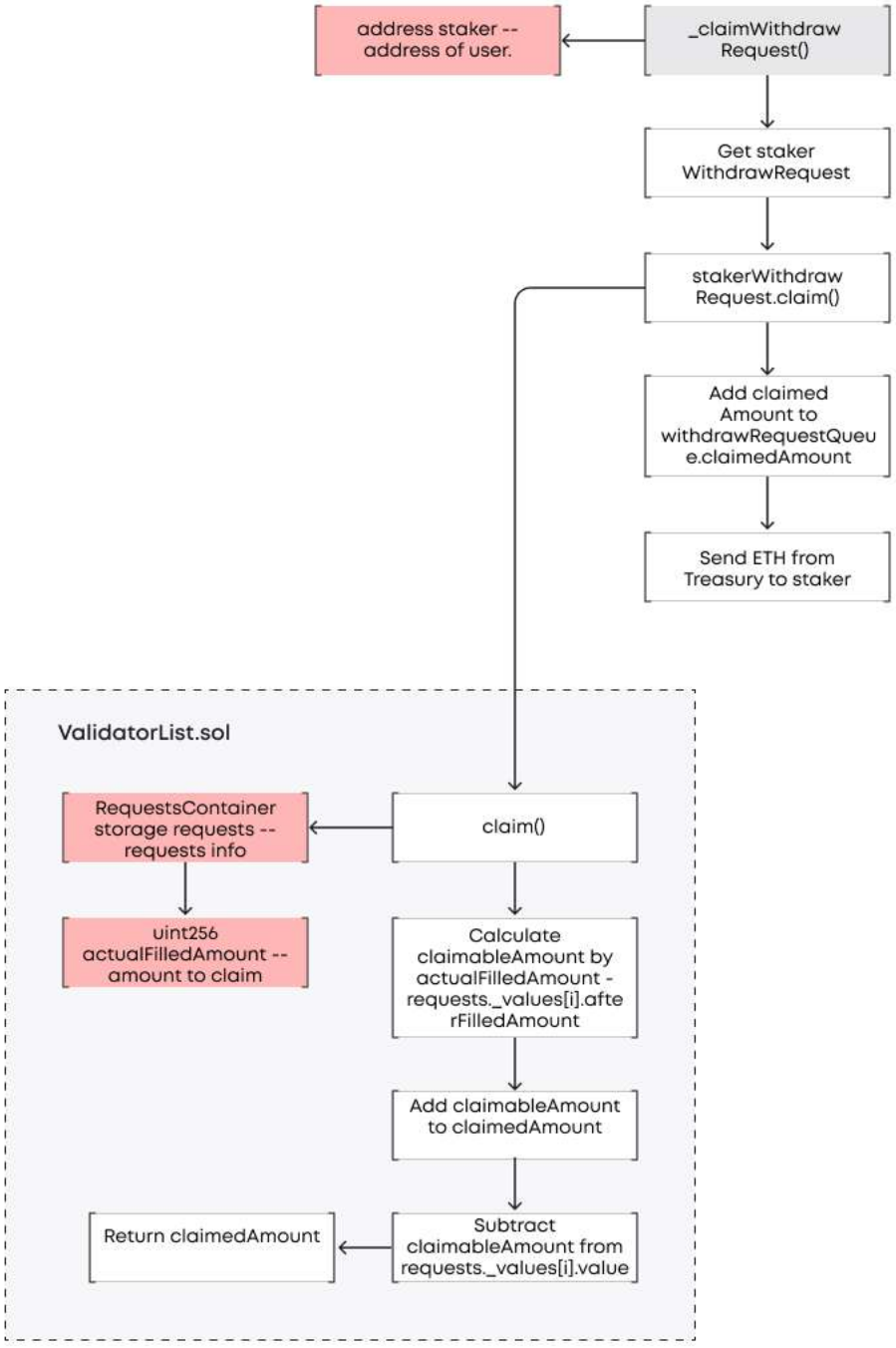
Withdrawer.sol

Withdrawer is a contract for storing withdrawal requests for ETH from Pool.
The contract stores information about the date of request and when withdraw was claimed.



EVERSTAKE

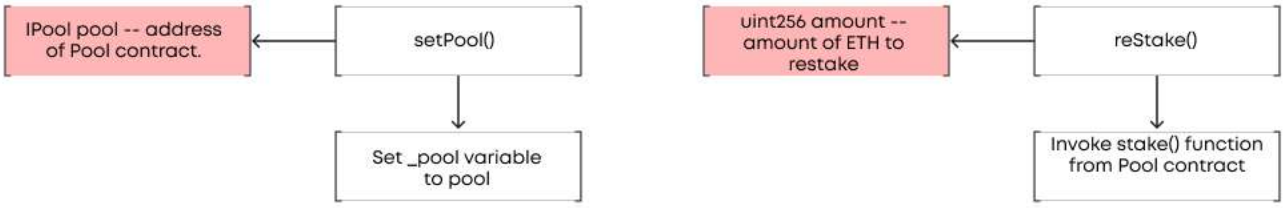
Withdrawer.sol



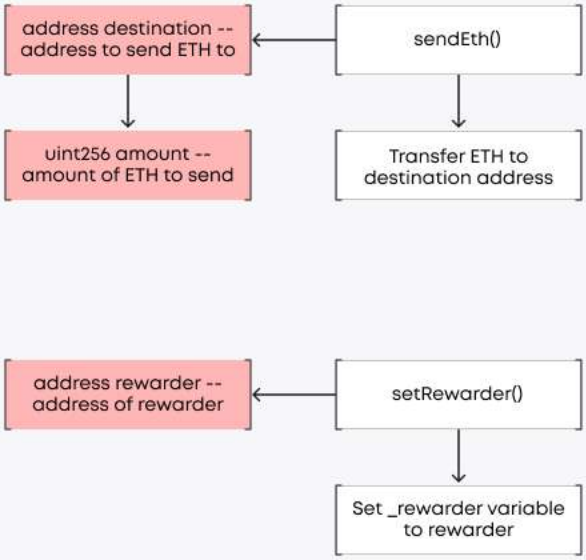
EVERSTAKE

RewardTreasury.sol

RewardTreasury is a contract for storing staked ETH.



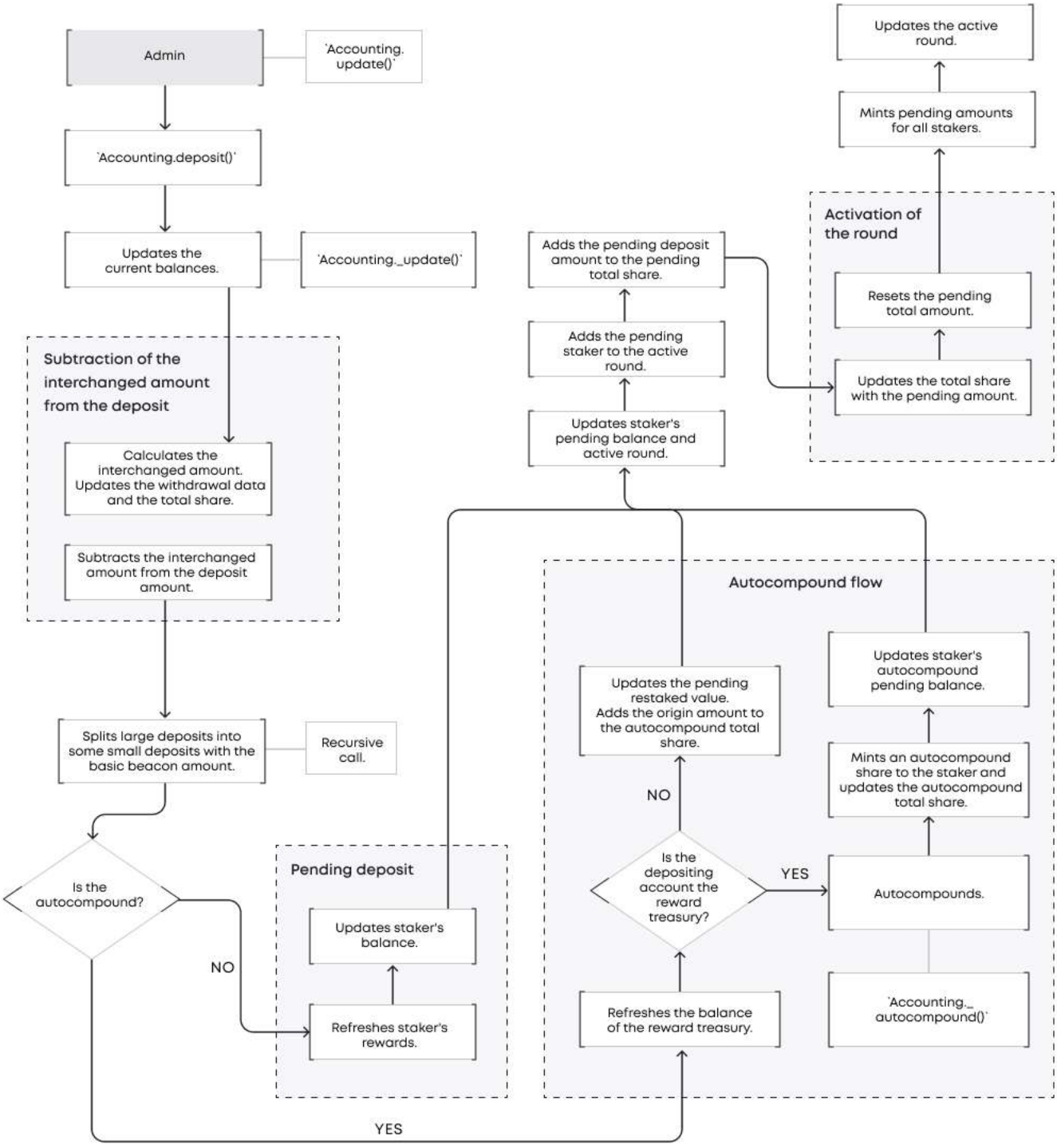
TreasuryBase.sol



EVERSTAKE

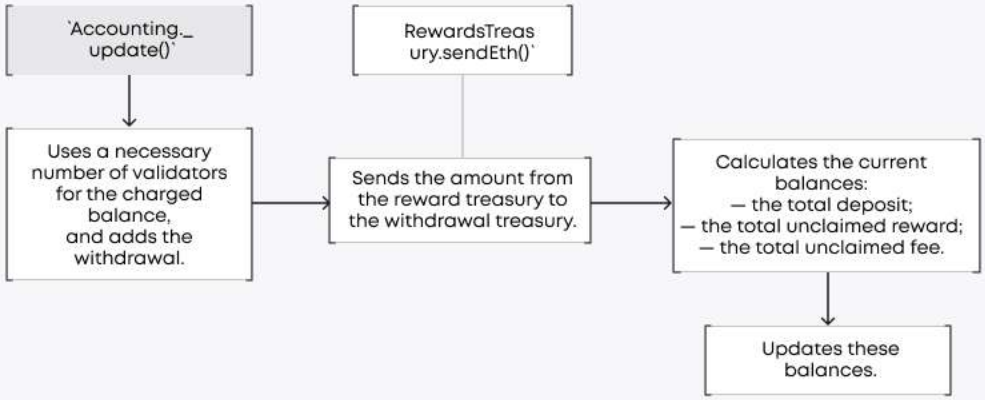
Accounting basic flows

Admin is assumed to be Pool.`

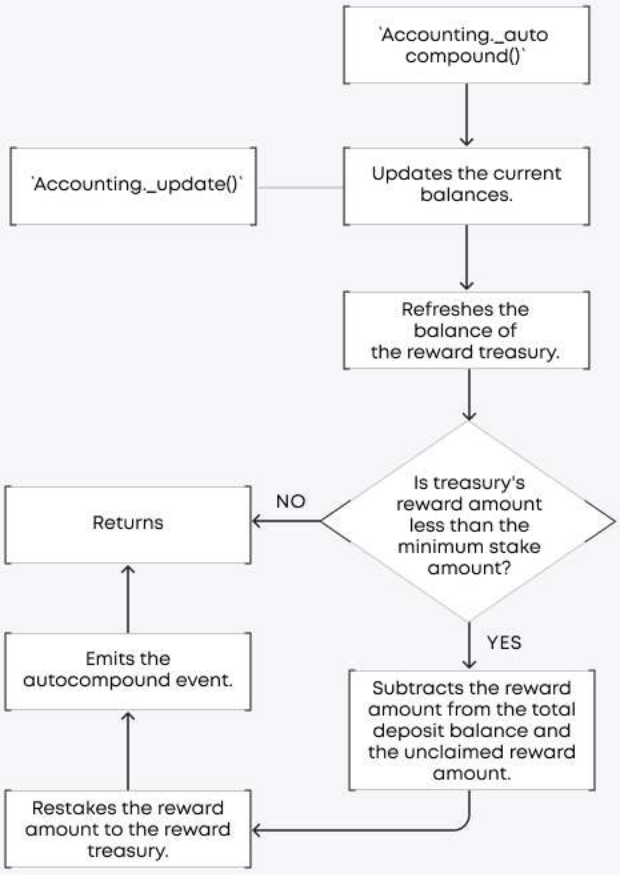


EVERSTAKE

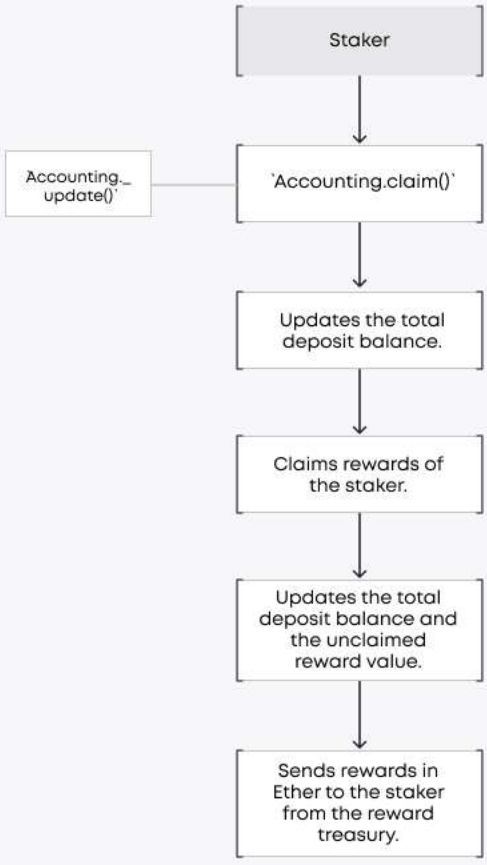
`Accounting` :update of the current balances.



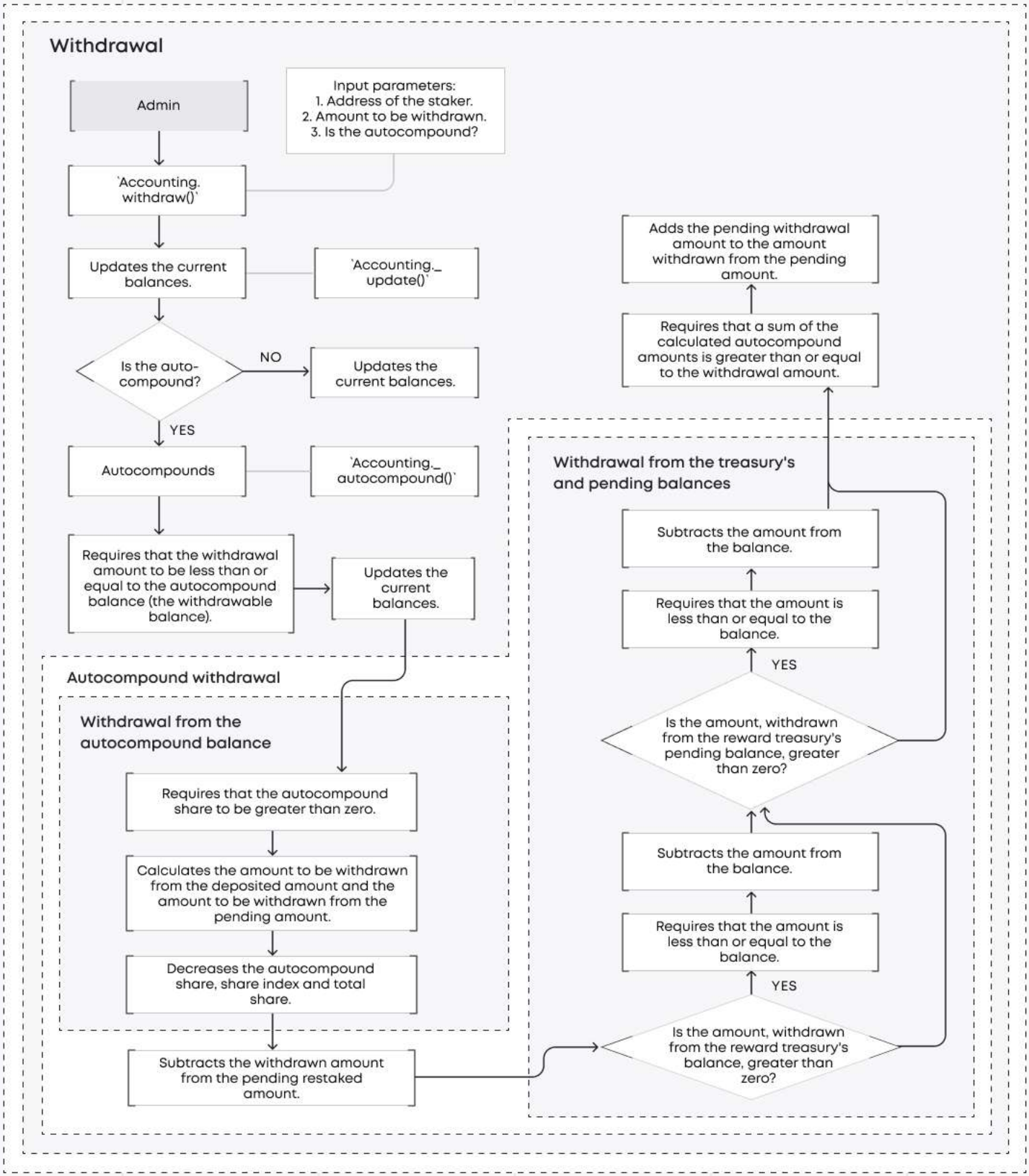
`Accounting` : autocompound



Claim of rewards

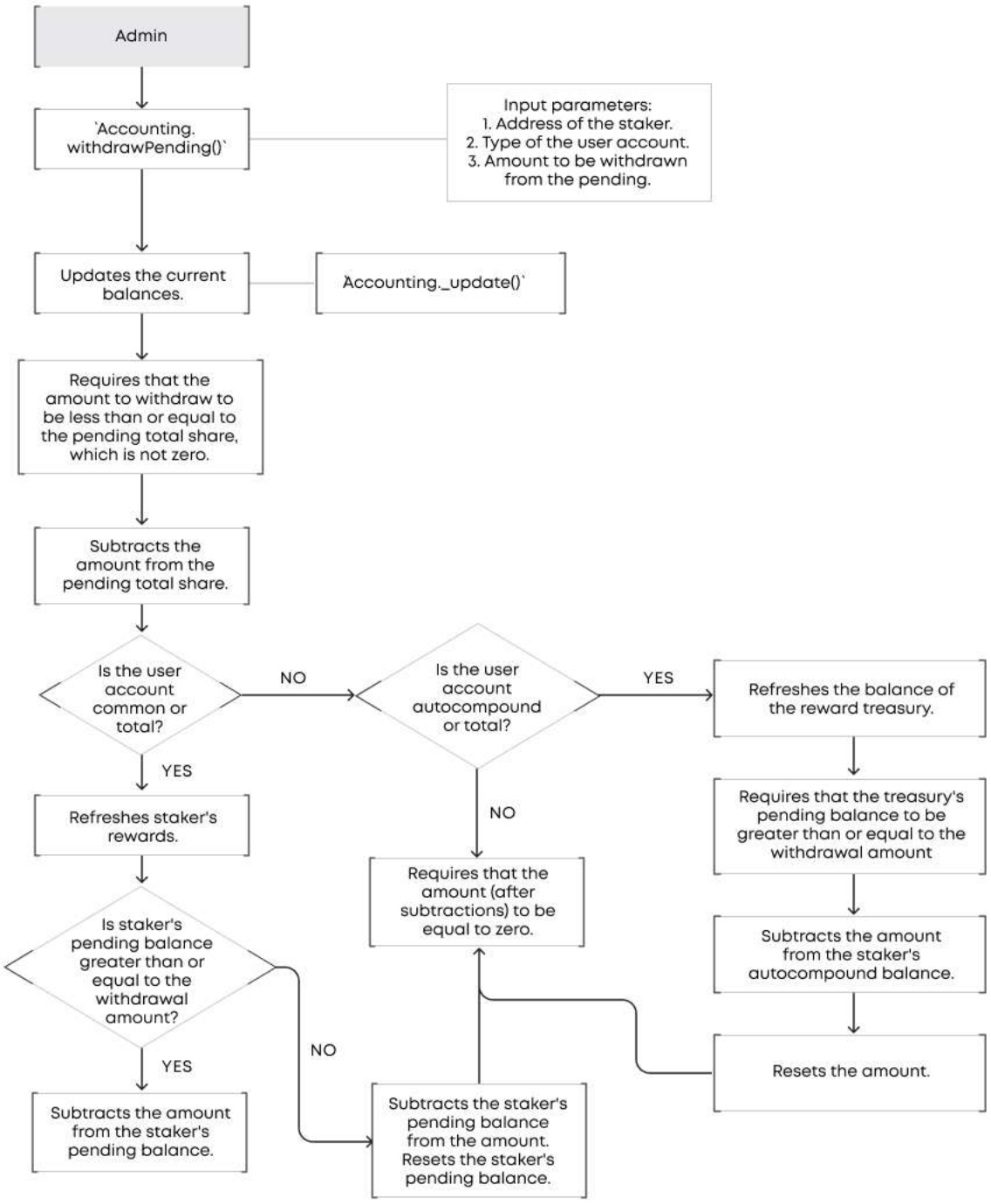


EVERSTAKE


Withdrawal
Admin
Input parameters:
1. Address of the staker.
2. Amount to be withdrawn.
3. Is the autocompound?
`Accounting.
withdraw()`
Updates the current balances.
`Accounting._
update()`
Is the auto-compound?
NO
Updates the current balances.
YES
Autocompounds
`Accounting._
autocompound()`
Requires that the withdrawal amount to be less than or equal to the autocompound balance (the withdrawable balance).
Updates the current balances.
Autocompound withdrawal
Withdrawal from the autocompound balance
Requires that the autocompound share to be greater than zero.
Calculates the amount to be withdrawn from the deposited amount and the amount to be withdrawn from the pending amount.
Decreases the autocompound share, share index and total share.
Subtracts the withdrawn amount from the pending restaked amount.
Withdrawal from the treasury's and pending balances
Subtracts the amount from the balance.
Requires that the amount is less than or equal to the balance.
YES
Is the amount, withdrawn from the reward treasury's pending balance, greater than zero?
Subtracts the amount from the balance.
Requires that the amount is less than or equal to the balance.
YES
Is the amount, withdrawn from the reward treasury's balance, greater than zero?
Adds the pending withdrawal amount to the amount withdrawn from the pending amount.
Requires that a sum of the calculated autocompound amounts is greater than or equal to the withdrawal amount.

EVERSTAKE

Withdrawal of pending rewards



COMPLETE ANALYSIS

CRITICAL-1	✔ Resolved

The super administrator can be set by anyone when it is equal to zero address.

OwnableWithSuperAdmin.sol: setSuperAdmin().
The state variable `_superAdmin` of the contract `OwnableWithSuperAdmin` is not set in the initializer. The function `setSuperAdmin()` can be called by anyone when `_superAdmin` is equal to zero address.
`OwnableWithSuperAdmin` is inherited by Pool`and `Accounting`, and can be inherited by another contract as this contract is an utility one. `Pool` is not required when the initialization of the super administrator should be set and has the vital modifier `onlyGovernor()`to which the administrator has access.
An intruder can monitor a mempool and cut in during deployment by setting the administrator's address. This can actually be done later, as the address setting is not guaranteed. Additionally, the function `setSuperAdmin()`does not emit any event, so the incident will probably not be noticed in time.

Recommendation:

Limit the ability to set the administrator address with the owner and set the address during initialization in contracts **OR** use OpenZeppelin's the `AccessControlUpgradeable` module.

Post-audit:
The Everstake team has added the modifier `ownerOrSuper` to setSuperAdmin() function.

MEDIUM-1	✔ Resolved

Iterating all the elements of the storage array during one transaction can result in a gas limit.

Accounting.sol: _activateRound().
There is a loop through all stakers of the storage array at SLOT_PENDING_STAKERS_POSITION` on lines 89–92. During the life of the contract, the array lengthens as new stackers are added. Eventually, the cost of the deposit transaction may exceed the gas limit, making it impossible for new accounts to make deposits.

Recommendation:

Implement the ability to split the calculation across multiple transactions by making the round activation functionality batch or otherwise.

Post-audit:

Pending stakers list now changes every new round. As a result, when a new round is coming, the pending stakers list refreshes till the next round.

LOWEST-1	✔ Resolved

Lack of events.

To keep track of historical changes of storage variables, it is recommended to emit events on every change in the functions that modify the storage:

- OwnableWithSuperAdmin.sol: setSuperAdmin().
- Accounting.sol: _activateRound(), deposit(), _update(), withdrawPending(), withdraw(), claimPoolFee(), setFee(), fromAutocompoundToMain(), fromMainToAutocompound(), claimWithdrawRequest(), setMinStakeAmount().
- AutocompoundAccounting.sol: _mint().
- RewardsTreasury.sol: setPool().
- Pool.sol: unstake(), setPendingValidators(), replacePendingValidator(), markValidatorsAsExited(), pauseStaking(), pauseWithdraw(), setMinStakeAmount().
- TreasuryBase.sol: sendEth(), setRewarder().
- Withdrawer.sol: _addWithdrawRequest(), _interchangeWithdraw(), _closedValidatorStop(), _claimWithdrawRequest().

Recommendation:

Emit event in these functions.

Post-audit.

All necessary events were added.

LOWEST-2	✔ Resolved

Custom errors can be used.

OwnableWithSuperAdmin.sol, Accounting.sol, AutocompoundAccounting.sol, CommonAccounting.sol, Pool.sol, TreasuryBase.sol, and Withdrawer.sol.
Since the Solidity version 0.8.4, custom errors can be used instead of requirements with string literals. Requirements with long error messages are costly regarding gas consumption and the length of bytecode taken up.
Therefore, it is recommended to replace all of them with the custom errors with meaningful names to

- reduce contract bytecode;
- make the code more readable in general;
- reduce the gas consumption on reverse.

It is worth noting that it can:

- transmit additional parameters to make the errors in specific cases more informative;
- be reused;
- be used cross-contract.

For example, `error MeaningfulName(parameters if needed);`

Recommendation:

Use custom errors instead.

Post-audit.

The Everstake team has created new contract called Errors, which is now used in contracts to implement custom errors instead of requires.

LOWEST-3	✔ Resolved

Incomplete code section.

Accounting.sol: line 264, 444.
There are to-do comments in the code which may indicate incomplete functionality.

Recommendation:

Complete this section **OR** verify that it is completed and delete the relevant comment.

Post-audit.
The comments were deleted.

LOWEST-4	✔ Resolved

Initializing variables in a loop.

Pool.sol: _stake(), line 113.
The local variables `validator` and `pendingValidatorPubKey` are declared in a loop, and therefore are destroyed and initialized with each iteration. For such cases, it is recommended to declare a variable before a loop to reduce the gas consumption.

Recommendation:

Declare the variables before the loop.

Post-audit.
The variables are now declared before the loop.

LOWEST-5	✔ Resolved

Assigning a zero value instead of deleting.

StakerAccount.sol, WithdrawRequests.sol, OwnableWithSuperAdmin.sol, Accounting.sol, AutocompoundAccounting.sol, and CommonAccounting.sol.
There is a resetting of storage variables by assignment a zero value instead of use the operator `delete`. For such cases, it is recommended to use it to reduce gas consumption.

- StakerAccount: `refreshRewards()` line 73.
- `WithdrawRequests`: `claim()` line 47.
- `OwnableWithSuperAdmin`: `renounceOwnership()` line 53.
- `Accounting`:
  - `withdrawPending()` line 201.
  - `_autocompound()` line 439.
- `AutocompoundAccounting`: `_autocompoundMintPending()` line 88.
- `CommonAccounting`: `_claimReward()` line 77.

Recommendation:

Use the `delete` operator for these cases.

Post-audit.

The `delete` operator is now used.

LOWEST-6	✔ Resolved

Local variables can be used.

WithdrawRequests.sol, Accounting.sol, and Pool.sol.
There are multiple readings of a storage variable or multiple calculations of an arithmetic expression without using a local variable.
It is recommended to create a local memory variable for such cases to locally store a value of a storage variable or a result of an expression to reduce gas consumption.

- `WithdrawRequests`:
  - `add()`: `requests._values.length` on lines 20, 27.
  - `claim()`:
    - `requests._values.length` on line 39;
    - `requests._values[i].value` on line 40, 45, 46.
  - info():
    - `requests._values.length` on line 57;
    - `requests._values[i].value` on lines 58, 62, 63.
- `Accounting`:
  - `_activateRound()`: `stakers.length()` on line 89.
  - `deposit()`: the passed variable `amount` can be used instead of `depositToPendingValue.` (`amount -= interchangedAmount`).
  - `_deposit()`:
    - PENDING_TOTAL_SHARE_POSITION.getStorageUint256()` on lines 116, 117;
    - the result of the sum on line 151 can be saved and also used instead of `PENDING_TOTAL_SHARE_POSITION.getStorageUint256()` on line 153.
  - `_autocompound()`: `rewardsTreasury` on line 428 can also be used for the call on line 445.

- Pool`, `_withdrawableBalance()`: `ACCOUNTING_CONTRACT_POSITION.getStorageAddress()`on lines 132, 133.

Recommendation:

Use local variables for the values.

Post-audit.

Local variables are now used.

LOWEST-7	✔ Resolved

Single-use local variables that do not save a value before overwriting.

StakerAccount.sol, Accounting.sol, AutocompoundAccounting.sol, Pool.sol, and Withdrawer.sol.

There are local variables that are used only once and do not save a value before its overwriting. For such cases, using an arithmetic expression directly without creating a local variable to reduce gas consumption is recommended.

- `StakerAccount,` `refreshRewards()`: a value of `staker.rewards`can be overwritten with the function result before `staker.index_with_precision = indexWithPrecision`.
- `Accounting`:
  - `withdraw()`:
    - `autocompoundBalance` on line 231;
    - `depositedAmount` on line 262;
    - `pendingStaker` on line 263.
  - `claimPoolFee()`: `rewarderBalance` on line 321.
  - `userPendingRewards()`: `balanceDiff` on line 397.
  - `_calculateBalanceChanges()`:
    - `index` on line 391;
    - `newIndexWithPrecision` on line 392.
  - `_autocompound()`: `rewardsTreasury` on line 428.
  - `autocompoundBalanceOf()`:
    - `balanceDiff` on line 454;

        - `autocompoundPoolPendingReward` on line 458;
        - `autoCompoundDepositPart` on line 460.
    - `fromAutocompoundToMain()`: `activeRound` on line 473.
- `AutocompoundAccounting:`
    - `_autoCompoundUserBalance()`: `staker` on line 27.
    - `_autocompoundMint()`: `stakerAccount` on line 94.
- `Pool`:
    - `_stake()`: `accountingContract` on line 104.
    - `_withdrawableBalance()`:
        - `commonBalance` on line 132;
        - `autocompoundBalance` on line 133.
    - `markValidatorsAsExited()`: `validatorsRegistry` on line 222.
- `Withdrawer:`
    - `_addWithdrawRequest()`: `stakerWithdrawRequest` on line 35.
    - `_closedValidatorStop()`: `withdrawRequestQueue` on line 76.
    - `_claimWithdrawRequest()`: `stakerWithdrawRequest` on line 97.
    - `withdrawRequest()`: `withdrawRequestQueue` on line 113.

Recommendation:

Use the arithmetic expressions directly at the point of need without creating a local variable.

Post-audit.

Direct arithmetic expressions are used now.

LOWEST-8	✔ Resolved

Usage of SafeMath`with the version 0.8.

StakerAccount.sol, Pool.sol.
The OpenZeppelin library SafeMathUpgradeable` is used for the library StakerAccount` and the contract Pool` that is implemented in Solidity version 0.8.17. SafeMath` is generally unnecessary starting with Solidity 0.8, since the compiler now has built in overflow checking. It is recommended to remove it and use the built-in checking of arithmetic operations to reduce gas consumption and improve code readability.

Recommendation:

Use the Solidity built-in checking instead of the library.

Post-audit.

`SafeMathUpgradeable` library was removed.

LOWEST-9	✔ Resolved

External visibility can be used.

- OwnableWithSuperAdmin.sol: renounceOwnership(), transferOwnership().
- Accounting.sol: userPendingRewards(), autocompoundBalanceOf(), fromAutocompoundToMain(), fromMainToAutocompound().

The functions have the `public` visibility, but they are not used anywhere inside the contract, inherited contracts and derived contracts. Therefore, it is recommended to use the `external` visibility for them to reduce the gas consumption and improve code readability.

Recommendation:

Use the `external` visibility for these functions.

Post-audit.

The functions are now declared as `external`.

LOWEST-10	✔ Resolved

Confusing naming.

Pool.sol: onlyGovernor().
The modifier is used to prevent unrestricted access to the functionality. Its name implies that it provides access to the functionality only for the governor, but in the condition it is also available to the owner and super administrator.

Recommendation:

Rename it according to usage.

Post-audit.

The modifier was renamed to `governorOwnerOrSuper`.

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

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

structs\ValidatorList.sol
structs\StakerAccount.sol
structs\WithdrawRequests.sol

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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY EVERSTAKE TEAM

> Compiled successfully using:
- solc: 0.8.17+commit.8df45f5f.Emscripten.clang
:. Fetching solc version list from solc-bin. Attempt #1
✓ Downloading compiler. Attempt #1.

Contract: Poolversion list from solc-bin. Attempt #1

Withdraw 9: 174377rsion list from solc-bin. Attempt #1
✓ success: ẁithdraw`(32 ETH(validator close) + claim) (2271ms)
success: ẁithdraw`(32 ETH(validator close) + claim) (2271ms)
✓ success: ẁithdraw`(32 ETH(validator close) + claim) (2271ms)
Withdraw 9: 197598rsion list from solc-bin. Attempt #1
✓ success: ẁithdraw autocompound`(32 ETH(validator close) + claim) (1660ms)
Withdraw 9: 218837rsion list from solc-bin. Attempt #1
✓ success: ẁithdraw autocompound`(1 ETH without pending) (2630ms)
Withdraw 9: 180292rsion list from solc-bin. Attempt #1
✓ success: ẁithdraw autocompound`(32 ETH(validator close) + claim. Deposit autocompound) (2814ms)wnloading compiler. Attempt #1.
Withdraw 9: 218837rsion list from solc-bin. Attempt #1
✓ success: ẁithdraw autocompound`(1 ETH without pending. Deposit autocompound) (2902ms)
Withdraw 9: 218837rsion list from solc-bin. Attempt #1
✓ success: ẁithdraw autocompound`(1 ETH without pending. Deposit autocompound) (2508ms)
✓ Downloading compiler. Attempt #1.

Contract: Pool

✓ success: ẁithdraw`(by 1 pending) (1733ms)pt #1
Withdraw 9: 401715rsion list from solc-bin. Attempt #1
✓ success: ẁithdraw`(by 9 pending) (7908ms)pt #1
✓ success: ẁithdraw`(2 withdraw rounds) (3047ms)
Withdraw Autocompound: 371992from solc-bin. Attempt #1
✓ success: ẁithdraw`(interchange with autocompound acc) (3044ms)
Autocompound GasUsed2: 358626from solc-bin. Attempt #1
Withdraw Autocompound: 157867from solc-bin. Attempt #1
✓ success: ẁithdraw`(interchange with autocompound pending rewards) (3928ms)

Autocompound GasUsed2: 284626from solc-bin. Attempt #1
Withdraw Autocompound: 183096from solc-bin. Attempt #1
Withdraw Autocompound: 143324from solc-bin. Attempt #1
✓ success: `withdraw`(interchange autocompound with common) (5463ms)
Autocompound GasUsed2: 318826 0xD5E8AA91BDa51688e6f3De82E52Fb38E3
Withdraw Autocompound: 117334from solc-bin. Attempt #1
✓ success: `withdraw`(interchange autocompound with autocompound rewards) (3095ms)
Withdraw Autocompound: 139935from solc-bin. Attempt #1
✓ success: `withdraw`(interchange autocompound with pending autocompound) (2669ms)
Autocompound GasUsed2: 243798 0x66078a97Def9d40B2cA7abb44733d D897Ec6231D
Withdraw Autocompound: 188810from solc-bin. Attempt #1
✓ success: `withdraw`(interchange autocompound with pending autocompound where (share > 1)) (3771ms)loading compiler. Attempt #1.

Contract: Pool

Stake GasUsed: 389028on list from solc-bin. Attempt #1
Stake with autocompound GasUsed2: 396866in. Attempt #1
Autocompound GasUsed2: 250426from solc-bin. Attempt #1
✓ success: `stake`(with autocompound) (4848ms) #1
✓ success: `off autocompound`(2 stakers) (2896ms)
✓ success: `on autocompound`(2 stakers) (1421ms)1
✓ Downloading compiler. Attempt #1.

Contract: Pool

✓ success: `deposit`(single user, stake completely used) (859ms)
✓ success: `deposit`(claim pool fee) (1154ms)t #1
✓ success: `deposit`(two users, stake completely used) (2162ms)
✓ success: `deposit`(change fee after first fill) (687ms)
✓ success: `unstake pending`(common) (951ms)pt #1
✓ success: `unstake pending`(autocompound) (1138ms)
✓ success: `unstake pending`(all) (1344ms)empt #1
✓ fail: `unstake pending`(empty, common) (1161ms)
✓ fail: `unstake pending`(empty, autocompound) (789ms)
✓ fail: `unstake pending`(empty, all) (679ms)t #1
✓ fail: `unstake pending`(empty pending) (728ms)1

- ✓ fail: `unstake pending`(common not empty, autocompound) (614ms)
- ✓ fail: `unstake pending`(autocompound not empty, common) (695ms)
- ✓ fail: `deposit`(too low stake amount) (165ms)1
- ✓ success: `pending validators`(replace) (671ms)1

  Full replace of pending validator: 106120n. Attempt #1
- ✓ success: `validators`(pending -> deposited -> exited -> replaced) (925ms)
- ✓ Downloading compiler. Attempt #1.

  Contract: Pool
- ✓ success: initializationfrom solc-bin. Attempt #1
- ✓ Downloading compiler. Attempt #1.

  36 passing (2m)

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY BLAIZE SECURITY TEAM

Accounting

Deposit and withdraw under amount

Autocompound Accounting contract

- ✓ Should change contract balance after deposit (356ms)
- ✓ Should pending balance of user equal zero (327ms)
- ✓ Should return correct autocompound balance of user (406ms)
- ✓ Should transfer balance from maine to autocompound (459ms)
- ✓ Should change balance (215ms)

Deposit and withdraw under amount

Deposit

- ✓ Should change storage data (244ms)
- ✓ Should activated slots when amount equal '0'
- ✓ Should change balance when autocompound is false (101ms)
- ✓ Should change balance when autocompound is true (140ms)
- ✓ Should revert with 'Not owner or super admin'

Withdraw call

- ✓ Should change balance after withdraw pending (191ms)
- ✓ Should change balance after withdraw pending with different user accounts (170ms)
- ✓ Should revert with 'Not owner or super admin' (47ms)
- ✓ Should revert with custom error 'ZeroValue'

Withdraw

- ✓ Should change balances (211ms)
- ✓ Should change balance when Autocompound is false (153ms)
- ✓ Should should revert with custom error 'InvalidValue' (118ms)
- ✓ Should revert with 'Not owner or super admin'

Claim

- \- Should change balance after claim call
- ✓ Should revert with custom error 'ZeroValue'

Claim pool fee

- \- Should change balance after claim pool fee call
- ✓ Should revert with custom error 'ZeroValue'

User pending rewards

- ✓ Should change balance

```
    # Set fee
✓   Should revert with 'Zero fee'
    # autocompound
✓   Should emit event 'Autocompound'
    # autocompoundBalanceOf
    0 result
✓   Should be equal zero
    # fromAutocompoundToMain
✓   Should revert with custom error 'InvalidValue'
✓   Should revert with custom error 'ZeroValue'
    # fromMainToAutocompound
✓   Should revert with custom error 'ZeroValue'
✓   Should revert with custom error 'InvalidValue'
    # claimWithdrawRequest
✓   Should revert with custom error 'ZeroValue'
    # closeValidatorsStat
✓   Should to be equal 0
    # update
-   Should return boolean
    # Setters
✓   Should set new amount '
✓   Should revert with 'Not owner or super admin'
✓   Should change fee
✓   Should revert with 'Not owner or super admin'
✓   Should revert with custom error 'InvalidValue'
    # Getters
✓   Returns pending balance
✓   Returns pending balance of account
✓   Returns common balance of
✓   Returns pool fee
✓   Returns fee balance
    # Initialize
✓   Should revert with 'Initializable: contract is already initialized'
✓   Should revert with custom error 'ZeroValue' (131ms)
    Deposit and withdraw under amount
      # Deposit & withdraw specific amount
    User call stake with amount under min 5000000000000000
```

```
Transaction reverted with message: Stake too small

================================================
MIN STAKE
================================================
User1 call stake with min amount  10000000000000000
Transaction success. Min amount staked:  10000000000000000
Contract balance after min stake:  10000000000000000

================================================
1 ETH STAKE
================================================
User2 call stake with 1 ETH amount  10000000000000000000
Transaction success. Min amount staked:  10000000000000000000
Contract balance after 1 ETH stake:  10100000000000000000

================================================
32 ETH STAKE
================================================
User2 call stake with 32 ETH amount  32000000000000000000
Transaction success. Min amount staked:  32000000000000000000

================================================
BIG STAKE
================================================
User3 call stake with 1000 ETH   100000000000000000000
Transaction success. Big amount staked:  1000000000000000000000

================================================
CHECK BALANCE AFTER MIN & BIG STAKE
================================================
User2 balance before stake:  9968999724209997793680
User2 balance after stake:  9968999724209997793680
Pool contract balance before big stake:  31010000000000000000
Pool contract balance after big stake:  70100000000000000000
RewardsTreasury balance after stake:  0
WithdrawTreasury balance after stake:  0
```

DepositTest balance after stake: 10240000000000000000

==========
USTAKE WITH MIN AMOUNT
==========
User1 call unstake with min amount 10000000000000000
Transaction success. Min amount unstaked: 10000000000000000
Contract balance before unstake: 7010000000000000000
Contract balance after unstake: 7000000000000000000

User call same unstake one more time
Transaction reverted. Not enough withdrawable balance

==========
USTAKE WITH 1 ETH AMOUNT
==========
User2 call unstake with 1ETH amount 1000000000000000000
Transaction success. 1 ETH unstaked: 1000000000000000000
User2 balance before unstake: 9968999724209997793680
User2 balance after unstake: 996999956089499648716O
Contract balance before unstake: 7000000000000000000
Contract balance after unstake: 6000000000000000000

==========
PENDING USTAKE WITH 32 ETH AMOUNT
==========
User2 call unstakePending with 32ETH amount 32000000000000000(
✓ Should stake min and max amount (1489ms)

==========
MIN STAKE
==========
User1 call stake with min amount 10000000000000000
Transaction success. Min amount staked: 10000000000000000
Contract balance after min stake: 10000000000000000

==========

```
FIRST UNSTAKE WITH MIN AMOUNT
==============================================
User1 call unstake with min amount  10000000000000000
Transaction reverted with message: Not enough withdrawable bala
Contract should receive 32 ETH in total or more funds to unstake mir

==============================================
Add 32 ETH STAKE
==============================================
User2 call stake with amount: -  32000000000000000000
Transaction success. Big amount staked:  32000000000000000000

==============================================
Second UNSTAKE WITH MIN AMOUNT
==============================================
User1 call unstake with min amount  10000000000000000
Transaction success. Min amount unstaked:  10000000000000000
Contract balance before unstake:  1
Contract balance after unstake:  0

==============================================
Add more ETH STAKE
==============================================
User2 call stake with amount: -  32000000000000000000
Transaction success. 5 ETH staked:  32000000000000000000
Contract balance after stake:  4990000000000000001
Transaction success. 5 ETH unstaked:  32000000000000000000
User balance after unstake:  9999999372260994978088
Contract balance after unstake:  4990000000000000001
✓ Should stake min and max amount (434ms)
==============================================
1 ETH STAKE bu user1
==============================================
User1 call stake with min amount  10000000000000000
Transaction success. Min amount staked:  10000000000000000
Contract balance after min stake:  10000000000000000

==============================================
```

1 ETH STAKE by user3
==========
User1 call stake with min amount 1000000000000000000
Transaction success. Min amount staked: 1000000000000000000

==========
Add 32 ETH STAKE
==========
User2 call stake with amount: - 32000000000000000000
Transaction success. Big amount staked: 32000000000000000000

==========
Check balance of contract after stake in total 34 ETH
==========
Contract balance after stake: 2000000000000000000
Amount 32 ETH transferred as staker balance

==========
First UNSTAKE WITH 0.5 ETH AMOUNT
==========
User1 call unstake with 0,5 eth amount 500000000000000000
Contract balance after unstake: 1500000000000000000

==========
Add 31 ETH STAKE
==========
User2 call stake with amount: - 31000000000000000000
Transaction success. Big amount staked: 31000000000000000000
Contract balance after stake: 500000000000000000
Amount 32 ETH transferred as staker balance

==========
Second UNSTAKE WITH 1 ETH AMOUNT
==========
User3 call unstake with 1 eth amount 1000000000000000000
Contract balance after unstake: 0

```
Amount 32 ETH transferred as staker balance

================================================
Second UNSTAKE WITH 1 ETH AMOUNT
================================================
User3 call unstake with 1 eth amount  1000000000000000000
Contract balance after unstake:  0

================================================
Add 10 ETH STAKE
================================================
User2 call stake with amount: -  10000000000000000000
Transaction success. Big amount staked:  10000000000000000000
Contract balance after stake:  9500000000000000000

Amount 0.5 ETH transferred as withdrawn by user3

================================================
Third UNSTAKE WITH 0.5 ETH AMOUNT
================================================
User3 call unstake with 0,5 eth amount  500000000000000000
Transaction reverted. Not enough withdrawable balance
Contract balance after unstake:  9500000000000000000
User3 balance before claimWithdrawRequest:  99994995910389967283
User3 balance after claimWithdrawRequest:  9999999516325996130608
Treasury balance after claimWithdrawRequest:  0
✓ Should stake 1 eth and unstake 1 eth  (708ms)

================================================
1 ETH STAKE
================================================
User1 call stake with 1 ETH amount  1000000000000000000
Transaction success. Min amount staked:  1000000000000000000
Contract balance before 1 ETH stake:  0
Contract balance after 1 ETH stake:  1000000000000000000
User balance before 1 ETH stake:  10000000000000000000000
User balance after 1 ETH stake:  9998999761440998091528
```

```
==========================================
30 ETH STAKE
==========================================
User2 call stake with 30 ETH amount  30000000000000000000
Transaction success. 30 ETH amount staked:  30000000000000000000
Contract balance before 30 ETH stake:  10000000000000000000
Contract balance after 30 ETH stake:  31000000000000000000
User balance before 30 ETH stake:  100000000000000000000000
User balance after 30 ETH stake:  996999984905299879242 4

==========================================
PENDING UNSTAKE WITH 30 ETH AMOUNT
==========================================
User2 call unstakePending with 30ETH amount  30000000000000000000
pendingBalance:  30000000000000000000
Contract balance before unstake:  31000000000000000000
Contract balance after unstake:  16000000000000000000
Transaction reverted. User try unstake more than he has in pending balan
```

- ✓ Should call unstakePending and receive a pending balance (224ms)

Medium check

Flow

- ✓ Set validator
- ✓ 3 users deposits total 12 ETH (54ms)
- ✓ Go to next round (autocompound balance will be staked balance) (50ms)
- ✓ Send tokens to reward (autocompound balance will increase)
- ✓ User1 unstakes tokens (63ms)

Pool

Main

- ✓ Stake
- ✓ Unstake pending
- ✓ Unstake (86ms)
- ✓ Unstake 32 ETH (57ms)

Validators

- ✓ Set pending validators
- ✓ Validators added after stake > 32 ETH (55ms)
- ✓ Replace pending validators

- ✓ Mark as exited validator (47ms)

Governor

- ✓ Set new governor
- ✓ Set new governor as superAdmin

Ownable

- ✓ Renounce ownership
- ✓ Transfer ownership

Revert

- ✓ When try to initialize contract second time
- ✓ When try to initialize if WithdrawTreasury is zero address
- ✓ When try to initialize if DepositContract is zero address
- ✓ When try to initialize if Accounting is zero address
- ✓ When try to stake if staking is paused
- ✓ When try to stake if not met min amount
- ✓ When try to stake 32 ETH if no pending validators
- ✓ When try to unstake pending if withdraw is paused
- ✓ When try to unstake pending 0 amount
- ✓ When not governor tries to set pending validators
- ✓ When try to set pending validators if pubkey !== 48
- ✓ When try to set pending validators if signature !== 96
- ✓ When try to set pending validators if already added
- ✓ When try to get pending validator if wrong index
- ✓ When try to get validator if wrong index
- ✓ When not governor tries to replace pending validator
- ✓ When try to replace pending validator if wrong index
- ✓ When not governor tries to mark validator as exited
- ✓ When try to mark validator as exited if not deposited status
- ✓ When try to mark validator as exited num > len
- ✓ When not governor tries to pause staking
- ✓ When not governor tries to pause withdraw
- ✓ When not governor tries to set governor
- ✓ When not governor tries to set min staking amount
- ✓ When ETH transfer was not successfull (73ms)

Scenario -- Token flow

Flow

- ✓ Set validator
- ✓ User1 deposits 11 ETH

- ✓ User2 deposits 25 ETH autocompound
- ✓ User1 tries to unstake (gets 4 ETH back immediately, 7 ETH requested)
- ✓ Owner stakes 10 ETH (7 ETH goes to WithdrawTreasury, 3 ETH goes to Pool
- ✓ User1 unstakes 7 ETH

RewardsTreasury

Deployment

- ✓ Sets the initial owner when initialization (56ms)

Action

Setting the pool

- ✓ Sets
- ✓ Prevents non-owners from setting

Restaking

- ✓ Restakes (51ms)
- ✓ Prevents non-rewarders from restaking

[Via WithdrawTreasury] TreasuryBase

Setting the rewarder

- ✓ Sets
- ✓ Prevents non-owners from setting

Sending Ether

- ✓ Sends
- ✓ Reverts when sending if an unsuccessful low-level call
- ✓ Prevents non-rewarders from sending

WithdrawTreasury

Deployment

- ✓ Sets the initial owner when initialization

Withdrawer

Adding a withdrawal request

- ✓ Adds (46ms)
- ✓ Adds if a number of validators to close is equal to the expected number (47ms)
- ✓ Adds if a zero number of validators to close
- ✓ Does not add if a zero amount

Interchange withdrawal

- ✓ Interchanges (40ms)
- ✓ Does not interchanges if an allowed amount is zero (44ms)

Updating the expected number of close validators

- ✓ Updates (46ms)

Calculation of validator's closure
✓ Calculates
✓ Calculates if no expected validators
✓ Calculates if the balance change is less than the beacon amount
Claim of a withdrawal request
✓ Claims (51ms)
✓ Reverts when claiming if zero amount to claim
Calculation of an amount of a withdrawal request
✓ Calculates
✓ Calculates if an amount to withdraw from pending is greater than the pending share
✓ Calculates if zero pending and zero withdrawal from pending
Getters
✓ Returns parameters of the queue of withdrawal requests
✓ Returns a withdrawal request

[Library] WithdrawRequests

Adding a request
✓ Adds
✓ Adds when pushing a new request
✓ Adds when pushing a new request if there are two requests (43ms)
✓ Adds when replacing a claimed request
Claim of a request
✓ Claims
✓ Claims if zero amount
✓ Claims if there is a claimed request (44ms)
✓ Claims if there is a request with a greater amount (39ms)
Getting total and claimable amounts
✓ Returns
✓ Returns if there is a claimed request (38ms)

[Library] Math

✓ Returns the smaller of two numbers if a is greater than b
✓ Returns the smaller of two numbers if b is greater than a
✓ Returns a number if equal numbers

OwnableWithSuperAdmin

Initialization
✓ Sets the deployer as the initial owner when initialization (81ms)
✓ Reverts when re-initialization (77ms)

Action

Transfer of ownership to the zero address

- ✓ Resets the owner by the owner itself
- ✓ Resets the owner by the super administrator
- ✓ Prevents non-owners and non-administrators from resetting

Transfer of ownership

- ✓ Transfers by the owner
- ✓ Transfers by the super administrator
- ✓ Reverts when transfer if a new owner is the zero address
- ✓ Prevents non-owners and non-administrators from transfer

Setting the super administrator

- ✓ Sets if the current one is the zero address
- ✓ Sets by the super administrator
- ✓ Does not set if the caller is not the super administrator which is not the zero address

Getters

- ✓ Returns the owner
- ✓ Returns the super administrator

+55 passing (3s)

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
AutocompoundAccounting.sol	92.31	77.78	100
CommonAccounting.sol	78.57	76.14	94.12
Pool.sol	100	100	100
Accounting.sol	92.11	82.95	97.06
RewardsTreasury.sol	100	100	100
TreasuryBase.sol	100	100	100
Withdrawer.sol	100	100	100
WithdrawTreasury.sol	100	100	100
OwnableWithSuperAdmin.sol	100	88.89	100
Math.sol	100	100	100

FILE	% STMTS	% BRANCH	% FUNCS
ValidatorList.sol	96.55	75	100
takerAccount.sol	95	87.5	100
WithdrawRequests.sol	100	100	100
All files	**94.54**	**89.19**	**98.54**

DISCLAIMER

The information presented in this report is an intellectual property of the customer, including all the presented documentation, code databases, labels, titles, ways of usage, as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else's requirements and be fully secure, complete, accurate, and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions, and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool, which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.