Everstake Ethereum Staking Protocol

Security Assessment (Summary Report)

January 30, 2025

Prepared for:
Everstake

Prepared by: **Samuel Moelius and Anish Naik**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Sam Greenup, Project Manager
sam.greenup@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Samuel Moelius, Consultant
sam.moelius@trailofbits.com

Anish Naik, Consultant
anisk.naik@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
November 14, 2024	Pre-project kickoff call
December 10, 2024	Delivery of report draft
December 10, 2024	Report readout meeting
January 16, 2025	Added appendix D: Fix Review Results
January 21, 2025	Adjusted appendix D per client's request
January 22, 2025	Delivery of final summary report
January 28, 2025	Added appendix F: Non-Security-Related Findings Fix Review Results
January 30, 2025	Added Fix Review section to Executive Summary

Project Targets

The engagement involved a review and testing of the following target.

ETH Staking B2C SC

Repository	https://github.com/everstake/ETH-Staking-B2C-SC
Version	`3a3fee5d8b284dd60cf156264cdfb1182716cfd1`
Type	Solidity
Platform	Ethereum

Executive Summary

Engagement Overview

Everstake engaged Trail of Bits to review the security of its Ethereum staking protocol (3a3fee5). The protocol consists of several smart contracts that hold user deposits and alert Everstake infrastructure when validators should come online and go offline.

A team of two consultants conducted the review from November 25 to December 6, 2024, for a total of three engineer-weeks of effort. With full access to source code and documentation, we performed static and dynamic testing of the codebase, using automated and manual processes.

Observations and Impact

The goals of the engagement were to answer questions such as the following, which were raised during the kickoff call:

- Can a balance be improperly changed?
- Can a user pose as an administrator and change a protocol limitation?

Our main concerns involve the project's overall structure. There are multiple undocumented contracts that serve only as base contracts. Because they are undocumented, it is difficult to tell why functions reside within them. Similarly, given a function in, say, the `Accounting` contract, it is difficult to predict whether the function resides directly in `Accounting` or in one of its base contracts.

Many of the contracts' functions have no documentation. For example, `Withdrawer.sol` contains eight functions; however, only one has an associated comment (figure 1). For the functions that lack documentation, the only way to determine their purposes is from the functions' signatures and their implementations.

```
    /// @dev Interchange amount with max allowed to interchange amount
    function _interchangeWithdraw(uint256 amount) internal returns (uint256
interchangedAmount) {
```

Figure 1: The one function in `Withdrawer.sol` with an associated comment (`ETH-Staking-B2C-SC/contracts/Withdrawer.sol#78–79`)

For the functions that have documentation, the documentation often presumes that the reader knows the definitions of various nonstandard terms such as "autocompound" and "interchange" (see figure 1 above).

Taking all of the above points into account, manually reviewing this codebase is very challenging.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Everstake take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or any refactor that may occur when addressing other recommendations.

- **Document every function.** Write down each function's purpose and how it achieves that purpose. Doing so will aid readers of the code. The exercise could also expose bugs by revealing functions that do not achieve their intended purposes.

Fix Review

From January 15 to January 16, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the Everstake team for the issues identified in this report. The results are in appendix D. At Everstake's request, we also reviewed the team's implementations of the recommendations in appendix C: Non-Security-Related Findings. The results are in appendix F.

The latest changes conducted during the fix review are represented at commit `6cdc41b`.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The project uses Solidity 0.8, which includes overflow checks.	**Satisfactory**
Auditing	The code features prominent use of events. Events are emitted under important conditions and with pertinent information.	**Satisfactory**
Authentication / Access Controls	There are numerous privileged roles (e.g., fee claimer, governor, owner, rewarder, and super admin). The ways in which they relate are not clear and are not documented.	**Moderate**
Complexity Management	The project uses archived tools and outdated dependencies. Structural design decisions lack a clear purpose and may hamper the project's maintainability. For example, the owner and super admin roles are handled by the `OwnableWithSuperAdmin` contract. The governor role is handled by the `Governor` contract. The fee claimer and rewarder roles are handled directly by the `Accounting` and `TreasuryBase` contracts (respectively).	**Moderate**
Cryptography and Key Management	Messages are constructed and hashed before the protocol interacts with the Ethereum deposit contract. However, we found no issues related to the protocol's use of hashing.	**Satisfactory**
Decentralization	As mentioned under "Authentication / Access Controls," the project features privileged roles and thus is centralized by design.	**Not Applicable**

Documentation	Code comments are sparse and assume the reader knows the definitions of nonstandard terms such as "autocompound" and "interchange." A `docs` directory gives summaries of several of the contracts' functions; however, it is out of sync with the code. There is also a `diagrams` directory, though it is difficult to relate the contained diagrams to the code. The documentation does not describe the relationships among the privileged roles (fee claimer, governor, owner, rewarder, and super admin). Much of the documentation contains grammatical and spelling errors.	**Weak**
Low-Level Manipulation	Low-level manipulation is used to construct messages hashed for the Ethereum deposit contract. However, we found no problems related to this use.	**Satisfactory**
Testing and Verification	Because an old version of Truffle is used, test coverage cannot be computed. Thus, it is impossible to judge the quality of the tests.	**Further Investigation Required**
Transaction Ordering	We found no issues related to transaction ordering.	**Satisfactory**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Project uses archived and outdated tools	Undefined Behavior	**Informational**
2	Project uses outdated dependencies	Undefined Behavior	**Informational**
3	Documentation is out of sync with the code	Patching	**Informational**
4	Documentation lacks a glossary	Patching	**Informational**
5	Duplicated code	Undefined Behavior	**Informational**
6	Storage variables updated by multiple contracts in the inheritance tree	Undefined Behavior	**Informational**
7	_update may fail to update REWARDER_BALANCE_POSITION	Undefined Behavior	**Informational**

Detailed Findings

1. Project uses archived and outdated tools	
Severity: **Informational**	Difficulty: **Undetermined**
Type: Undefined Behavior	Finding ID: TOB-EVERSTAKE-1
Target: `package.json`	

Description

The project uses Truffle, an archived tool, to build and test contracts (figure 1.1). Moreover, the version of Truffle used (5.7.5; see figure 1.2) is not the most recent one (which is 5.11.5). Use of an archived or outdated tool has several downsides:

- Flaws could be introduced into the compiled code. Even if such flaws were discovered, they might not be fixed.
- Bugs could be missed during testing because the tool does not run the tests correctly.
- Modern features such as test coverage and logging to the console from contracts are not available.

⚠️ The Truffle Suite is being sunset. For information on ongoing support, migration options and FAQs, visit the Consensys blog. Thank you for all the support over the years.

Figure 1.1: Message on Truffle's GitHub repository

```
  "truffle": {
    "version": "5.7.5",
    "resolved": "https://registry.npmjs.org/truffle/-/truffle-5.7.5.tgz",
    "integrity":
"sha512-JA2/ISQ1fkgozYdmDnOubHkfSpsMeRK17OgL+nI2r9jKrnGIt1NjLDCmVGuuBrz4BBZpgZAstZhk
L6cEJiwYNQ==",
    "requires": {
      "@truffle/db": "^2.0.14",
      "@truffle/db-loader": "^0.2.14",
      "@truffle/debugger": "^11.0.25",
      "app-module-path": "^2.2.0",
      "ganache": "7.7.3",
      "mocha": "10.1.0",
      "original-require": "^1.0.1"
    }
22026    },
```

Figure 1.2: The version of Truffle currently used by Everstake (`ETH-Staking-B2C-SC/package-lock.json#L22013–L22026`)

Exploit Scenario

Alice, an Everstake developer, believes that all of the Everstake contracts' code is tested. However, she is unable to confirm this because she is using a Truffle/Solidity combination that does not support test coverage. Alice deploys the contracts, and a bug is found in part of the code that was untested. Everstake suffers financial loss.

Recommendations

Short term, migrate to Hardhat or Foundry. Hardhat and Foundry are modern Solidity testing tools and are maintained.

Long term, regularly compute the project's test coverage. Doing so will allow Everstake to ensure that all important conditions are tested.

References

- Truffle Suite: Migrate to Hardhat
- Hardhat: Migrating from Truffle

2. Project uses outdated dependencies

Severity: **Informational**	Difficulty: **Undetermined**
Type: Undefined Behavior	Finding ID: TOB-EVERSTAKE-2
Target: `package-lock.json`	

Description

The project uses version 4.8.1 of the OpenZeppelin contracts (figure 2.1). However, the following more recent versions exist. Since silent bug fixes are common, projects should strive to use the most recent versions of their dependencies whenever possible.

```
 13618    "@openzeppelin/contracts": {
 13619      "version": "4.8.1",
 13620      "resolved":
"https://registry.npmjs.org/@openzeppelin/contracts/-/contracts-4.8.1.tgz",
 13621      "integrity":
"sha512-xQ6eUZl+RDyb/FiZe1h+U7qr/f4p/SrTSQcTPH2bjur3C5DbuW/zFgCU/b1P/xcIaEqJep+9ju4x
DRi3rmChdQ=="
 13622    },
 13623    "@openzeppelin/contracts-upgradeable": {
 13624      "version": "4.8.1",
 13625      "resolved":
"https://registry.npmjs.org/@openzeppelin/contracts-upgradeable/-/contracts-upgradea
ble-4.8.1.tgz",
 13626      "integrity":
"sha512-1wTv+20lNiC0R07jyIAbHU7TNHKRwGiTGRfiNnA8jOWjKT98g5OgLpYWOi40Vgpk8SPLA9EvfJAb
AeIyVn+7Bw=="
 13627    },
```

Figure 2.1: Version of the OpenZeppelin contracts currently used by Everstake (`ETH-Staking-B2C-SC/package-lock.json#L13618–L13627`)

The following more recent versions of the OpenZeppelin contracts exist.

- 4.8.3 (greater patch version)
- 4.9.3 (greater minor version)
- 5.1.0 (greater major version)

Exploit Scenario

Mallory notices that the Everstake contracts execute OpenZeppelin code that was patched since version 4.8.1. Mallory exploits the flaw and drains funds from the Everstake contracts.

Recommendations
Short term, upgrade the OpenZeppelin contracts to the most recent version feasible. If a version cannot be used because of an API change, document the problem and develop a plan to achieve compatibility with the new API. Taking these steps will ensure that the Everstake contracts use the most up-to-date versions of the OpenZeppelin contracts.

Long term, regularly check the OpenZeppelin contracts for new releases. When new releases become available, update the Everstake contracts to use them. Doing so will avoid delays in the Everstake contracts receiving bug fixes.

3. Documentation is out of sync with the code	
Severity: **Informational**	Difficulty: **High**
Type: Patching	Finding ID: TOB-EVERSTAKE-3
Target: `docs` directory, various source files	

Description

The project includes a `docs` directory with two markdown files, one for the `Pool` contract and one for the `Accounting` contract. The markdown files contain summaries of many of the contracts' functions. However, the markdown files are out of sync with the code they represent.

Figure 3.1 shows the `Pool` contract's functions (left) along with what is reflected in the markdown file (right). Note that nine functions in the contract are not reflected in the markdown file (green). Furthermore, the functions `getValidator` and `getValidatorCount`, referred to by the markdown file, do not exist in the contract (red).

`activateStake`	`activateStake`
`getPendingValidator`	`getPendingValidator`
`getPendingValidatorCount`	`getPendingValidatorCount`
	`getValidator`
	`getValidatorCount`
`getValidatorsStash`	
`minStakeAmount`	
`pauseStaking`	
`pauseWithdraw`	
`replacePendingValidator`	
`restake`	
`setGovernor`	
`setMinStakeAmount`	
`setPendingValidators`	
`stake`	`stake`
`unstake`	`unstake`
`unstakePending`	`unstakePending`

Figure 3.1: External `Pool` contract functions (left) and external `Pool` contract functions referred to by the documentation (right)

Figure 3.2 shows the `Accounting` contract's functions (left) along with what is reflected in the markdown file (right). Note that 17 functions in the contract are not reflected in the markdown file (green).

activateBalance	
activateValidators	
autocompound	autocompound
autocompoundBalanceOf	
balance	balance
claimPoolFee	
claimWithdrawRequest	claimWithdrawRequest
closeValidatorsStat	closeValidatorsStat
deposit	
depositedBalanceOf	depositedBalanceOf
fastenValidatorsToStop	
feeBalance	
getPoolFee	getPoolFee
pauseRewardsUpdate	
pauseWithdrawClaim	
pendingBalance	pendingBalance
pendingBalanceOf	pendingBalanceOf
pendingDepositedBalance	pendingDepositedBalance
pendingDepositedBalanceOf	pendingDepositedBalanceOf
pendingRestakedRewardOf	pendingRestakedRewardOf
pendingRestakedRewards	pendingRestakedRewards
readyforAutocompoundRewardsAmount	readyforAutocompoundRewardsAmount
restakedRewardOf	
setFee	
setFeeClaimer	
setGovernor	
setMinRestakeAmount	
update	
withdraw	
withdrawPending	
withdrawRequest	withdrawRequest
withdrawRequestQueueParams	withdrawRequestQueueParams

Figure 3.2: External `Accounting` contract functions (left) and external `Accounting` contract functions referred to by the documentation (right)

Exploit Scenario

Alice, an Everstake developer, is tasked with making a change to an Everstake contract function. The relevant function is not reflected in the `docs` directory. Alice tries to understand the function's inner workings, but does so incorrectly. Alice's change introduces a bug into the function.

Recommendations

Short term, update the contents of the `docs` directory so that they reflect the `Pool` and `Accounting` contracts in their present form. Documentation must be kept up to date to be of value.

Long term, investigate a solution for automatically preparing documentation from doc comments (e.g., with `OpenZeppelin/solidity-docgen`). Such a solution would allow documentation to be written in one place (i.e., the code). Moreover, being able to automatically generate documentation would reduce the likelihood that it will get out of sync with the code.

References

- `OpenZeppelin/solidity-docgen`

4. Documentation lacks a glossary

Severity: **Informational**	Difficulty: **High**
Type: Patching	Finding ID: TOB-EVERSTAKE-4
Target: Various source files	

Description

Many comments assume that the reader knows the definitions of nonstandard terms. Having a glossary that defines such terms would help to avoid confusion.

Examples of presumptive comments appear in figures 4.1 through 4.3.

```
    /// @dev Simulate full cycle of autocompound
    function _simulateAutocompound() private view returns (uint256
totalPoolBalance, uint256 pendingRestaked, uint256 pendingAmount, uint256
activePendingRound, WithdrawRequestQueue memory queue) {
```

Figure 4.1: The comment assumes the reader knows the definition of "autocompound." (`ETH-Staking-B2C-SC/contracts/Accounting.sol#L272–L273`)

```
    /// @dev Withdraw from pending balance
    function _withdrawFromPending(address account, uint256 amount, uint256
activePendingRound, uint256 activatedRoundsNum) internal returns (uint256
pendingBalance){
```

Figure 4.2: The comment assumes the reader knows the definition of "pending balance." (`ETH-Staking-B2C-SC/contracts/AutocompoundAccounting.sol#L256–L257`)

```
    /// @dev Interchange amount with max allowed to interchange amount
    function _interchangeWithdraw(uint256 amount) internal returns (uint256
interchangedAmount) {
```

Figure 4.3: The comment assumes the reader knows the definition of "interchange." (`ETH-Staking-B2C-SC/contracts/Withdrawer.sol#L78–L79`)

A glossary should include definitions of at least the following terms:

- Active
- Autocompound
- Claim/claimed
- Deposit/deposited
- Fill/filled
- Interchange

- Known/unknown (as in “known/unknown validator”)
- Pending
- Round
- Share
- Stake/restake/unstake
- Withdraw/withdrawal

Exploit Scenario
The exploit scenario is essentially that of TOB-EVERSTAKE-3.

Recommendations
Short term, prepare a glossary with definitions of each of the terms in the bulleted list above. Giving readers such a resource will reduce the likelihood that comments will cause confusion.

Long term, regularly review language used in comments and update the glossary as appropriate. Documentation must be kept up to date to be of value.

5. Duplicated code

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-EVERSTAKE-5
Target: `contracts/{utils/OwnableWithSuperAdmin.sol`, `TreasuryBase.sol}`	

Description

The `OwnableWithSuperAdmin` and `TreasuryBase` contracts both have a notion of an owner. The code to handle their respective owners is duplicated across the two contracts. Duplicated code can lead to situations where a bug is fixed in one copy of the code but not in the other.

```
/**
 * @dev Leaves the contract without
owner. It will not be possible to call
 * `onlyOwner` functions anymore. Can
only be called by the current owner.
 *
 * NOTE: Renouncing ownership will leave
the contract without an owner,
 * thereby removing any functionality
that is only available to the owner.
 */
function renounceOwnership() external
virtual onlyOwner {
    address prevOwner = _owner;
    delete _owner;
    delete _pendingOwner;
    emit OwnershipTransferred(prevOwner,
address(0));
}




/**
 * @dev Returns the address of the
pending owner.
```

```
/**
 * @dev Leaves the contract without
owner. It will not be possible to call
 * `onlyOwner` functions anymore. Can
only be called by the current owner.
 *
 * NOTE: Renouncing ownership will leave
the contract without an owner,
 * thereby removing any functionality
that is only available to the owner.
 */
function renounceOwnership() external
virtual onlyOwner {
    address prevOwner = _owner;
    delete _owner;
    delete _pendingOwner;
    emit OwnershipTransferred(prevOwner,
address(0));
}

// LINES 72-89 OMITTED

/**
 * @dev Returns the address of the
pending owner.
*/
```

```
 */
function pendingOwner() public view
virtual returns (address) {
    return _pendingOwner;
}

/**
 * @dev Transfers ownership of the
contract to a new account (`newOwner`).
 * Can only be called by the current
owner.
 */
function transferOwnership(address
newOwner) public virtual onlyOwner {
    if (newOwner == address(0)) revert
Errors.ZeroValue("newOwner");

    _pendingOwner = newOwner;
    emit
OwnershipTransferStarted(owner(),
newOwner);
}

/**
 * @dev Transfers ownership of the
contract to a new account (`newOwner`)
and deletes any pending owner.
 * Internal function without access
restriction.
 */
function _transferOwnership(address
newOwner) internal virtual {
    delete _pendingOwner;
    address prevOwner = _owner;
    _owner = newOwner;
    emit OwnershipTransferred(prevOwner,
newOwner);
}

/**
 * @dev The new owner accepts the
ownership transfer.
```

```
function pendingOwner() public view
virtual returns (address) {
    return _pendingOwner;
}

/**
 * @dev Starts the ownership transfer of
the contract to a new account. Replaces
the pending transfer if there is one.
 * Can only be called by the current
owner.
 */
function transferOwnership(address
newOwner) public virtual onlyOwner {
    if (newOwner == address(0)) revert
Errors.ZeroValue("newOwner");
    _pendingOwner = newOwner;
    emit OwnershipTransferStarted(_owner,
newOwner);
}

/**
 * @dev Transfers ownership of the
contract to a new account (`newOwner`)
and deletes any pending owner.
 * Internal function without access
restriction.
 */
function _transferOwnership(address
newOwner) internal virtual {
    delete _pendingOwner;
    address prevOwner = _owner;
    _owner = newOwner;
    emit OwnershipTransferred(prevOwner,
newOwner);
}

/**
 * @dev The new owner accepts the
ownership transfer.
 */
 */
function acceptOwnership() public virtual
{
    address sender = _msgSender();
    if (pendingOwner() != sender) revert
Errors.InvalidParam("sender");

    _transferOwnership(sender);
}
```

```
function acceptOwnership() public virtual
{
    address sender = msg.sender;
    if (pendingOwner() != sender) revert
Errors.InvalidParam("sender");

    _transferOwnership(sender);
}
```

Figure 5.1: ETH-Staking-B2C-SC/contracts/utils/OwnableWithSuperAdmin.sol#L59-L110 (left) and ETH-Staking-B2C-SC/contracts/TreasuryBase.sol#L58-L127 (right)

Exploit Scenario

Alice, an Everstake developer, is tasked with fixing a bug in the ownership code. Alice fixes the bug in the `OwnableWithSuperAdmin` contract but does not realize the ownership code also exists in `TreasuryBase`. The bug persists in `TreasuryBase`.

Recommendations

Short term, take the following steps:

- Remove the ownership code from `TreasuryBase`.
- Remove the `OwnableWithSuperAdmin` contract entirely.
- Create two contracts: `Ownable`, which captures the "ownable" aspects of `OwnableWithSuperAdmin`, and `SuperAdmin`, which captures the "super admin" aspects of `OwnableWithSuperAdmin`.
- Have `TreasuryBase` descend from `Ownable`.
- Have each contract that descended from `OwnableWithSuperAdmin` descend from both `Ownable` and `SuperAdmin`.

Taking these steps will eliminate the present code duplication and the possibility that a bug is fixed in one location but not in all.

Long term, as new code is added to the codebase, resist the urge to copy existing code. Instead, look for opportunities to consolidate code that already exists.

6. Storage variables updated by multiple contracts in the inheritance tree	
Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-EVERSTAKE-6
Target: `contracts/{Accounting.sol`, `AutocompoundAccounting.sol}`	

Description

Several of the protocol's contracts use string hashes to determine slots in which to store values. Two such contracts are `Accounting` and `AutocompoundAccounting`. Note that the former inherits from the latter. Some storage locations are modified by both of these contracts. While we do not believe there is a problem currently, this practice could lead to a situation where a storage variable is overwritten by both `Accounting` and its parent contract, `AutocompoundAccounting`.

Storage variables modified by both contracts include the following:

- `PENDING_RESTAKED_VALUE_POSITION`

  Modified by `Accounting` in the following locations:

  - `ETH-Staking-B2C-SC/contracts/Accounting.sol#L234`
  - `ETH-Staking-B2C-SC/contracts/Accounting.sol#L494`

  Modified by `AutocompoundAccounting` in the following location:

  - `ETH-Staking-B2C-SC/contracts/AutocompoundAccounting.sol#L143`

- `POOL_PENDING_BALANCE_POSITION`

  Modified by `Accounting` in the following locations:

  - `ETH-Staking-B2C-SC/contracts/Accounting.sol#L177`
  - `ETH-Staking-B2C-SC/contracts/Accounting.sol#L224`
  - `ETH-Staking-B2C-SC/contracts/Accounting.sol#L557`

  Modified by `AutocompoundAccounting` in the following locations:

  - `ETH-Staking-B2C-SC/contracts/AutocompoundAccounting.sol#L142`

    ○ ETH-Staking-B2C-SC/contracts/AutocompoundAccounting.sol#L258

- TOTAL_BALANCE_POSITION

  Modified by Accounting in the following location:

    ○ ETH-Staking-B2C-SC/contracts/Accounting.sol#L238

  Modified by AutocompoundAccounting in the following locations:

    ○ ETH-Staking-B2C-SC/contracts/AutocompoundAccounting.sol#L340

    ○ ETH-Staking-B2C-SC/contracts/AutocompoundAccounting.sol#L359

To help illustrate why this could become a problem, consider the last case: the writes to TOTAL_BALANCE_POSITION. Writes to this storage variable occur in Accounting's _depositAccount function (figure 6.1). However, _depositAccount calls _deposit (line 242 in figure 6.1), which calls _mintToUser (line 128 in figure 6.2), which calls _mint (line 346 in figure 6.3), which modifies TOTAL_BALANCE_POSITION. Note that the return on line 239 (figure 6.1) prevents conflict in this case. Nonetheless, if care is not exercised, one risks overwriting one function's changes with another's.

```
 231    function _depositAccount(address account, uint256 interchangedAmount, uint256 depositToPendingValue, uint256 activePendingRound, uint256 activatedRoundsNum, bool isRewardsAutocompound) private {
 232        if (isRewardsAutocompound) {
 233            if (depositToPendingValue > 0) {
 234 PENDING_RESTAKED_VALUE_POSITION.setStorageUint256(PENDING_RESTAKED_VALUE_POSITION.getStorageUint256() + depositToPendingValue);
 235            }
 236
 237            // Add origin amount
 238 TOTAL_BALANCE_POSITION.setStorageUint256(TOTAL_BALANCE_POSITION.getStorageUint256() + (interchangedAmount + depositToPendingValue));
 239            return;
 240        }
 241
 242        _deposit(account, interchangedAmount, depositToPendingValue, activePendingRound, activatedRoundsNum);
 243        emit DepositPending(account, depositToPendingValue);
 244    }
```

Figure 6.1: The definition of _depositAccount (ETH-Staking-B2C-SC/contracts/Accounting.sol#L231–L244)

```
 119    function _deposit(address sourceAccount, uint256 amount, uint256 pendingAmount, uint256 activePendingRound, uint256 activatedRoundsNum) internal {
        if (pendingAmount + amount == 0) {
            return;
        }

        StakerAccount storage sourceStaker = _refreshedAccount(sourceAccount, activePendingRound, activatedRoundsNum);
        sourceStaker.depositedBalance += (pendingAmount + amount);

        if (amount > 0){
            _mintToUser(sourceStaker, amount);
        }

        if (pendingAmount > 0){
            sourceStaker.pendingBalance.balance += pendingAmount;
            sourceStaker.pendingBalance.period = activePendingRound;
        }
    }
```

Figure 6.2: The definition of `_deposit`
(`ETH-Staking-B2C-SC/contracts/AutocompoundAccounting.sol#L119–L135`)

```
    function _mintToUser(StakerAccount storage staker, uint256 amount) private {
        uint256 totalPoolBalance = TOTAL_BALANCE_POSITION.getStorageUint256();
        uint256 totalMintedShare = TOTAL_MINTED_SHARE_POSITION.getStorageUint256();
        uint256 share = _mint(totalPoolBalance, totalMintedShare, amount);
        // Case when amount <= 1 share
        if (share == 0) {
            return;
        }

        staker.share += share;
    }

    function _mint(uint256 totalPoolBalance, uint256 totalMintedShare, uint256 amount) private returns (uint256 share) {
        share = _amountToShare(amount, totalMintedShare, totalPoolBalance);

        // Update total share to consist total balance
        TOTAL_BALANCE_POSITION.setStorageUint256(totalPoolBalance + amount);

        // Case when amount <= 1 share
        if (share == 0) {
            return share;
        }

        TOTAL_MINTED_SHARE_POSITION.setStorageUint256(totalMintedShare + share);
        return share;
    }
```

Figure 6.3: The definitions of `_mintToUser` and `_mint`
(`ETH-Staking-B2C-SC/contracts/AutocompoundAccounting.sol#L343–L368`)

Exploit Scenario
Alice, an Everstake developer, is tasked with adding a new feature to the `Accounting` contract. The feature requires overwriting a storage variable. Unbeknownst to Alice, the storage variable is also overwritten by the `AutocompoundAccounting` contract. The conflict is uncovered after the change is deployed.

Recommendations
Short term, refactor the code so that each storage variable is maintained by exactly one contract. Doing so will reduce the likelihood that one contract overwrites the changes of another.

Long term, as new code is added to the codebase, ensure that the above standard is maintained. That is, ensure that each storage variable is modified by at most one contract.

7. _update may fail to update REWARDER_BALANCE_POSITION

Severity: **Informational**	Difficulty: **Undetermined**
Type: Undefined Behavior	Finding ID: TOB-EVERSTAKE-7
Target: `contracts/{Accounting.sol`, `Withdrawer.sol}`	

Description

The function `_update` reads `REWARDER_BALANCE_POSITION`, updates it in memory, and then writes the updated value back to storage (figure 7.1). Between the in-memory update and the write to storage, several operations are performed, including a transfer and an early return. The late write of `REWARDER_BALANCE_POSITION` could allow an attacker to perform the transfer and the early return repeatedly to drain the protocol of funds.

```
609    uint256 rewarderBalance = REWARDER_BALANCE_POSITION.getStorageUint256();
610    if (currentRewarderBalance == rewarderBalance) {
611        return;
612    }
613
614    uint256 balanceDiff = currentRewarderBalance - rewarderBalance;
615    uint256 chargedBalance = _closedValidatorStop(balanceDiff);
616    if (chargedBalance > 0) {
617        balanceDiff -= chargedBalance;
618        currentRewarderBalance -= chargedBalance;
619        // Send amount from rewards treasury to withdraw treasury
620
IRewardsTreasury(rewardsTreasury).sendEth(WITHDRAW_TREASURY_POSITION.getStorageAddre
ss(), chargedBalance);
621    }
622
623    // Not update if nothing on deposit
624    if (TOTAL_BALANCE_POSITION.getStorageUint256() == 0){
625        return;
626    }
627
628    (uint256 rewardsFee, uint256 rewards) =
629    _calculateBalanceChanges(balanceDiff);
630    if (rewards == 0 && rewardsFee == 0) {
631        return;
632    }
633
634    REWARDER_BALANCE_POSITION.setStorageUint256(currentRewarderBalance);
```

Figure 7.1: The read and write of the `REWARDER_BALANCE_POSITION` storage variable (green); the potentially re-executable transfer (red); the call that may mitigate the vulnerability (yellow) (`ETH-Staking-B2C-SC/contracts/Accounting.sol#L609–L634`)

Everstake argues that the described scenario is impossible because `_closedValidatorStop`, a called function, modifies a storage variable that prevents `_update` from being executed more than once (figure 7.2). We cannot say with certainty whether Everstake is correct. Regardless, we recommend that the storage write be moved earlier to eliminate the possibility of any such attack.

```
 92    function _closedValidatorStop(uint256 balanceChange) internal returns
(uint256 chargedBalance) {
 93        uint256 closedValidatorsNum = _calculateValidatorClose(balanceChange);
 94        if (closedValidatorsNum == 0) {
 95            return 0;
 96        }
 97
 98        // Event emit how much validators closed
 99        emit ChangeExpectValidatorsToStop(-int256(closedValidatorsNum));
100        //Update expected close validators num
101
EXPECTED_CLOSE_VALIDATORS_POSITION.setStorageUint256(EXPECTED_CLOSE_VALIDATORS_POSIT
ION.getStorageUint256() - closedValidatorsNum);
102
103        chargedBalance = closedValidatorsNum * BEACON_AMOUNT;
104        // withdrawRequestQueue
105        _getQueue().filledAmount += chargedBalance;
106    }
```

Figure 7.2: The call to `_closedValidatorStop` in figure 7.1 mitigates the vulnerability if there is no way to undo the change on line 101.
(`ETH-Staking-B2C-SC/contracts/Withdrawer.sol#L92–L106`)

Exploit Scenario

Mallory finds a way to exercise the transfer on line 620 of figure 7.1 and then to restore `EXPECTED_CLOSE_VALIDATORS_POSITION` in figure 7.2 to its original value. In doing so, Mallory is able to drain the Everstake rewards treasury of its funds.

Recommendations

Short term, move the write of `REWARDER_BALANCE_POSITION` on line 634 of figure 7.1 to before the conditional on line 624 of figure 7.1. Doing so will ensure that the storage variable is updated even if the return on line 625 or 631 is executed.

Long term, develop and adopt a formal policy for working with storage variables. The policy might include rules such as the following:

- If a storage variable is updated, the updated value is based on a read performed within the same scope.
- Between a read and write of a storage variable, there are no intervening returns.

Developing and adopting such a policy will help to avoid problems like the one alleged above.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	Description
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system’s resistance to transaction-ordering attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.

Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Non-Security-Related Recommendations

The following recommendations are not associated with specific vulnerabilities. However, implementing them may enhance code readability and prevent the introduction of vulnerabilities in the future.

- **Use formatters to format both the Solidity and JavaScript files.** Currently, neither appear to be formatted. For example, there are several lines that contain nothing but whitespace. Furthermore, indentation is inconsistent, as demonstrated by figure C.1.

```
983   });
984
985     it("success: `withdraw` (with round by 2 wei)", async () => {
```

Figure C.1: An example of inconsistent indentation (`ETH-Staking-B2C-SC/test/pool_withdraw.js#L983–L985`)

- **Make the contracts in figures C.2 through C.5 abstract.** Currently, none of the contracts are instantiated.

```
8     contract AutocompoundAccounting {
```

Figure C.2: `AutocompoundAccounting` could be made abstract. (`ETH-Staking-B2C-SC/contracts/AutocompoundAccounting.sol#L8`)

```
7     contract Governor {
```

Figure C.3: `Governor` could be made abstract. (`ETH-Staking-B2C-SC/contracts/Governor.sol#L7`)

```
10    contract Withdrawer {
```

Figure C.4: `Withdrawer` could be made abstract. (`ETH-Staking-B2C-SC/contracts/Withdrawer.sol#L10`)

```
7     contract OwnableWithSuperAdmin is Initializable, ContextUpgradeable {
```

Figure C.5: `OwnableWithSuperAdmin` could be made abstract. (`ETH-Staking-B2C-SC/contracts/utils/OwnableWithSuperAdmin.sol#L7`)

- **Consider using `SLOT` rather than `POSITION` as a suffix for constants.** `SLOT` is a more widely used term. It is also more concise.

```
37    bytes32 internal constant POOL_FEE_POSITION =
keccak256("accounting.poolFee");
```

Figure C.6: An example constant with the `POSITION` suffix (`ETH-Staking-B2C-SC/contracts/Accounting.sol#L37`)

- **Use `Errors.ZeroValue` rather than `Errors.InvalidAmount` in figure C.7.**

```
if (amount == 0) revert Errors.InvalidAmount("0");
```

Figure C.7: Use of `Errors.InvalidAmount` to flag an invalid value of `0` (`ETH-Staking-B2C-SC/contracts/Pool.sol#L160`)

- **In `WithdrawRequestQueue`, change `Queue` to `Info`, `Stats`, or some other similar term.** "Queue" suggests a sequence of elements; however, the data structure does not store a sequence of elements.

```
/// @dev Global pool withdrawal queue struct
struct WithdrawRequestQueue {
    /// @dev Alltime withdraw requested amount
    uint256 requestedAmount;
    /// @dev Actual allowed to intechange amount
    uint256 allowedInterchangeAmount;
    /// @dev Alltime withdraw filled amount
    uint256 filledAmount;
    /// @dev Alltime withdraw claimed amount
    uint256 claimedAmount;
}
```

Figure C.8: The definition of `WithdrawRequestQueue` (`ETH-Staking-B2C-SC/contracts/Withdrawer.sol#L29–L39`)

- **In figures C.9 and C.10, change the arguments to `Errors.InvalidParam`.** The problem is not `owner` or `ownerOrSuper`, but rather the message sender.

```
if (!(owner() == _msgSender() || _superAdmin == _msgSender())) revert
Errors.InvalidParam("ownerOrSuper");
```

Figure C.9: A misleading error message (`ETH-Staking-B2C-SC/contracts/utils/OwnableWithSuperAdmin.sol#L47`)

```
if (owner() != _msgSender()) revert Errors.InvalidParam("owner");
```

Figure C.10: Another misleading error message (`ETH-Staking-B2C-SC/contracts/utils/OwnableWithSuperAdmin.sol#L55`)

- **Change `getStorageAddress` to `getStorageAsAddress`—that is, insert `As`; make the same change to all similar function names.** `getStorageAddress` sounds like it returns "the storage's address" as opposed to "the storage's contents as an address."

```
function getStorageAddress(bytes32 position) internal view returns (address
data) {
 17         assembly { data := sload(position) }
 18     }
```

Figure C.11: Misleading function name (`ETH-Staking-B2C-SC/contracts/lib/UnstructuredStorage.sol#L16–L18`)

- **Fix the spelling error highlighted in figure C.12.**

```
526     uint256 lenght = _roundPendingStakers()[activePendingRound].length();
```

Figure C.12: Spelling error (`ETH-Staking-B2C-SC/contracts/Accounting.solL#L513`)

D. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From January 15 to January 16, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the Everstake team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the seven issues described in this report, Everstake has resolved five issues, has partially resolved one issue, and has not resolved the remaining one issue. For additional information, please see the Detailed Fix Review Results below.

Separate from our review, Everstake noticed that `return` statements added to the `Pool` contract's `deposit` function could cause it to fail to emit a `StakeAdded` event. Everstake resolved this issue in PR #17 by moving the event's `emit` statement to before the `return` statements. Everstake additionally modified its existing tests in PR #37 to check for the event's emission. We reviewed the two PRs and verified that they resolved the issue.

ID	Title	Severity	Status
1	Project uses archived and outdated tools	Informational	Resolved
2	Project uses outdated dependencies	Informational	Resolved
3	Documentation is out of sync with the code	Informational	Resolved
4	Documentation lacks a glossary	Informational	Resolved
5	Duplicated code	Informational	Resolved
6	Storage variables updated by multiple contracts in the inheritance tree	Informational	Partially Resolved
7	_update may fail to update REWARDER_BALANCE_POSITION	Informational	Unresolved

Detailed Fix Review Results

TOB-EVERSTAKE-1: Project uses archived and outdated tools
Resolved in PR #16 and PR #36. The project now uses Hardhat, rather than Truffle, as its development framework. All configuration files and tests were updated accordingly.

TOB-EVERSTAKE-2: Project uses outdated dependencies
Resolved in PR #19. The `@openzeppelin/contracts` and `@openzeppelin/contracts-upgradeable` dependencies were upgraded to version 4.9.6, which is the latest with major version 4. We verified that the Everstake contracts compile and that the tests pass with the new version of the OpenZeppelin contracts.

TOB-EVERSTAKE-3: Documentation is out of sync with the code
Resolved in PR #24 and PR #34. The `pool.md` and `accounting.md` files were updated to reflect the functions in the `Pool` and `Accounting` contracts (respectively).

TOB-EVERSTAKE-4: Documentation lacks a glossary
Resolved in PR #24. A `glossary.md` file was added. The glossary defines each term mentioned in TOB-EVERSTAKE-4.

TOB-EVERSTAKE-5: Duplicated code
Resolved in PR #18. Everstake created an `Ownable` contract that contains the common elements of the `TreasuryBase` and `OwnableWithSuperAdmin` contracts. The `TreasuryBase` and `OwnableWithSuperAdmin` contracts now inherit from the new `Ownable` contract.[1]

TOB-EVERSTAKE-6: Storage variables updated by multiple contracts in the inheritance tree
Partially resolved in PR #35. A function `_poolRewardsAutocompound` was added to the `AutocompoundAccounting` contract. This function performs the writes to the `PENDING_RESTAKED_VALUE` and `TOTAL_BALANCE_POSITION` storage variables that were previously performed by the `Accounting` contract's `_depositAccount` function. That function now calls `_poolRewardsAutocompound` to perform the writes.

As a result of this change, the `Accounting` contract no longer writes to the `TOTAL_BALANCE_POSITION` storage variable. The contract does still write to the `PENDING_RESTAKED_VALUE_POSITION` storage variable, however.

Regarding the cases not fixed by PR #35, Everstake provided the following explanation:

[1] Note that Everstake chose not to eliminate the `OwnableWithSuperAdmin` contract and create a `SuperAdmin` contract, as suggested in TOB-EVERSTAKE-5. Nonetheless, the duplicated code has been eliminated.

To address this, we have implemented a fix for one of the cases. For the remaining instances, we have opted to retain the current implementation for the following reasons:

> *Avoiding a Global Refactor: A comprehensive restructuring of all contracts would require extensive modifications, introducing additional complexity and potential risks without delivering proportional benefits.*
>
> *Optimization of Storage Calls: The current approach is designed to minimize gas costs, which is critical for the protocol's efficiency. Changing the remaining cases would lead to increased gas usage, which we aim to avoid.*

We believe this approach strikes a balance between addressing risks and maintaining the protocol's efficiency. We will continue to monitor and reassess as needed to ensure the protocol's safety and reliability.

TOB-EVERSTAKE-7: _update may fail to update REWARDER_BALANCE_POSITION
Unresolved. Everstake provided the following explanation for this finding's fix status:

Thank you for bringing up your concerns regarding the late write of REWARDER_BALANCE_POSITION and the associated risks. We have carefully considered the points raised and would like to provide clarity on our reasoning for maintaining the current implementation:

> *Trusted Address: The rewardsTreasury is a trusted address within the protocol, and its operations are in line with the intended behavior.*
>
> *Gas Optimization: The current implementation is designed to optimize gas costs, a critical factor in maintaining protocol efficiency.*
>
> *Storage Update Preceding External Call: The _closedValidatorStop function ensures that storage is updated before any external transfer calls, mitigating potential reentrancy risks.*
>
> *Transfer Logic: The transfer operation reduces rewardsTreasury.balance, which ensures that subsequent _closedValidatorStop calls cannot proceed if the balance is insufficient.*

Considering these safeguards, we believe the current implementation appropriately balances security and efficiency. However, we remain vigilant and will reassess should any new developments or changes in trust assumptions arise.

E. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.

F. Non-Security-Related Findings Fix Review Results

At Everstake's request, we reviewed the team's implementations of the recommendations in appendix C. Everstake implemented all eight recommendations. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Use formatters to format both the Solidity and JavaScript files.	**Implemented**
2	Make the contracts in figures C.2 through C.5 abstract.	**Implemented**
3	Consider using SLOT rather than POSITION as a suffix for constants.	**Implemented**
4	Use Errors.ZeroValue rather than Errors.InvalidAmount in figure C.7.	**Implemented**
5	In WithdrawRequestQueue, change Queue to Info, Stats, or some other similar term.	**Implemented**
6	In figures C.9 and C.10, change the arguments to Errors.InvalidParam.	**Implemented**
7	Change getStorageAddress to getStorageAsAddress—that is, insert As; make the same change to all similar function names.	**Implemented**
8	Fix the spelling error highlighted in figure C.12.	**Implemented**

Detailed Fix Review Results

1: Use formatters to format both the Solidity and JavaScript files.
Implemented in PR #22. The JavaScript and Solidity files were formatted using VS Code's Prettier extension (`esbenp.prettier-vscode`), version 11.0.0. To help verify the fix, we formatted all of the project's files using the Format Files extension (`jbockle.jbockle-format-files`), version 3.4.0.

In our results, additional blank lines were added to the following files in the `contracts` directory:

- `Accounting.sol`
- `interfaces/IAccounting.sol`
- `interfaces/IPool.sol`
- `test/OwnableWithSuperAdminDeprecated.sol`
- `utils/Ownable.sol`

Also, two commas were added to the `_const.js` file in the `test` directory. Nonetheless, because the additional changes are minor and involve only blank lines and punctuation, we are considering the recommendation implemented.

2: Make the contracts in figures C.2 through C.5 abstract.
Implemented in PR #23. Each of the named contracts (`AutocompoundAccounting`, `Governor`, `Withdrawer`, and `OwnableWithSuperAdmin`) was made abstract.

3: Consider using SLOT rather than POSITION as a suffix for constants.
Implemented in PR #20. All `POSITION` suffixes were changed to `SLOT`.

4: Use Errors.ZeroValue rather than Errors.InvalidAmount in figure C.7.
Implemented in PR #23. `Errors.InvalidAmount` was changed to `Errors.ZeroValue`.

5: In WithdrawRequestQueue, change Queue to Info, Stats, or some other similar term.
Implemented in PR #23. Rather than change `Queue` to `Info`, `Info` was appended so that the resulting struct name is `WithdrawRequestQueueInfo`. Nonetheless, we are considering the recommendation implemented.

6: In figures C.9 and C.10, change the arguments to Errors.InvalidParam.
Implemented in PR #23. The arguments to `Errors.InvalidParam` were each changed to `“sender”`.

7: Change getStorageAddress to getStorageAsAddress—that is, insert As; make the same change to all similar function names.

Implemented in PR #21 and PR #38. For each function beginning with `getStorage` or `setStorage`, `As` was inserted just after `Storage`.

8: Fix the spelling error highlighted in figure C.12.
Implemented in PR #23. The spelling error was fixed.

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.