Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

GENESIS

Liquid Restaking

Veridise Inc.

January 10, 2024

► **Prepared For:**

GenesisLRT
https://www.genesislrt.com/

► **Prepared By:**

Kostas Ferles
Andreea Buțerchi

► **Contact Us:** contact@veridise.com

► **Version History:**

Jan. 10, 2024	V1
Dec. 27, 2023	Initial Draft

Contents

1 Executive Summary

From Dec. 20, 2023 to Dec. 22, 2023, GenesisLRT engaged Veridise to review the security of their Liquid Restaking project. The review covered the project's main contracts that implement the liquid restaking logic. Veridise conducted the assessment over 6 person-days, with 2 engineers reviewing code over 3 days from commits `dd17853 - 29d5f51`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The Liquid Restaking developers provided the source code of the Liquid Restaking contracts for review. To facilitate the Veridise auditors' understanding of the code, the Liquid Restaking developers provided some short documentation in the form of a README file and also met with our auditors to give a brief walk-through of the codebase.

The source code contained a test suite, which the Veridise auditors used to understand the expected usage of the protocol and also understand how the protocol is expected to be deployed.

During the audit, the GenesisLRT developers made several functional changes to the code. This is because they fixed some orthogonal issues while they were fixing the issues discovered by Veridise auditors and they also implemented a new minor feature. Due to this, Veridise auditors reviewed the additional functionality in these commits as requested by GenesisLRT. The Veridise auditors only focused their review only on the parts of the contracts mentioned by the client during the additional review.

Summary of issues detected. The audit uncovered 14 issues, 0 of which are assessed to be of high or critical severity by the Veridise auditors. The highest severity issue discovered by the Veridise auditors is a medium reentrancy issue that can complicate the way of pending unstake requests are distributed. The Veridise auditors also identified several low-severity issues, including a contract field that was never set as well as a number of minor issues. The Liquid Restaking developers fixed all but two issues because of reasons that are explained later in this report.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Liquid Restaking	`dd17853 - 29d5f51`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Dec. 20 - Dec. 22, 2023	Manual & Tools	2	6 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	0	0
High-Severity Issues	0	0
Medium-Severity Issues	1	1
Low-Severity Issues	2	1
Warning-Severity Issues	7	6
Informational-Severity Issues	4	4
TOTAL	14	12

Table 2.4: Category Breakdown.

Name	Number
Maintainability	7
Logic Error	3
Usability Issue	2
Reentrancy	1
Gas Optimization	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Liquid Restaking's smart contracts. In our audit, we sought to answer the following questions:

- Do the contracts implement the restaking logic correctly?
- Is the project's ERC20 secure and properly implemented?
- Are user funds secure?
- Are there any usability issues?
- Are there any known vulnerabilities (e.g., reentrancies)?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard. These type of tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.
- *Fuzzing/Property-based Testing.* We also leverage fuzz testing to determine if the protocol may deviate from the expected behavior. To do this, we formalize the desired behavior of parts of the protocol as [V] specifications and then use our fuzzing framework OrCa to determine if a violation of the specification can be found.

Scope. The scope of this audit is limited to the `projects/liquid-restaking/contracts` (excluding the `libraries` sub-folder) folder of the source code provided by the Liquid Restaking developers, which contains the smart contract implementation of the Liquid Restaking.

Methodology. Veridise auditors reviewed the reports of previous audits for Liquid Restaking, inspected the provided tests, and read the Liquid Restaking documentation. They then began a manual audit of the code assisted by both static analyzers and automated testing.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

Table 3.4: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-GENS-VUL-001	Reentrancy issue in distributeUnstakes	Medium	Fixed
V-GENS-VUL-002	Index of historicalRatios is skipped	Low	Intended Behavior
V-GENS-VUL-003	Field _minStakeAmount is never set	Low	Fixed
V-GENS-VUL-004	No setter for EigenPodManager	Warning	Fixed
V-GENS-VUL-005	Duplicate code snippet	Warning	Fixed
V-GENS-VUL-006	_checkRatioRules relies on default init	Warning	Fixed
V-GENS-VUL-007	Hardcoded constant in RatioFeed	Warning	Fixed
V-GENS-VUL-008	Missing modifiers	Warning	Fixed
V-GENS-VUL-009	Max constants do not represent maximum values	Warning	Fixed
V-GENS-VUL-010	Interaction before state update	Warning	Acknowledged
V-GENS-VUL-011	Unused Imports	Info	Fixed
V-GENS-VUL-012	Typos	Info	Fixed
V-GENS-VUL-013	Missed opportunity for calling _getRestakerOrRe.	Info	Fixed
V-GENS-VUL-014	Field _minUnstakeAmount is unused	Info	Fixed

3.4 Detailed Description of Issues

3.4.1 V-GENS-VUL-001: Reentrancy issue in distributeUnstakes

Severity	Medium	**Commit**	dd17853
Type	Reentrancy	**Status**	Fixed
File(s)	`RestakingPool.sol`		
Location(s)	Function distributeUnstakes		
Confirmed Fix At	2163855		

There is a possible DoS attack due to reentrancy that would target the check `i < _pendingUnstakes.length` in the following loop. More details below:

- There is a reentry point in function `distributeUnstakes`, via the call to `_sendValue`, on line 248.
- Even though `distributeUnstakes` is nonReentrant, the same doesn't hold for `unstake` whose state modifying set intersects that of `distributeUnstakes`.
- So a malicious actor could stake via a smart contract with a fallback function that calls `unstake`, which, if executed successfully, will increase the length of array `_pendingUnstakes`.
- The problem lies on line 258 that sets the j-th element of array `unstakes` whose length is determined by the length of `_pendingUnstakes` at the beginning of the transaction (hence its value could become stale). So, if the attack contract successfully unstakes through the fallback function, line 258 could lead to an out of bounds exception because the array's length will be larger.
- For this attack to become successful, the attacker will need to append sufficient amount of entries in `_pendingUnstakes` to ensure an out of bound index on line 258. So, the attacker will be bounded by the amount of shares they own.

```
1        Unstake[] memory unstakes = new Unstake[](
2        _pendingUnstakes.length - _pendingGap   // Setting array length
3    );
4    ...
5    while (
6        i < _pendingUnstakes.length &&          // Loop check
7        poolBalance > 0 &&
8        gasleft() > _distributeGasLimit
9    ) {
10       ...
11       bool success = _sendValue(              // Reentry point
12           unstake_.recipient,
13           unstake_.amount,
14           true
15       );
16       ...
17       unstakes[j] = unstake_;                 // Possible out-of-bound index
18       ++j;
19   }
```

Figure 3.1: Snippet from `distributeUnstakes`

Impact This might complicate the process of distributing the pending unstake requests. However, even in the presence of such malicious contract, it should be eventually feasible to process all pending requests because malicious actors would be bounded by the amount of shares they own.

Recommendation There are two possible solutions to this issue:

1. Mark function `unstake` as `nonReentrant`
2. Save `_pendingUnstakes.length` to a temporary variable at the beginning of the function and use that variable to perform the check in the loop condition.

Developer Response The issue was fixed by commit `2163855`.

3.4.2 V-GENS-VUL-002: Index of historicalRatios is skipped

Severity	Low	**Commit**	dd17853
Type	Logic Error	**Status**	Intended Behavior
File(s)	`RatioFeed.sol`		
Location(s)	Function updateRatio		
Confirmed Fix At	N/A		

Function `updateRatio` seems to be skipping an index when appending the new ratio to the `hisRatio.historicalRatios` array. In the first snippet below , the updated index is `((latestOffset + 1) % 8) + 1`, which effectively adds two to `latestOffset`. Even though the second addition is intended to skip the first position of the array that stores `latestOffset` itself, the first addition seems unnecessary.

```
hisRatio.historicalRatios[((latestOffset + 1) % 8) + 1] = uint64(
        newRation
);
```

Figure 3.2: Snippet from `updateRatio`

For example, consider the following calculation from `averagePercentageRate` that retrieves the `oldestRatio`. Let's assume we only have one day of historical ratios (i.e., latestOffset is 1), then oldestRatio will equal to `hisRatio.historicalRatios[1]`. But, as mentioned above, this slot is skipped by `updateRation`.

```
uint256 oldestRatio = hisRatio.historicalRatios[
            ((latestOffset - day) % 8) + 1
        ];
```

Figure 3.3: Snippet from `averagePercentageRate`

Impact The results returned by `averagePercentageRate` will be inaccurate.

Recommendation If the above is not the intended behavior (which is not clear from the documentation), `updateRatio` should update `(latestOffset % 8) + 1` instead of the current index.

Developer Response The developers chose to ignore this recommendation as it was their intention.

After discussing with the developers, it is still unclear what is the intended behavior of function `averagePercentageRate`. However, this is a read-only function and the exposed risk is minimal to negligible (since all contracts are upgradeable).

3.4.3 V-GENS-VUL-003: Field _minStakeAmount is never set

Severity	Low	**Commit**	dd17853
Type	Logic Error	**Status**	Fixed
File(s)	`RestakingPool.sol`		
Location(s)	RestakingPool		
Confirmed Fix At	2163855		

Field `_minStakeAmount`, despite being used in several locations, it is never being set.

Impact Effectively, this means that the minimum stake amount will always be determined by the current `ETH/genETH` ratio.

Recommendation Provide a setter function for `_minStakeAmount`.

Developer Response The issue was fixed by commit `2163855`.

3.4.4 V-GENS-VUL-004: No setter for EigenPodManager

Severity	Warning	**Commit**	dd17853
Type	Usability Issue	**Status**	Fixed
File(s)	`ProtocolConfig.sol`		
Location(s)	contract ProtocolConfig		
Confirmed Fix At	2163855		

Contract `ProtocolConfig` does not provide a way for setting the `EigenPodManager` field.

Impact This might introduce usability issues if retrieving the manager from this context becomes necessary.

Recommendation We recommend adding a setter for `EigenPodManager`.

Developer Response Field `EigenPodManager` was removed from the contract by commit `2163855`.

3.4.5 V-GENS-VUL-005: Duplicate code snippet

Severity	Warning	**Commit**	dd17853
Type	Maintainability	**Status**	Fixed
File(s)	`restaker/RestakerDeployer.sol`		
Location(s)	Functions deployRestaker and getRestaker		
Confirmed Fix At	2163855		

Expression `abi.encodePacked(BEACON_PROXY_BYTECODE, abi.encode(beacon, ""))` is used in two places within `restaker/RestakerDeployer.sol`.

Impact In future iterations of the protocol, there is the risk of one location being updated but not the other.

Recommendation We recommend creating a global constant in the contract.

Developer Response The issue was fixed by commit `2163855`.

3.4.6 V-GENS-VUL-006: _checkRatioRules relies on default init

Severity	Warning	**Commit**	dd17853
Type	Maintainability	**Status**	Fixed
File(s)	`RatioFeed.sol`		
Location(s)	Function _checkRatioRules		
Confirmed Fix At	2163855		

Function `_checkRatioRules` relies on the default initialization of return variables `valid` and `reason` (see snippet below).

```
function _checkRatioRules(
  uint256 lastUpdated,
  uint256 newRatio,
  uint256 oldRatio
) internal view returns (bool valid, string memory reason) {
    if (oldRatio == 0) {
        return (valid = true, reason); // reason is default init.
    }

    if (block.timestamp - lastUpdated < 12 hours) {
        return (valid, reason = "ratio was updated less than 12 hours ago"); // valid
     is default init
    }

    ...
```

Figure 3.4: Snippet from `_checkRatioRules`

Impact It is complicated to reason about correctness with the current version of the function.

Recommendation We recommend removing both return variables and simply return a tuple of constants in each branch. For instance, the first return statement can be simplified to `return (true, "");`

Developer Response The developers now return an enum from the function that does not exhibit this behavior.

3.4.7 V-GENS-VUL-007: Hardcoded constant in RatioFeed

Severity	Warning	**Commit**	dd17853
Type	Maintainability	**Status**	Fixed
File(s)	`RatioFeed.sol`		
Location(s)	Function repairRatio		
Confirmed Fix At	2163855		

Function `repairRatio` uses a hardcoded constant to check whether the new ration exceeds a threshold.

```
1 if (newRatio > 1e18 || newRatio == 0) {
2     revert RatioNotUpdated("not in range");
3 }
```

Figure 3.5: Snippet from `repairRatio`

Impact As the protocol involves such hardcoded constants might introduce maintainability issues.

Recommendation Consider introducing a `MAX_RATIO` constant.

Developer Response The issue was fixed by commit `2163855`.

3.4.8 V-GENS-VUL-008: Missing modifiers

Severity	Warning	**Commit**	dd17853
Type	Logic Error	**Status**	Fixed
File(s)	`RestakingPool.sol`		
Location(s)	withdrawNonBeaconChainETHBalanceWei, recoverTokens		
Confirmed Fix At	2163855		

Given the context, functions `withdrawNonBeaconChainETHBalanceWei` and `recoverTokens` are intended to be executed only by the contract's operator. However, these two functions are not marked with the `onlyOperator` modifier.

Impact Anyone would be able to withdraw non-beacon ETH or recover tokens. Even though everything will go to the pod's owner, it would be advisable to limit access to this function.

Recommendation Mark both functions with `onlyOperator`.

Developer Response The issue was fixed by commit `2163855`.

3.4.9 V-GENS-VUL-009: Max constants do not represent maximum values

Severity	Warning	**Commit**	dd17853
Type	Usability Issue	**Status**	Fixed
File(s)	`RatioFeed.sol RestakingPool.sol`		
Location(s)	_setDistributeGasLimit, _setRatioThreshold		
Confirmed Fix At	2163855		

Due to the implementation of functions `_setDistributeGasLimit` and `_setRatioThreshold` (see snippets below), constants `MAX_THRESHOLD` (in `RatioFeed.sol`) and `MAX_GAS_LIMIT` (in `RestakingPool.sol`) do not actually reflect the maximum allowed value. In both cases, the maximum allowed is one less than the value of the constant.

```
function _setDistributeGasLimit(uint32 newValue) internal {
    if (newValue >= MAX_GAS_LIMIT || newValue == 0) {
            revert RatioNotUpdated("not in range");
```

Figure 3.6: Snippet from `_setDistributeGasLimit`

```
function _setRatioThreshold(uint256 value) internal {
    if (value >= MAX_THRESHOLD || value == 0) {
            revert RatioThresholdNotInRange();
```

Figure 3.7: Snippet from `_setRatioThreshold`

Impact This can be misleading to users of the contracts.

Recommendation Replace operators >= with > in both of the snippets above.

Developer Response The issue was fixed by commit `2163855`.

3.4.10 V-GENS-VUL-010: Interaction before state update

Severity	Warning	**Commit**	dd17853
Type	Maintainability	**Status**	Acknowledged
File(s)	`RestakingPool.sol`		
Location(s)	Function unstake		
Confirmed Fix At	N/A		

Function `unstake` calls `token.burn` before performing several state updates. When possible, it is recommended to perform any external interactions as the last part of the function.

```
        token.burn(from, shares);

    _addIntoQueue(to, amount);

    _totalUnstaked += amount;
    emit Unstaked(from, to, amount, shares);
```

Impact Currently, the ERC20 token used by the project does not contain any re-entry points. However, if that changes in the future, this will increase the security risks.

Recommendation Move the call to `burn` at the end of the function.

Developer Response Due to future needs of the protocol, the developers chose to ignore this recommendation.

The Veridise team agrees that is safe to ignore this recommendation since the protocol has complete control of the used token.

3.4.11 V-GENS-VUL-011: Unused Imports

Severity	Info	**Commit**	dd17853
Type	Maintainability	**Status**	Fixed
File(s)	Multiple Files		
Location(s)	Multiple Locations		
Confirmed Fix At	2163855		

The following files import an unnecessary file.

- `restaker/Restaker.sol`: unused import `ReentrancyGuardUpgradeable.sol`
- `cToken.sol`: unused import `PausableUpgradeable.sol`
- `RestakingPool.sol`: unused import `IRestakerDeployer.sol`
- `restaker/RestakerFacet.sol`: unused import `ReentrancyGuardUpgradeable.sol`

Recommendation We recommend removing the unused imports.

Developer Response The issue was fixed by commit `2163855`.

3.4.12 V-GENS-VUL-012: Typos

Severity	Info	**Commit**	dd17853
Type	Maintainability	**Status**	Fixed
File(s)	Multiple Files		
Location(s)	Several locations		
Confirmed Fix At	2163855		

Consider fixing the following typos:

- File `ProtocolConfig.sol` line 19: `OnlyGovernancAllowed -> OnlyGovernanceAllowed`
- File `ProtocolConfig.sol` line 145: `setRestakedDeployer -> setRestakerDeployer`

Developer Response The issue was fixed by commit `2163855`.

3.4.13 V-GENS-VUL-013: Missed opportunity for calling _getRestakerOrRevert

Severity	Info	**Commit**	dd17853
Type	Maintainability	**Status**	Fixed
File(s)	`RestakingPool.sol`		
Location(s)	batchDeposit		
Confirmed Fix At	2163855		

Function `batchDeposit` can replace the following snippet with a call to function `_getRestakerOrRevert` .

```
1     address restaker = _restakers[_getProviderHash(provider)];
2     if (restaker == address(0)) {
3         revert PoolRestakerNotExists();
4     }
```

Figure 3.8: Snippet from `batchDeposit`

Impact This can lead to maintainability issues down the road.

Recommendation We recommend using existing functions, like `_getRestakerOrRevert`, whenever possible.

Developer Response The issue was fixed by commit `2163855`.

3.4.14 V-GENS-VUL-014: Field _minUnstakeAmount is unused

Severity	Info	**Commit**	dd17853
Type	Gas Optimization	**Status**	Fixed
File(s)	`RestakingPool.sol`		
Location(s)	RestakingPool		
Confirmed Fix At	2163855		

Field `_minUnstakeAmount` in the `RestakingPool` contract is never used.

Impact This can lead to extra gas costs and maintainability issues.

Recommendation Consider removing the field if it is not needed.

Developer Response The developers added a setter function for this field.

4 Fuzz Testing

4.1 Methodology

Our goal was to fuzz test Liquid Restaking to assess its functional correctness (i.e, whether the implementation deviates from the intended behavior). We used Hardhat to setup the environment and write the deployment script. The Hardhat test suite provided by the Liquid Restaking developers helped us in setting up the fuzzing process. Based on the deployed artifacts, we performed fuzzing campaigns using OrCa in order to find violations for the specifications detailed below.

4.2 Properties Fuzzed

Table 4.1 describes the invariants we fuzz-tested. The first column states which component the invariant is associated with. The second describes the invariant informally in English, and the third shows the total amount of compute time spent fuzzing this property. The last column notes whether we found a bug when fuzzing the invariant (✗ indicates no bug was found and ✓ means fuzzing this invariant revealed a bug).

The Veridise auditors devoted a total of 20 compute-hours to fuzzing this protocol, identifying a total of 0 bugs.

Table 4.1: Invariants Fuzzed.

Specification	Invariant	Minutes Fuzzed	Bugs Found
V-GENS-SPEC-001	ERC20.01: transfer should revert if a user atte. . .	300	0
V-GENS-SPEC-002	ERC20.02: Funds should be successfully transfer	300	0
V-GENS-SPEC-003	ERC20.03: approve makes appropriate state cha	300	0
V-GENS-SPEC-004	ERC20: static totalSupply	300	0
V-GENS-SPEC-005	ERC20: allowance/balances	300	0
V-GENS-SPEC-006	ERC20: no extra modfications	300	0
V-GENS-SPEC-007	ERC20: burnFrom burns correct amount	300	0
V-GENS-SPEC-008	ERC20: burnFrom reduces total supply	300	0
V-GENS-SPEC-009	ERC20: mint increases totalSupply/balance	300	0
V-GENS-SPEC-010	StakingPool: Reentrancy issue in distributeUnst.	300	0
V-GENS-SPEC-011	cToken: pause/ unpause access modifiers	300	0

4.3 Detailed Description of Fuzzed Specifications

4.3.1 V-GENS-SPEC-001: ERC20.01: transfer should revert if a user attempts to send more funds than they have

Minutes Fuzzed	300	Bugs Found	0

Specification

Scope cToken.sol

Natural Language `transfer` should revert if a user attempts to send more funds than they have.

```
vars: cToken ct
inv: reverted(ct.transfer(to, amt), amt > ct.balanceOf(sender))
```

4.3.2 V-GENS-SPEC-002: ERC20.02: Funds should be successfully transferred from sender to to as long as sender is not to

Minutes Fuzzed	300	Bugs Found	0

Specification

Scope cToken.sol

Natural Language Funds should be successfully transferred from `sender` to `to` as long as `sender` $\neq$ `to`.

```
vars: cToken ct
inv: finished(ct.transfer(to, amt),
  to != sender |=>
    ct.balanceOf(sender) = old(ct.balanceOf(sender)) - amt &&
    ct.balanceOf(to) = old(ct.balanceOf(to)) + amt
)
```

4.3.3 V-GENS-SPEC-003: ERC20.03: approve makes appropriate state changes

Minutes Fuzzed	300	Bugs Found	0

Specification

Scope cToken.sol

Natural Language `approve` makes appropriate state changes.

`approve` should never finish in a state where the allowance of the `spender` is not equal to the given `amount`. `totalSupply`, other allowances and balances should not be modified.

```
vars: cToken ct, address o1, address o2, address o3
inv: finished(ct.approve(spender, amt),
  o2 != sender && o1 != spender |=>
    ct.allowance(sender, spender) = amt &&
    ct.allowance(sender, o1) = old(ct.allowance(sender, o1)) &&
    ct.allowance(o2, o3) = old(ct.allowance(o2, o3)) &&
    ct.balanceOf(o3) = old(ct.balanceOf(o3)) &&
    ct.totalSupply() = old(ct.totalSupply())
)
```

4.3.4 V-GENS-SPEC-004: ERC20.03: transfer should not modify totalSupply, allowances, or balances other than sender and to

Minutes Fuzzed	300	Bugs Found	0

Specification

Scope cToken.sol

Natural Language `transfer` should not modify `totalSupply`, allowances, or balances other than `sender` and `to`.

```
vars: cToken ct, address o1, address o2, address o3
inv: finished(ct.transfer(to, amt),
  o1 != sender && o1 != to |=>
    ct.totalSupply() = old(ct.totalSupply()) &&
    ct.balanceOf(o1) = old(ct.balanceOf(o1)) &&
    ct.allowance(o2, o3) = old(ct.allowance(o2, o3))
)
```

4.3.5 V-GENS-SPEC-005: ERC20.04: transferFrom should enforce allowance and user balance

Minutes Fuzzed	300	Bugs Found	0

Specification

Scope cToken.sol

Natural Language `transferFrom` should enforce allowance and user balance.

`transferFrom` should revert when the amount requested is greater than what the spender owns or beyond the recipient's allowance.

```
vars: cToken ct
inv: reverted(ct.transferFrom(from, to, amt),
  amt > ct.balanceOf(from) || (from != sender && amt > ct.allowance(from, sender))
)
```

4.3.6 V-GENS-SPEC-006: ERC20.06: transferFrom should not modify totalSupply, other allowances, or balances

Minutes Fuzzed	300	Bugs Found	0

Specification

Scope cToken.sol

Natural Language `transferFrom` should not modify `totalSupply`, other allowances, or balances.

```
vars: cToken ct, address o1, address o2, address o3, address o4
inv: finished(ct.transferFrom(from, to, amt),
  o1 != from && o1 != to && o2 != sender && o3 != from |=>
    ct.balanceOf(o1) = old(ct.balanceOf(o1)) &&
    ct.allowance(from, o2) = old(ct.allowance(from, o2)) &&
    ct.allowance(o3, o4) = old(ct.allowance(o3, o4)) &&
    ct.totalSupply() = old(ct.totalSupply())
)
```

4.3.7 V-GENS-SPEC-007: ERC20.13: burn will revert if a user attempts to burn more than they own or more than their allowance

Minutes Fuzzed	300	Bugs Found	0

Specification

Scope cToken.sol

Natural Language `burn` will revert if a user attempts to burn more than they own or more than their allowance.

```
vars: cToken ct
inv: reverted(ct.burn(from, amt),
  amt > ct.balanceOf(from) || (from != sender && amt > ct.allowance(from, sender))
)
```

4.3.8 V-GENS-SPEC-008: ERC20.14: burn will reduce the total supply and the balance of from by the indicated amount

Minutes Fuzzed	300	Bugs Found	0

Specification

Scope cToken.sol

Natural Language `burn` will reduce the total supply and the balance of from by the indicated amount.

```
vars: cToken ct
inv: finished(ct.burn(from, amt),
  ct.balanceOf(from) = old(ct.balanceOf(from)) - amt &&
  ct.totalSupply() = old(ct.totalSupply()) - amt &&
  ( (from != sender || old(ct.allowance(from, sender)) != MAX_UINT256) ==>
    ct.allowance(from, sender) = old(ct.allowance(from, sender)) - amt )
)
```

4.3.9 V-GENS-SPEC-009: ERC20.16: mint will increase totalSupply and a user's balance by the indicated amount

Minutes Fuzzed	300	Bugs Found	0

Specification

Scope cToken.sol

Natural Language `mint` will increase `totalSupply` and a user's balance by the indicated amount.

```
vars: cToken ct
inv: finished(ct.mint(acc, amt),
  ct.balanceOf(acc) = old(ct.balanceOf(acc)) + amt &&
  ct.totalSupply() = old(ct.totalSupply()) + amt
)
```

4.3.10 V-GENS-SPEC-010: StakingPool: Reentrancy issue in distributeUnstakes

Minutes Fuzzed	300	Bugs Found	0

Specification

Scope RestakingPool.sol

Natural Language A complete description of this potential reentrancy issue can be found here.

Note: we added to `RestakingPool.sol` a new method called `getLengthPendingUnstakes` to be able to express the spec below.

Given that the attacker can exploit the reentrancy vulnerability by altering the length of `_pendingUnstakes`, the following spec says that it is never the case that the length of `_pendingUnstakes` grows after a call to `distributeUnstake`.

```
vars: RestakingPool rp
spec: []!finished(rp.distributeUnstakes, old(rp.getLengthPendingUnstakes()) <= rp.
    getLengthPendingUnstakes())
```

4.3.11 V-GENS-SPEC-011: cToken: pause/ unpause can only be called by governance

Minutes Fuzzed	300	Bugs Found	0

Specification

Scope cToken.sol

Natural Language `pause/ unpause` will revert if a user other than governance attempts to call them.

```
vars: cToken ct, ProtocolConfig pc
spec: []!finished(ct.pause, sender != pc.getGovernance())
```

```
vars: cToken ct, ProtocolConfig pc
spec: []!finished(ct.unpause, sender != pc.getGovernance())
```