Quantstamp

Auxo (Diff)

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Governance - ERC20
Timeline	2023-03-22 through 2023-04-03
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• Alexintosh/auxo-governance #a1f69a9
Auditors	• Ruben Koch Auditing Engineer • Ed Zulkoski Senior Auditing Engineer • Cameron Biniamow Auditing Engineer • Mostafa Yassin Auditing Engineer

Documentation quality	Medium
Test quality	High
Total Findings	5 Acknowledged: 5
High severity findings	0
Medium severity findings	0
Low severity findings	1 Acknowledged: 1
Undetermined severity findings	0
Informational findings	4 Acknowledged: 4

Summary of Findings

This diff-audit is based on Quantstamp's previous AUXO Governance audit. In scope are only the modified `PRV.sol` and the newly added `PRVMerkleVerifier.sol` files. No logical changes were performed in the contracts that were scope in the last audit, except some name changes. Since the initial audit, two of the three tokens have been rebranded. The `veAUXO` governance token is now called `ARV` and `xAUXO`, the liquid staking derivative of staked `AUXO`, is now called `PRV`.

In the codebase's state of the previous audit, `AUXO` tokens could never be unstaked, so the conversion to the `PRV` liquid staking derivative was only possible in one way and not back. We pointed out our concerns regarding the economical incentives for such a design in AUX-12 of the initial audit.

In the updated code of the `PRV` contract, such a conversion back to `AUXO` is now possible if the withdrawing user can provide a valid claim to burn some of their `PRV` and withdraw the equivalent amount of `AUXO`. The verification of claims via merkle trees is handled by the newly added `PRVMerkleVerifier` contract. The amount of `AUXO` that can be withdrawn depends on an off-chain calculation that assures that the cumulative price of the locked `AUXO` remains close to the net asset value of the treasury of the protocol. If `AUXO` were to trade at a significant premium so that the summed locked assets exceed the treasury's funds, a new window in the `PRVMerkleVerifier` contract would be instantiated that enables users to withdraw a portion of the total locked `AUXO`, if they can provide a appropriate claim signed by the DAO . Each user's withdraw eligibility depends on the amount associated with each claim.

No new major issues were identified and the code and test quality of the new files continue to be high.

ID	DESCRIPTION	SEVERITY	STATUS
AUX2-1	Privileged Roles and Ownership	Low	Acknowledged
AUX2-2	Missing Input Validation	Informational	Acknowledged
AUX2-3	No Guarantee for Successful Withdraw with Valid Claim	Informational	Acknowledged
AUX2-4	Multiple Claims for One Account per Window Have to Contain	Informational	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
	Cumulative Amounts		
AUX2-5	Previous Window Can Still Be Active when Overwritten	Informational	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report. Specifically, the off-chain calculation of the amount of `AUXO` that can be withdrawn for a window, depending on the `AUXO` price and the treasury net asset value, remained out-of-scope of this audit.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Findings

AUX2-1 Privileged Roles and Ownership

Low | Acknowledged

Update

The client acknowledged the issue with the following comment: *"In line with Quantstamp recommendations, we acknowledge that the following contracts have privileged roles and ownership, and users should therefore be aware that they are not fully trustless: `PRVMerkleVerifier.sol`"*

File(s) affected: `PRVMerkleVerifier.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. In the `PRVMerkleVerifier` contract, the owner can pause or set arbitrarily low withdrawal limits, which could prevent withdrawals from `PRV`.

Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

AUX2-2 Missing Input Validation

Informational | Acknowledged

Update

The client acknowledged the issue with the following comment: *"We acknowledge that `_endBlock`, `_maxAmount` and `_merkleRoot` are not validated when setting windows on the `MerkleVerifier`. As setting windows is an administrative function, we are happy to leave the inputs without contract-level validation - presuming that responsibility for checking the validity of these variables lies with the admin and any other operational checks and balances."*

File(s) affected: `PRVMerkleVerifier.sol`

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error.

- `PRVMerkleVerifier.setWindow()` should check that `_endblock > block.number`.
- `PRVMerkleVerifier.setWindow()` should check that `_merkleRoot` is not the `0x0` address and that `_maxAmount` is greater than zero.

Recommendation: We recommend adding the relevant checks.

AUX2-3 No Guarantee for Successful Withdraw with Valid Claim

Informational | Acknowledged

Update

The client acknowledged the issue with the following comment: *"Quantstamp notes that, with the current design, claims may exceed the `maxAmount` of a window. Quantstamp also notes that this is a constraint of the merkle root design, and cannot be enforced in code with the current design. As with AUX2-2, we assume that contract owners will ensure that claims can be processed within the `maxAmount` of a window."*

File(s) affected: `PRVMerkleVerifier.sol`

Description: Users can withdraw their `PRV` based on issued claims. In theory, the summed amounts of issued claims could exceed the `maxAmount` of a window, leaving users who attempt to withdraw after `maxAmount` `PRV` have already been withdrawn empty handed.

Recommendation: As the claims are not stored on-chain, there is no way to assure this on a smart contract level with the current design. Provide explaining documentation to the end user.

AUX2-4 Multiple Claims for One Account per Window Have to Contain Cumulative Amounts

Informational | Acknowledged

Update

The client acknowledged the issue with the following comment: *"Quantstamp notes that if a user is issued multiple claims in the same window then there is a potential problem that they will not be able to fully withdraw. This is because `amountWithdrawnFromWindow` is a nested mapping of `windowIndex` and `adddress`, so in the event that an account has multiple claims, they will only be able to withdraw a total equal to the highest single value across all claims. We acknowledge this is by design: accounts are only intended to have one claim per window, but can make partial claims if they wish. We also acknowledge that this limitation is not enforced anywhere in the contract and is up to the generator of the merkle tree to avoid adding duplicate accounts in claims for the window."*

File(s) affected: `PRVMerkleVerifier.sol`

Description: Currently, the `availableToWithdrawInClaim()` returns the difference between the `amount` field in the claim from the total amount withdrawn by a user in that window. In case multiple claims would get issued for the same user in a window, the additional claim's `amount` field would need to be summed with `amount` of the original claim to make the additional claim fully redeemable, as else (e.g. in case the second claim's `amount` is smaller than the first's) the check in `availableToWithdrawInClaim()` would fail.

Recommendation: Consider if this is a desired design. If not, add a `mapping(uint256 => mapping(bytes32 => uint256)) amountWithdrawnFromWindowPerClaim` mapping that keeps track of how much of a given claim has been withdrawn and adjust the `withdrawn()` function accordingly.

AUX2-5
Previous Window Can Still Be Active when Overwritten

Informational | Acknowledged

> **Update**
> The client acknowledged the issue with the following comment: *"We acknowledge that it is possible to overwrite an active window when calling the `setWindow` function. We also acknowledge that this would affect any users and claims in the overwritten window. As with AUX2-2, we rely on contract owners to understand the implications of the `setWindow` function and choose inputs accordingly."*

File(s) affected: `PRVMerkleVerifier.sol`

Description: Currently, active windows (meaning a window with `startBlock <= block.number` and `endBlock > block.number`) can be deleted by the owner by setting a new window via `PRVMerkleVerifier.setWindow()`. With that, unredeemed and valid claims of users would get invalidated.

Recommendation: Consider if this is by design. If not, require that `windows[nextWindowIndex - 1].endBlock <= block.number` in the `setWindow()` function.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Code Documentation

1. Fixed In `PRV._chargeFee()`, the comment "calculates the entry fee" is no longer correct, as the fee only applies to withdrawals.
2. Fixed The documentation for `PRVMerkleVerifier.Window` should note that `startBlock` is inclusive and `endBlock` is exclusive.

Adherence to Best Practices

1. Fixed In `PRVMerkleVerifier.sol`, the events `BudgetUpdated()` and `LockSet()`, along with the state variable `lockBlock`, are not used.
2. Unresolved The `inBudget()` modifier and the `budgetRemaining()` function share the same functionality. Consider requiring `budgetRemaining()` to be greater than 0 within the `inBudget()` modifier to unify the logic.
3. Fixed The `ReentrancyGuardUpgradeable` import is unused in the `PRVMerkleVerifier` contract. Consider removing it.

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

- `0ae...e6a ./src/interfaces/IPRV.sol`
- `06a...b92 ./src/interfaces/IWithdrawalManager.sol`

- `422...080 ./src/modules/PRV/PRV.sol`
- `a52...d40 ./src/modules/PRV/PRVMerkleVerifier.sol`

Tests

- `256...9b5 ./test/UpgradeDeployer.sol`
- `163...ea5 ./test/Upgrades.t.sol`
- `a4f...a31 ./test/utils.sol`
- `ff4...3f7 ./test/ARV.sol/MintBurn.t.sol`
- `28c...323 ./test/ARV.sol/Permit.t.sol`
- `113...25f ./test/ARV.sol/Setup.sol`
- `5f1...06b ./test/ARV.sol/NonTransferability.t.sol`
- `440...2df ./test/Upgradoor.sol/MigrateToPRV.t.sol`
- `c90...aba ./test/Upgradoor.sol/MigrateToARV.t.sol`
- `67a...09a ./test/Upgradoor.sol/Getters.t.sol`
- `7e0...3be ./test/Upgradoor.sol/Setup.sol`
- `548...7fe ./test/Upgradoor.sol/sharesTimelock/Migrate.t.sol`
- `b50...e29 ./test/Upgradoor.sol/integration/MigrateToPRV.t.sol`
- `162...112 ./test/Upgradoor.sol/integration/MigrateToARV.t.sol`
- `a30...97b ./test/Upgradoor.sol/integration/Setup.t.sol`
- `cc2...14b ./test/Upgradoor.sol/integration/PreviewAggregationIfLocksHaveExpired.t.sol`
- `0a9...f76 ./test/Upgradoor.sol/integration/ARVNoBoostValidLocks.t.sol`
- `77d...2e0 ./test/Upgradoor.sol/integration/PreviewSingleLock.t.sol`
- `27d...a4e ./test/Upgradoor.sol/integration/GetAmountAndLongestDuration.t.sol`
- `8ff...5a2 ./test/Upgradoor.sol/integration/MigrateSingleLockARV.t.sol`
- `84c...1f9 ./test/PRV/PRVBase.t.sol`
- `d0f...17b ./test/PRV/PRV.t.sol`
- `398...1a6 ./test/PRV/bitfields.t.sol`
- `594...a45 ./test/PRV/PRVRouter.t.sol`
- `f52...f62 ./test/PRV/verifier/PRVMerkleVerifier.t.sol`
- `3ac...a21 ./test/PRV/verifier/PRVIntegrated.t.sol`
- `cf1...4a3 ./test/PRV/verifier/PRVVerifierLiveTree.t.sol`
- `ac8...f36 ./test/PRV/rollStaker/RollStakerTestInitializer.sol`
- `685...6f1 ./test/PRV/rollStaker/RollStaker.t.sol`
- `2c0...aa0 ./test/PRV/rollStaker/RollStaker.scenario.t.sol`
- `1dc...a43 ./test/PRV/rollStaker/invariant/RollStakerNoUpgrade.sol`
- `5c4...a04 ./test/PRV/rollStaker/invariant/RollStaker.invariant.t.sol`
- `b64...046 ./test/rewards/MerkleTreeInitializer.sol`
- `7d8...357 ./test/rewards/MerkleTreeUpgrades.t.sol`
- `0ec...df3 ./test/rewards/RewardsDelegation.t.sol`
- `f1b...a3f ./test/rewards/MerkleDistributorReentrant.t.sol`
- `617...1cd ./test/rewards/MerkleDistributor.t.sol`
- `0a0...2bb ./test/Auxo.sol/TransferAdmin.t.sol`
- `99d...7d6 ./test/Auxo.sol/MintBurn.t.sol`
- `b16...a1e ./test/Auxo.sol/Permit.t.sol`
- `1dc...e30 ./test/Auxo.sol/Setup.sol`
- `2c6...438 ./test/Governor/GovSetup.t.sol`
- `c2e...f20 ./test/TokenLocker.sol/EarlyTerminationPenalty.t.sol`
- `41e...869 ./test/TokenLocker.sol/IncreaseLock.t.sol`
- `e06...332 ./test/TokenLocker.sol/InitialState.t.sol`
- `5d5...8b8 ./test/TokenLocker.sol/Migrate.t.sol`
- `f5f...a78 ./test/TokenLocker.sol/EarlyTermination.sol`
- `0b7...774 ./test/TokenLocker.sol/Eject.t.sol`
- `7d5...288 ./test/TokenLocker.sol/AdminSetter.t.sol`
- `5b7...8c9 ./test/TokenLocker.sol/Deposits.t.sol`
- `25a...a24 ./test/TokenLocker.sol/Access.t.sol`

- `c1e...128 ./test/TokenLocker.sol/Setup.t.sol`
- `d46...fc6 ./test/TokenLocker.sol/Withdraw.t.sol`
- `e6a...db3 ./test/TokenLocker.sol/EmergencyWithdraw.t.sol`
- `380...df4 ./test/TokenLocker.sol/Migrator.t.sol`
- `e2e...63b ./test/TokenLocker.sol/Boost.t.sol`
- `198...884 ./test/TokenLocker.sol/IncreaseAmountsFor.t.sol`
- `527...c7d ./test/TokenLocker.sol/invariant/Migrator.sol`
- `793...1de ./test/TokenLocker.sol/invariant/LockerNonUpgradeable.sol`
- `90c...33d ./test/TokenLocker.sol/invariant/EarlyTermination.sol`
- `6f7...172 ./test/TokenLocker.sol/invariant/Locker.Invariant.t.sol`
- `225...9fc ./test/fork/TestUpgradoorFork.t.sol`
- `2f5...9be ./test/fork/jailwarden/ISafe130.sol`
- `a07...4fd ./test/fork/jailwarden/Sig.t.sol`
- `70a...094 ./test/mocks/Token.sol`
- `5bc...9ed ./test/mocks/MockMigrator.sol`
- `7c1...99c ./test/mocks/SharesTimeLockMock.sol`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:
- Slither v0.9.3

Steps taken to run the tools:
1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

Slither found 61 issues in the two files, which have been either identified as false-positives or integrated into the report.

Test Suite Results

The test were executed with `make test-unit`. All tests pass successfully. Overall, the test quality is high and covers the codebase well.

```
forge test --no-match-path "test/fork/**/*.sol"
[⠒] Compiling...
No files changed, compilation skipped

Running 1 test for test/TokenLocker.sol/Access.t.sol:TestlockerAccess
[PASS] testAdminGetter() (gas: 20849)
Test result: ok. 1 passed; 0 failed; finished in 3.51ms

Running 1 test for test/ARV.sol/MintBurn.t.sol:TestMintBurn
[PASS] testFuzz_RestrictedMintBurn(address) (runs: 256, μ: 13806, ~: 13806)
Test result: ok. 1 passed; 0 failed; finished in 15.38ms

Running 1 test for test/Auxo.sol/MintBurn.t.sol:TestMintBurn
[PASS] testFuzz_RestrictedMint(address,address,uint256) (runs: 256, μ: 131996, ~: 134329)
Test result: ok. 1 passed; 0 failed; finished in 138.31ms

Running 2 tests for test/TokenLocker.sol/EmergencyWithdraw.t.sol:TestlockerEmergencyWithdraw
[PASS] testEmergencyReverts() (gas: 32856)
[PASS] testEmergencyWithdraw(address,uint128,uint8) (runs: 256, μ: 250782, ~: 250773)
Test result: ok. 2 passed; 0 failed; finished in 188.08ms
```

```
Running 1 test for test/PRV/verifier/PRVVerifierLiveTree.t.sol:PRVTestCoreAndVerifier
[PASS] testClaimVerifierWithRealTree() (gas: 471655)
Test result: ok. 1 passed; 0 failed; finished in 2.72ms

Running 2 tests for test/Auxo.sol/Permit.t.sol:TestPermit
[PASS]
testFuzz_MalformedMessageWillNotFailSilently(uint128,address,uint256,uint256,uint8,bytes32,bytes32)
(runs: 256, μ: 37563, ~: 37563)
[PASS] testFuzz_Permit(uint128,address,uint256,uint256) (runs: 256, μ: 116288, ~: 117416)
Test result: ok. 2 passed; 0 failed; finished in 290.77ms

Running 5 tests for test/TokenLocker.sol/AdminSetter.t.sol:TestlockerAdminSetter
[PASS] testEmergencyUnlock() (gas: 19925)
[PASS] testFuzz_AdminFunctionNotCallableByNonAdmin(address) (runs: 256, μ: 178553, ~: 178553)
[PASS] testFuzz_SetEjectBuffer(uint32) (runs: 256, μ: 25224, ~: 25224)
[PASS] testFuzz_SetMinLock(uint192) (runs: 256, μ: 25226, ~: 25226)
[PASS] testFuzz_SetWhiteListed(address,bool) (runs: 256, μ: 29605, ~: 23853)
Test result: ok. 5 passed; 0 failed; finished in 452.00ms

Running 2 tests for test/ARV.sol/Permit.t.sol:TestPermit
[PASS] testFuzz_Permit(uint128,address,uint256,uint256) (runs: 256, μ: 48146, ~: 48146)
[PASS] testFuzz_PermitDelegate(uint128,uint128,uint256) (runs: 256, μ: 78345, ~: 78345)
Test result: ok. 2 passed; 0 failed; finished in 478.24ms

Running 1 test for
test/Upgradoor.sol/integration/GetAmountAndLongestDuration.t.sol:TestGetAmountAndLongestDuration
[PASS] testFuzz_GetAmountAndLongestDuration(address,uint8[5],uint128[5]) (runs: 256, μ: 644572, ~:
644272)
Test result: ok. 1 passed; 0 failed; finished in 363.57ms

Running 3 tests for test/Upgradoor.sol/Getters.t.sol:TestGetters
[PASS] testGetAmountAndLongestDuration() (gas: 239142)
[PASS] testGetMonthsNewLock(uint32) (runs: 256, μ: 12684, ~: 12696)
[PASS] testGetOldLock() (gas: 47871)
Test result: ok. 3 passed; 0 failed; finished in 28.48ms

Running 6 tests for test/TokenLocker.sol/Migrate.t.sol:TestlockerMigrate
[PASS] testFuzz_CannotMigrateByDefault(address) (runs: 256, μ: 23853, ~: 23853)
[PASS] testFuzz_CannotMigrateIfAmountIsZero(address) (runs: 256, μ: 75527, ~: 75527)
[PASS] testFuzz_CannotMigrateIfLockIsExpired(address) (runs: 256, μ: 269884, ~: 269894)
[PASS] testFuzz_CannotMigrateIfMigratorIsNotSet(address) (runs: 256, μ: 264756, ~: 264756)
[PASS] testFuzz_Migrate(address,uint128,uint8) (runs: 256, μ: 279224, ~: 279225)
[PASS] testFuzz_OnlyMigratorCanCallMigrate(address) (runs: 256, μ: 267717, ~: 267717)
Test result: ok. 6 passed; 0 failed; finished in 603.81ms

Running 1 test for test/Upgradoor.sol/integration/ARVNoBoostValidLocks.t.sol:TestARVNoBoostValidLocks
[PASS] testFuzz_VeAuxoNoBoostValidLocks(address,uint8[5],uint128[5],uint256,address) (runs: 256, μ:
808194, ~: 809510)
Test result: ok. 1 passed; 0 failed; finished in 672.52ms

Running 3 tests for test/TokenLocker.sol/InitialState.t.sol:TestlockerInitialState
[PASS] testInitialStateRatioArray(address,address,uint32,uint32,uint192) (runs: 256, μ: 5049052, ~:
5049052)
[PASS] testInitialStateRevertsMinGteMax(address,address,uint32,uint32,uint192) (runs: 256, μ: 4890508, ~:
4890508)
[PASS] testInitialStateVariables(address,address,uint32,uint32,uint192,address,address) (runs: 256, μ:
4980841, ~: 4980841)
Test result: ok. 3 passed; 0 failed; finished in 744.62ms

Running 1 test for test/TokenLocker.sol/EarlyTermination.sol:TestlockerTerminateEarly
[PASS] testFuzz_TerminateEarly(address,uint128,uint8,address,uint256) (runs: 256, μ: 361507, ~: 358607)
Test result: ok. 1 passed; 0 failed; finished in 364.52ms

Running 1 test for test/rewards/MerkleDistributorReentrant.t.sol:TestDistributor
[PASS] testNoReentrant() (gas: 149057)
Test result: ok. 1 passed; 0 failed; finished in 5.46ms

Running 3 tests for test/rewards/MerkleTreeUpgrades.t.sol:MerkleDistributorUpgradeTest
[PASS] testCannotInitializeTheImplementation() (gas: 1979248)
[PASS] testNoReinitialize() (gas: 16455)
[PASS] testUpgrade() (gas: 2108492)
Test result: ok. 3 passed; 0 failed; finished in 1.43ms
```

```
Running 1 test for test/Governor/GovSetup.t.sol:GovSetupTest
[PASS] testSettingControllerRoles(uint256,address,address) (runs: 256, μ: 2078834, ~: 2079767)
Test result: ok. 1 passed; 0 failed; finished in 443.41ms

Running 2 tests for test/TokenLocker.sol/Boost.t.sol:TestlockerBoost
[PASS] testFuzz_BoostToMax(address,uint128,uint8) (runs: 256, μ: 262908, ~: 263027)
[PASS] testSmallQtyIncreaseDoesNotBrickBoost(address,uint128,uint128,uint8) (runs: 256, μ: 306116, ~:
306254)
Test result: ok. 2 passed; 0 failed; finished in 675.11ms

Running 4 tests for test/TokenLocker.sol/EarlyTerminationPenalty.t.sol:TestEarlyTermination
[PASS] testCannotSetBeneficiaryToZero() (gas: 24153)
[PASS] testFuzz_CannotSetInvalidPenalty(uint256) (runs: 256, μ: 25931, ~: 25931)
[PASS] testFuzz_SetPenalty(uint256,address) (runs: 256, μ: 82535, ~: 82535)
[PASS] testFuzz_SetPenaltyBeneficiary(address,address) (runs: 256, μ: 81275, ~: 81275)
Test result: ok. 4 passed; 0 failed; finished in 431.24ms

Running 1 test for
test/Upgradoor.sol/integration/PreviewAggregationIfLocksHaveExpired.t.sol:TestPreviewAggregationIfLocksHa
veExpired
[PASS]
testFuzz_CorrectPreviewAggregationIfLocksHaveExpired(address,uint8[5],uint128[5],uint64,uint256,address)
(runs: 256, μ: 795820, ~: 797723)
Test result: ok. 1 passed; 0 failed; finished in 853.07ms

Running 1 test for test/Upgradoor.sol/integration/MigrateSingleLockARV.t.sol:TestMigrateSingleLockARV
[PASS] testFuzz_SingleLockVeAuxoNoExpiry(address,address[5],uint8[5],uint128[5]) (runs: 256, μ: 1593093,
~: 1593039)
Test result: ok. 1 passed; 0 failed; finished in 1.59s

Running 7 tests for test/Upgradoor.sol/MigrateToARV.t.sol:MigrateToARV
[PASS] testAggregateAndBoost() (gas: 567344)
[PASS] testPreviewAggregateAndBoost() (gas: 635077)
[PASS] testPreviewupgradeSingleLockARV() (gas: 575234)
[PASS] testUpgradeFailIfReceiverHasVeDOUGH() (gas: 61349)
[PASS] testaggregateToARV() (gas: 648433)
[PASS] testpreviewAggregateARV() (gas: 795374)
[PASS] testupgradeSingleLockARV() (gas: 6015373)
Test result: ok. 7 passed; 0 failed; finished in 51.67ms

Running 3 tests for test/TokenLocker.sol/invariant/Locker.Invariant.t.sol:RollStakerInvariantTest
[PASS] invariantTestNobodyWithoutALockHasARewardBalanceAndViceVersa() (runs: 256, calls: 3840, reverts:
3399)
[PASS] invariantTestRewardAlwaysEqExpected() (runs: 256, calls: 3840, reverts: 3399)
[PASS] invariantTestRewardTotalSupplyLEDepositTokenLocked() (runs: 256, calls: 3840, reverts: 3399)
Test result: ok. 3 passed; 0 failed; finished in 1.85s

Running 2 tests for test/Upgradoor.sol/integration/PreviewSingleLock.t.sol:PreviewSingleLock
[PASS]
testFuzz_PreviewSingleLockVeAndXAuxoWithExpiry(address,address[5],uint8[5],uint128[5],bool[5],uint256,add
ress,uint64) (runs: 256, μ: 1172035, ~: 1350528)
[PASS] testFuzz_PreviewSingleLockXAuxoNoExpiry(address,address[5],uint8[5],uint128[5],uint256,address)
(runs: 256, μ: 1412073, ~: 1413232)
Test result: ok. 2 passed; 0 failed; finished in 2.41s

Running 2 tests for test/TokenLocker.sol/Migrator.t.sol:TestMigrator
[PASS] testFuzz_SetMigration(bool,address) (runs: 256, μ: 80761, ~: 80761)
[PASS] testFuzz_SetMigrator(address,address) (runs: 256, μ: 80789, ~: 80789)
Test result: ok. 2 passed; 0 failed; finished in 339.12ms

Running 1 test for test/ARV.sol/NonTransferability.t.sol:TestNonTransferability
[PASS] testFuzz_CannotBeTransferred(address) (runs: 256, μ: 111222, ~: 111222)
Test result: ok. 1 passed; 0 failed; finished in 24.83ms

Running 9 tests for test/TokenLocker.sol/Deposits.t.sol:TestlockerDeposits
[PASS] testFuzz_CannotDepositOnBehalfOfAnotherUnlessWhitelisted(address,uint128,uint8,address) (runs:
256, μ: 265648, ~: 265648)
[PASS] testFuzz_CannotDepositToContractUnlessWhitelisted(address,uint128,uint8) (runs: 256, μ: 459276, ~:
459276)
[PASS] testFuzz_CannotDepositTwice(address,uint128,uint8) (runs: 256, μ: 235475, ~: 235495)
[PASS] testFuzz_DepositRevertsBelowMin(address,uint192,uint8,uint192) (runs: 256, μ: 111098, ~: 113604)
```

```
[PASS] testFuzz_DepositRevertsOutOfRangeMonths(address,uint128,uint8,uint8,uint8) (runs: 256, μ: 5024470,
~: 5024550)
[PASS] testFuzz_DepositWithSignature(uint128,uint128,uint8,uint24,uint256) (runs: 256, μ: 252330, ~:
252330)
[PASS] testFuzz_DepositWithSignatureToExternalReceiver(uint128,address,uint128,uint8,uint24,uint256)
(runs: 256, μ: 293478, ~: 293478)
[PASS] testFuzz_HasLock(address,uint128,uint8) (runs: 256, μ: 236180, ~: 236180)
[PASS] testFuzz_SuccessfulDeposit(address,uint128,uint8,uint24) (runs: 256, μ: 240427, ~: 240427)
Test result: ok. 9 passed; 0 failed; finished in 3.01s

Running 8 tests for test/TokenLocker.sol/IncreaseAmountsFor.t.sol:TestlockerIncreaseAmountFor
[PASS] testFuzz_AllReceiversNeedALock(uint120,uint184[5]) (runs: 256, μ: 747503, ~: 747516)
[PASS] testFuzz_CanIncreaseAmountForMany(uint120,uint128[5],uint8[5]) (runs: 256, μ: 806845, ~: 806958)
[PASS] testFuzz_CannotHaveAZeroAmount(uint120,uint128[5]) (runs: 256, μ: 765506, ~: 768355)
[PASS] testFuzz_CannotHaveDifferentLenghtParam(address[],uint192[]) (runs: 256, μ: 163208, ~: 161885)
[PASS] testFuzz_CannotIncreaseAmountForExpiredLock(address,uint128,uint8) (runs: 256, μ: 358163, ~:
358163)
[PASS] testFuzz_CannotIncreaseAmountForManyBelowMin(address,uint128,uint8) (runs: 256, μ: 360993, ~:
360993)
[PASS] testFuzz_DepositorCanUseIncreaseAmount(address,uint128,uint8) (runs: 256, μ: 273995, ~: 274005)
[PASS] testFuzz_DepositorCannotUseIncreaseAmountForMany(address,uint128,uint8) (runs: 256, μ: 341835, ~:
341835)
Test result: ok. 8 passed; 0 failed; finished in 3.22s

Running 8 tests for test/rewards/RewardsDelegation.t.sol:TestDistributorDelegate
[PASS] testCanAddRemoveDelegate(address,address) (runs: 256, μ: 72209, ~: 72209)
[PASS] testCannotClaimIfInvalidDelegate(address) (runs: 256, μ: 451097, ~: 451097)
[PASS] testCannotClaimIfSomeoneElsesDelegate(address,address) (runs: 256, μ: 530572, ~: 530572)
[PASS] testCannotDelegateUnlessWhiteListed(address,address) (runs: 256, μ: 81749, ~: 81749)
[PASS] testDelegatedClaim(address) (runs: 256, μ: 505754, ~: 505754)
[PASS] testInvalidDelegatedMultiClaim(address,address) (runs: 256, μ: 545110, ~: 545110)
[PASS] testNoEmptyClaims(address) (runs: 256, μ: 69080, ~: 69080)
[PASS] testSuccessfulDelegatedMultiClaim(address) (runs: 256, μ: 576689, ~: 576689)
Test result: ok. 8 passed; 0 failed; finished in 831.00ms

Running 5 tests for test/Upgradoor.sol/integration/MigrateToARV.t.sol:TestMigrateToARV
[PASS] testFuzz_ExitToVeAuxoBoosted(address,uint8[5],uint128[5]) (runs: 256, μ: 856636, ~: 856413)
[PASS] testFuzz_ExitToVeAuxoBoostedWithExpiry(address,uint8[5],uint128[5],uint64) (runs: 256, μ: 791326,
~: 869309)
[PASS] testFuzz_ExitToVeAuxoNonBoosted(address,uint8[5],uint128[5]) (runs: 256, μ: 879805, ~: 879526)
[PASS] testFuzz_ExitToVeAuxoNonBoostedWithExpiry(address,uint8[5],uint128[5],uint64) (runs: 256, μ:
813027, ~: 892323)
[PASS] testFuzz_migrateNoLocksRevertsGracefully() (gas: 425277)
Test result: ok. 5 passed; 0 failed; finished in 2.14s

Running 6 tests for test/Upgradoor.sol/MigrateToPRV.t.sol:TestAggregateToPRV
[PASS] testPreviewaggregateToPRV() (gas: 564915)
[PASS] testaggregateToPRV() (gas: 489378)
[PASS] testaggregateToPRVAndStake() (gas: 655836)
[PASS] testpreviewUpgradeSingleLockPRV() (gas: 477837)
[PASS] testupgradeSingleLockPRV() (gas: 3843755)
[PASS] testupgradeSingleLockPRVAndStake() (gas: 6671647)
Test result: ok. 6 passed; 0 failed; finished in 56.43ms

Running 17 tests for test/PRV/PRV.t.sol:PRVTestCore
[PASS] testDepositFor(uint184,address,address) (runs: 256, μ: 160924, ~: 160924)
[PASS] testDepositForSignatureInValid(uint128,address,uint128,uint256,uint8,bytes32,bytes32) (runs: 256,
μ: 100705, ~: 101614)
[PASS] testDepositForSignatureValid(uint128,address,uint128,uint256) (runs: 256, μ: 176693, ~: 176693)
[PASS] testFeeIsNotChargedOnDeposit(uint256,uint184) (runs: 256, μ: 138339, ~: 140749)
[PASS] testFuzz_WithdrawalFeeNotChargedIfNoBeneficiarySet(uint256,uint128) (runs: 256, μ: 1946688, ~:
1948018)
[PASS] testFuzz_canWithdraw(address,address,uint256,uint256,uint256) (runs: 256, μ: 229391, ~: 228947)
[PASS] testInitialState(address,address,uint256,address,address) (runs: 256, μ: 2432877, ~: 2434199)
[PASS] testInitialStateReverts(address,address,uint256,address,address) (runs: 256, μ: 2251142, ~:
2251142)
[PASS] testNoZeroWithdrawals() (gas: 19202)
[PASS] testOnlyGovernance(address) (runs: 256, μ: 70594, ~: 70594)
[PASS] testSetFeeBeneficiary(address) (runs: 256, μ: 42654, ~: 42654)
[PASS] testSetFeeBeneficiaryRevertsZeroAddr() (gas: 16032)
[PASS] testSetFeePolicy(address,uint256) (runs: 256, μ: 66199, ~: 67910)
[PASS] testSetFeeRevertAboveMax(uint256) (runs: 256, μ: 18231, ~: 18231)
```

```
[PASS] testSetGovernor(address) (runs: 256, μ: 23608, ~: 23608)
[PASS] testSetGovernorRevertsZeroAddr() (gas: 16043)
[PASS] testWillRevertOnWithdrawalFailure() (gas: 194231)
Test result: ok. 17 passed; 0 failed; finished in 1.19s

Running 7 tests for test/PRV/rollStaker/invariant/RollStaker.invariant.t.sol:RollStakerInvariantTest
[PASS] invariantTestActiveBalanceEqActive() (runs: 256, calls: 3840, reverts: 3010)
[PASS] invariantTestActivePlusPendingIsAlwaysEqDeposited() (runs: 256, calls: 3840, reverts: 3010)
[PASS] invariantTestEpochPendingGEPendingDeposits() (runs: 256, calls: 3840, reverts: 3010)
[PASS] invariantTestNotActiveThisEpochActiveNextEqPendingBalanceEqTotal() (runs: 256, calls: 3840,
reverts: 3010)
[PASS] invariantTestTokenBalanceContactGEContractLocked() (runs: 256, calls: 3840, reverts: 3010)
[PASS] invariantTestTotalInContractAlwaysGeUserTotal() (runs: 256, calls: 3840, reverts: 3010)
[PASS] invariantTestZeroBalanceEqInactive() (runs: 256, calls: 3840, reverts: 3010)
Test result: ok. 7 passed; 0 failed; finished in 783.56ms

Running 2 tests for test/Upgradoor.sol/integration/MigrateToPRV.t.sol:TestMigrateToPRV
[PASS] testFuzz_ExitToXAuxo(address,uint8[5],uint128[5],uint256,address) (runs: 256, μ: 839865, ~:
840908)
[PASS] testFuzz_ExitToXAuxoWithExpiry(address,uint8[5],uint128[5],uint64,uint256,address) (runs: 256, μ:
825947, ~: 870672)
Test result: ok. 2 passed; 0 failed; finished in 1.13s

Running 2 tests for test/TokenLocker.sol/Eject.t.sol:TestlockerEject
[PASS] testFuzz_CanEjectGetter(address,uint128,uint8,uint32,address) (runs: 256, μ: 259323, ~: 259323)
[PASS] testFuzz_canEject(address[2],uint128[2],uint8[2],uint32) (runs: 256, μ: 406232, ~: 406230)
Test result: ok. 2 passed; 0 failed; finished in 5.06s

Running 2 tests for test/Auxo.sol/TransferAdmin.t.sol:TestTransferAdmin
[PASS] testFuzz_ChangingAuxoAdmin(address,address) (runs: 256, μ: 90764, ~: 90764)
[PASS] testFuzz_TransferOfAdminRole(address) (runs: 256, μ: 44794, ~: 44794)
Test result: ok. 2 passed; 0 failed; finished in 163.41ms

Running 6 tests for test/Upgrades.t.sol:TestUpgrades
[PASS] testCannotInitializeImplementationContract() (gas: 3585498)
[PASS] testCannotReinitAfterUpgrading() (gas: 3769560)
[PASS] testCollision(address) (runs: 256, μ: 3599570, ~: 3599570)
[PASS] testNoReinitialize() (gas: 21392)
[PASS] testOnlyAdminCanSetImplementation(address) (runs: 256, μ: 3591149, ~: 3591149)
[PASS] testUpgrade() (gas: 3792241)
Test result: ok. 6 passed; 0 failed; finished in 228.10ms

Running 3 tests for test/TokenLocker.sol/Withdraw.t.sol:TestlockerWithdraw
[PASS] testFuzz_CanWithdrawOtherwise(address,uint128,uint8,uint32) (runs: 256, μ: 243540, ~: 243547)
[PASS] testFuzz_CannotMakeEmptyWithdraw() (gas: 15950)
[PASS] testFuzz_CannotWithdrawEarly(address,uint128,uint8,uint32) (runs: 256, μ: 232223, ~: 232233)
Test result: ok. 3 passed; 0 failed; finished in 770.52ms

Running 1 test for test/PRV/rollStaker/RollStaker.scenario.t.sol:RollStakerScenarioTest
[PASS] testKitchenSink(uint8,uint96[10],uint96[10]) (runs: 256, μ: 2129528, ~: 2134053)
Test result: ok. 1 passed; 0 failed; finished in 2.97s

Running 5 tests for test/PRV/PRVRouter.t.sol:PRVTestRouter
[PASS] testFuzz_CanConvertAndStake(address,uint120) (runs: 256, μ: 300591, ~: 300586)
[PASS] testFuzz_CanConvertAndStakeWithFee(address,uint120,uint256,address) (runs: 256, μ: 341824, ~:
342858)
[PASS] testFuzz_CanConvertAndStakeWithReceiver(address,address,uint120) (runs: 256, μ: 304254, ~: 304228)
[PASS]
testFuzz_convertAndStakeWithSignatureInValid(uint128,address,uint120,uint256,uint8,bytes32,bytes32)
(runs: 256, μ: 96070, ~: 97056)
[PASS] testFuzz_convertAndStakeWithSignatureValid(uint128,address,uint120,uint256) (runs: 256, μ: 334935,
~: 334939)
Test result: ok. 5 passed; 0 failed; finished in 739.65ms

Running 1 test for test/PRV/verifier/PRVIntegrated.t.sol:PRVTestCoreAndVerifier
[PASS] testFuzz_cannotClaimMorePRVThanInWallet(uint256,bytes32,uint32,uint32,uint256,address,
(uint256,uint256,bytes32[],address)) (runs: 256, μ: 1576770, ~: 1574890)
Test result: ok. 1 passed; 0 failed; finished in 3.90s

Running 22 tests for test/PRV/verifier/PRVMerkleVerifier.t.sol:PRVTestVerifier
[PASS] testCanHandleClaimsTooSmall(bytes) (runs: 256, μ: 21644, ~: 21634)
[PASS] testFuzz_Ownable(address) (runs: 256, μ: 54315, ~: 54315)
```

```
[PASS] testFuzz_amountGtClaimReverts(uint256,bytes32,uint32,uint32,uint256,
(uint256,uint256,bytes32[],address)) (runs: 256, μ: 1370693, ~: 1370188)
[PASS] testFuzz_canMakeMultipleClaims(uint256,bytes32,uint32,uint32,uint256,
(uint256,uint256,bytes32[],address)) (runs: 256, μ: 1515877, ~: 1516473)
[PASS] testFuzz_canSetNewWindowAfterDeletingPrevious(uint256,bytes32,uint32,uint32) (runs: 256, μ:
155119, ~: 157224)
[PASS] testFuzz_cannotClaimPastExhaust(uint256,bytes32,uint32,uint32,(uint256,uint256,bytes32[],address))
(runs: 256, μ: 1452663, ~: 1451264)
[PASS] testFuzz_cannotClaimPastTotalBudget(uint256,bytes32,uint32,uint32,
(uint256,uint256,bytes32[],address)) (runs: 256, μ: 1455376, ~: 1456279)
[PASS] testFuzz_cannotClaimPastTotalForUser(uint256,bytes32,uint32,uint32,uint256,
(uint256,uint256,bytes32[],address)) (runs: 256, μ: 1460524, ~: 1458490)
[PASS] testFuzz_cannotProcessClaimWhenPaused() (gas: 42906)
[PASS] testFuzz_cannotSetWindowWithEndBeforeStart(uint256,bytes32,uint32,uint32) (runs: 256, μ: 17245, ~:
17245)
[PASS] testFuzz_cannotSetWindowWithoutSufficientAuxo(uint256,bytes32,uint32,uint32) (runs: 256, μ:
170628, ~: 172313)
[PASS] testFuzz_claimMustBeCorrectWindow(uint256,bytes32,uint32,uint32,uint256) (runs: 256, μ: 110464, ~:
110657)
[PASS] testFuzz_claimMustBeInBudget(uint256,bytes32,uint32,uint32,uint256) (runs: 256, μ: 131419, ~:
132430)
[PASS] testFuzz_claimMustBeInsideWindow(uint256,bytes32,uint32,uint32,uint256,uint256) (runs: 256, μ:
131716, ~: 131854)
[PASS] testFuzz_claimMustBeValid(uint256,bytes32,uint32,uint32,uint256,
(uint256,uint256,bytes32[],address)) (runs: 256, μ: 226090, ~: 227611)
[PASS] testFuzz_claimNeverLessThan192Bytes((uint256,uint256,bytes32[],address)) (runs: 256, μ: 23252, ~:
23910)
[PASS] testFuzz_claimSuccess(uint256,bytes32,uint32,uint32,uint256,(uint256,uint256,bytes32[],address))
(runs: 256, μ: 1468952, ~: 1472515)
[PASS] testFuzz_createDeleteEventsWindow(uint256,bytes32,uint32,uint32) (runs: 256, μ: 98558, ~: 99296)
[PASS] testFuzz_onlyPRVCanCall(uint256,address) (runs: 256, μ: 23287, ~: 23287)
[PASS] testFuzz_setNewWindowDeletesPrevious(uint256,bytes32,uint32,uint32) (runs: 256, μ: 158582, ~:
159820)
[PASS] testFuzz_zeroClaimOKwithZeroAmount(uint256,bytes32,uint32,uint32,
(uint256,uint256,bytes32[],address)) (runs: 256, μ: 1431363, ~: 1429861)
[PASS] testFuzz_zeroClaimRevertsWithPositiveAmount(uint256,bytes32,uint32,uint32,uint256,
(uint256,uint256,bytes32[],address)) (runs: 256, μ: 1370021, ~: 1368969)
Test result: ok. 22 passed; 0 failed; finished in 6.78s

Running 6 tests for test/Upgradoor.sol/sharesTimelock/Migrate.t.sol:TestSharesTimelock
[PASS] testCannotMigrateTwice(address,uint8[10],uint128[10]) (runs: 256, μ: 1284139, ~: 1287082)
[PASS] testMigrate(address,uint8[10],uint128[10]) (runs: 256, μ: 1673471, ~: 1677485)
[PASS] testMigrateMany(address,uint8[10],uint128[10],uint8[]) (runs: 256, μ: 1160787, ~: 1147928)
[PASS] testMigrateManyReverts(address,uint8[10],uint128[10],uint8[]) (runs: 256, μ: 1144502, ~: 1146677)
[PASS] testMigrateReverts(address,address,uint256,uint8[10],uint128[10]) (runs: 256, μ: 1084301, ~:
1083876)
[PASS] testOwnable(address) (runs: 256, μ: 35512, ~: 35512)
Test result: ok. 6 passed; 0 failed; finished in 12.12s

Running 3 tests for test/PRV/bitfields.t.sol:TestBitfields
[PASS] testBitFieldLib(uint8,uint8,uint8) (runs: 256, μ: 6153320, ~: 7050651)
[PASS] testLastActiveDoesNotModifyLocalVariable(uint8,uint8,uint8) (runs: 256, μ: 35007, ~: 31219)
[PASS] testRepeatedActivationsAndDeactivationsAreIdempotent(uint8,uint8,uint8) (runs: 256, μ: 36175, ~:
33842)
Test result: ok. 3 passed; 0 failed; finished in 7.26s

Running 12 tests for test/TokenLocker.sol/IncreaseLock.t.sol:TestlockerIncreaseLock
[PASS] testCannotGovernanceAttack(address) (runs: 256, μ: 300336, ~: 300336)
[PASS] testFuzz_CanIncreaseLockDuration(address,uint128,uint8,uint8,uint32) (runs: 256, μ: 265078, ~:
265078)
[PASS] testFuzz_CanIncreaseLockQty(address,uint128,uint8,uint128,uint32) (runs: 256, μ: 274751, ~:
274781)
[PASS] testFuzz_CanIncreaseQtyWithSig(uint128,uint128,uint128,uint8,uint256,uint32) (runs: 256, μ:
302349, ~: 302349)
[PASS] testFuzz_CannotIncreaseLockMoreThanMax(address,uint128,uint8,uint8) (runs: 256, μ: 234452, ~:
234472)
[PASS] testFuzz_CannotIncreaseLockQtyBelowMin(address,uint128,uint8) (runs: 256, μ: 268941, ~: 268940)
[PASS] testFuzz_CannotIncreaseLockWithZeroChange(address,uint128,uint8) (runs: 256, μ: 239196, ~: 239196)
[PASS] testFuzz_CannotIncreaseLockWithoutDeposit(address,uint128,uint8) (runs: 256, μ: 49250, ~: 49250)
[PASS] testFuzz_TerminateEarlyRevertsWithoutxAUXO(address,uint128,uint8) (runs: 256, μ: 236641, ~:
236641)
[PASS] testFuzz_repeatedIncreasesDoNotCauseRoundingErrors(address,uint128,uint8,uint128[50]) (runs: 256,
```

```
μ: 1106874, ~: 1107057)
[PASS] testFuzz_repeatedIncreasesDurationDoNotCauseRoundingErrors(address,uint128) (runs: 256, μ: 775374,
~: 775384)
[PASS] testFuzz_repeatedIncreasesInBothDoesntCauseIssues(address,uint128,uint128[50],bool[50]) (runs:
256, μ: 1088386, ~: 1092086)
Test result: ok. 12 passed; 0 failed; finished in 10.16s

Running 17 tests for test/rewards/MerkleDistributor.t.sol:TestDistributor
[PASS] testBadMultiClaim((uint256,uint256,uint256,address,bytes32[],address)[],address,address) (runs:
256, μ: 9503741, ~: 8240409)
[PASS] testCannotClaimForPrevWindow() (gas: 401686)
[PASS] testCannotClaimForTokenInPreviousWindow() (gas: 747805)
[PASS] testCannotMultiClaimForMultipleTokens(address) (runs: 256, μ: 402238, ~: 402238)
[PASS] testCannotMultiClaimForSomeoneElse((uint256,uint256,uint256,address,bytes32[],address)
[],address,address,address) (runs: 256, μ: 9755049, ~: 9195852)
[PASS] testCannotMultiClaimForSomeoneElse(address) (runs: 256, μ: 373982, ~: 373982)
[PASS] testCannotMultiClaimWithPaddedArray() (gas: 374650)
[PASS] testClaim() (gas: 512175)
[PASS] testDeleteWindow(bytes32,string,uint256) (runs: 256, μ: 744110, ~: 746749)
[PASS] testEmergencyWithdraw(uint256,uint256) (runs: 256, μ: 775183, ~: 775261)
[PASS] testInvalidClaims((uint256,uint256,uint256,address,bytes32[],address)) (runs: 256, μ: 114212, ~:
114347)
[PASS] testLockedClaims(uint256,(uint256,uint256,uint256,address,bytes32[],address)) (runs: 256, μ:
100067, ~: 99600)
[PASS] testNoEmptyClaims() (gas: 38701)
[PASS] testOwnable(address) (runs: 256, μ: 33586, ~: 33586)
[PASS] testPausable(address) (runs: 256, μ: 68878, ~: 68878)
[PASS] testSetWindow(bytes32,string,uint256) (runs: 256, μ: 852131, ~: 864299)
[PASS] testSuccessfulMultiClaim() (gas: 461981)
Test result: ok. 17 passed; 0 failed; finished in 18.91s

Running 20 tests for test/PRV/rollStaker/RollStaker.t.sol:RollStakerTest
[PASS] testActivateNewEpoch(uint128,uint128,uint8,uint256) (runs: 256, μ: 336227, ~: 209851)
[PASS] testCanWithdrawAcrossMultipleEpochs(uint120,address) (runs: 256, μ: 274616, ~: 274616)
[PASS] testCannotWithdrawMoreThanDeposited(uint120,address,uint256) (runs: 256, μ: 196897, ~: 196881)
[PASS] testDepositFor(uint120,address,address) (runs: 256, μ: 296311, ~: 296311)
[PASS] testDepositPermit(uint120,uint256,uint128) (runs: 256, μ: 185399, ~: 185399)
[PASS] testDepositRollStaker(uint120,address) (runs: 256, μ: 286792, ~: 286792)
[PASS] testEmergencyWithdraw(address,uint120) (runs: 256, μ: 212763, ~: 212757)
[PASS] testGettersComputeCorrectlyAfterWithdraw(uint120,address) (runs: 256, μ: 246857, ~: 246857)
[PASS] testGettersDoNotRevert(uint8,address) (runs: 256, μ: 27135, ~: 27446)
[PASS] testLastEpochUserWasActive(address,uint8,uint8,uint8) (runs: 256, μ: 634380, ~: 745990)
[PASS] testOperatorRestrictedFunctions(address) (runs: 256, μ: 208748, ~: 208748)
[PASS] testPendingDepositsUpdateCorrectlyMultipleWithdrawal(address,address,uint96,uint96) (runs: 256, μ:
352342, ~: 352317)
[PASS] testPublicFunctionsPaused(uint120,address) (runs: 256, μ: 83563, ~: 83563)
[PASS] testQuit(uint120,address,uint256) (runs: 256, μ: 266456, ~: 266483)
[PASS] testRevertDeposit(uint120,address,uint256) (runs: 256, μ: 210448, ~: 210519)
[PASS] testRevertDepositActiveUserStaysActive(uint120,address) (runs: 256, μ: 256844, ~: 256838)
[PASS] testRevertDepositInActiveUserGoesBackToInactive(uint120,address) (runs: 256, μ: 246399, ~: 246396)
[PASS] testUserStaysActive(address,uint8,uint8,uint8) (runs: 256, μ: 1248510, ~: 1135357)
[PASS] testWithdraw(uint120,address,uint256) (runs: 256, μ: 242689, ~: 242514)
[PASS] testZeroAmountReverts() (gas: 69337)
Test result: ok. 20 passed; 0 failed; finished in 15.41s
```

Code Coverage

The test coverage was executed with `forge coverage`.

Note that the script provided run coverage for test files as well, which decrease the overall coverage. In general, it is recommended to have at least 90% for line and branch coverage.

The newly added `PRVMerkleVerifier` contract has a line coverage that is slightly below our recommendation with 75.68%. The `PRV` contract has a 100% coverage in all of the metrics but the branch coverage, where its also ends up slightly below our recommendation with 85.71%.

File	% Lines	% Statements	% Branches	% Funcs
script/Deploy-v1.s.sol	0.00% (**0/**159)	0.00% (**0/**173)	0.00% (**0/**68)	0.00% (**0/**13)

File	% Lines	% Statements	% Branches	% Funcs
script/HealthCheck.sol	0.00% (**0/**110)	0.00% (**0/**116)	0.00% (**0/**178)	0.00% (**0/**16)
script/ReadTree.s.sol	0.00% (**0/**8)	0.00% (**0/**11)	100.00% (**0/**0)	0.00% (**0/**4)
script/Simulation.s.sol	0.00% (**0/**347)	0.00% (**0/**440)	0.00% (**0/**122)	0.00% (**0/**22)
script/old/DeployOracle.sol	0.00% (**0/**4)	0.00% (**0/**4)	0.00% (**0/**2)	0.00% (**0/**1)
script/old/DeployRewards.s.sol	0.00% (**0/**12)	0.00% (**0/**13)	100.00% (**0/**0)	0.00% (**0/**5)
script/old/EjectVeDOUGHFreeriders.sol	0.00% (**0/**19)	0.00% (**0/**31)	0.00% (**0/**2)	0.00% (**0/**1)
src/ARV.sol	100.00% (**7/**7)	100.00% (**7/**7)	100.00% (**0/**0)	100.00% (**7/**7)
src/AUXO.sol	100.00% (**1/**1)	100.00% (**1/**1)	100.00% (**0/**0)	100.00% (**1/**1)
src/modules/PRV/PRV.sol	100.00% (**50/**50)	100.00% (**56/**56)	85.71% (**24/**28)	100.00% (**13/**13)
src/modules/PRV/PRVMerkleVerifier.sol	75.68% (**28/**37)	78.05% (**32/**41)	93.75% (**15/**16)	80.00% (**12/**15)
src/modules/PRV/PRVRouter.sol	100.00% (**16/**16)	100.00% (**16/**16)	100.00% (**0/**0)	100.00% (**3/**3)
src/modules/PRV/RollStaker.sol	86.08% (**68/**79)	81.91% (**77/**94)	59.09% (**13/**22)	88.89% (**24/**27)
src/modules/PRV/StakingManager.sol	0.00% (**0/**17)	0.00% (**0/**19)	100.00% (**0/**0)	0.00% (**0/**6)
src/modules/PRV/bitfield.sol	100.00% (**13/**13)	100.00% (**15/**15)	100.00% (**2/**2)	100.00% (**7/**7)
src/modules/governance/EarlyTermination.sol	85.71% (**6/**7)	85.71% (**6/**7)	100.00% (**4/**4)	66.67% (**2/**3)
src/modules/governance/Governor.sol	0.00% (**0/**10)	0.00% (**0/**10)	100.00% (**0/**0)	0.00% (**0/**10)
src/modules/governance/IncentiveCurve.sol	66.67% (**2/**3)	66.67% (**2/**3)	100.00% (**0/**0)	66.67% (**2/**3)
src/modules/governance/Migrator.sol	80.00% (**4/**5)	80.00% (**4/**5)	100.00% (**0/**0)	66.67% (**2/**3)
src/modules/governance/TokenLocker.sol	98.45% (**127/**129)	97.99% (**146/**149)	80.00% (**48/**60)	92.86% (**26/**28)
src/modules/reward-**policies/**PolicyManager.sol	0.00% (**0/**9)	0.00% (**0/**14)	0.00% (**0/**2)	0.00% (**0/**3)
src/modules/reward-**policies/**SimpleDecayOracle.sol	0.00% (**0/**2)	0.00% (**0/**3)	100.00% (**0/**0)	0.00% (**0/**1)
src/modules/reward-**policies/policies/**DecayPolicy.sol	0.00% (**0/**14)	0.00% (**0/**23)	0.00% (**0/**6)	0.00% (**0/**3)

File	% Lines	% Statements	% Branches	% Funcs
src/modules/rewards/DelegationRegistry.sol	90.00% (**9/**10)	90.00% (**9/**10)	100.00% (**2/**2)	80.00% (**4/**5)
src/modules/rewards/MerkleDistributor.sol	96.88% (**62/**64)	97.33% (**73/**75)	86.36% (**19/**22)	100.00% (**17/**17)
src/modules/vedough-**bridge/**SharesTimeLock.sol	58.14% (**75/**129)	60.26% (**94/**156)	45.59% (**31/**68)	46.88% (**15/**32)
src/modules/vedough-**bridge/**Upgradoor.sol	100.00% (**104/**104)	99.33% (**149/**150)	79.41% (**27/**34)	100.00% (**19/**19)
test/ARV.**sol/**Setup.sol	0.00% (**0/**1)	0.00% (**0/**1)	100.00% (**0/**0)	0.00% (**0/**1)
test/Auxo.**sol/**Setup.sol	0.00% (**0/**1)	0.00% (**0/**1)	100.00% (**0/**0)	0.00% (**0/**1)
test/PRV/PRV.t.sol	100.00% (**2/**2)	100.00% (**2/**2)	100.00% (**0/**0)	100.00% (**2/**2)
test/PRV/PRVBase.t.sol	21.43% (**3/**14)	17.65% (**3/**17)	100.00% (**0/**0)	40.00% (**2/**5)
test/PRV/rollStaker/RollStakerTestInitializer.sol	53.33% (**8/**15)	44.44% (**8/**18)	100.00% (**0/**0)	80.00% (**4/**5)
test/PRV/rollStaker/invariant/RollStaker.invariant.t.sol	0.00% (**0/**61)	0.00% (**0/**82)	0.00% (**0/**30)	0.00% (**0/**8)
test/PRV/rollStaker/invariant/RollStakerNoUpgrade.sol	0.00% (**0/**71)	0.00% (**0/**82)	0.00% (**0/**14)	0.00% (**0/**26)
test/TokenLocker.**sol/**Setup.t.sol	9.09% (**2/**22)	8.70% (**2/**23)	100.00% (**0/**0)	16.67% (**1/**6)
test/TokenLocker.**sol/invariant/**EarlyTermination.sol	0.00% (**0/**6)	0.00% (**0/**6)	0.00% (**0/**2)	0.00% (**0/**3)
test/TokenLocker.**sol/invariant/**Locker.Invariant.t.sol	0.00% (**0/**40)	0.00% (**0/**54)	0.00% (**0/**22)	0.00% (**0/**5)
test/TokenLocker.**sol/invariant/**LockerNonUpgradeable.sol	0.00% (**0/**125)	0.00% (**0/**145)	0.00% (**0/**44)	0.00% (**0/**28)
test/TokenLocker.**sol/invariant/**Migrator.sol	0.00% (**0/**5)	0.00% (**0/**5)	100.00% (**0/**0)	0.00% (**0/**3)
test/UpgradeDeployer.sol	0.00% (**0/**69)	0.00% (**0/**91)	100.00% (**0/**0)	0.00% (**0/**15)
test/Upgrades.t.sol	33.33% (**1/**3)	33.33% (**1/**3)	100.00% (**0/**0)	33.33% (**1/**3)
test/Upgradoor.**sol/**Setup.sol	0.00% (**0/**28)	0.00% (**0/**30)	0.00% (**0/**4)	0.00% (**0/**2)
test/Upgradoor.**sol/integration/**Setup.t.sol	0.00% (**0/**98)	0.00% (**0/**121)	0.00% (**0/**18)	0.00% (**0/**10)
test/Upgradoor.**sol/sharesTimelock/**Migrate.t.sol	100.00% (**2/**2)	100.00% (**2/**2)	100.00% (**0/**0)	100.00% (**2/**2)
test/mocks/MockMigrator.sol	100.00% (**1/**1)	100.00% (**1/**1)	100.00% (**0/**0)	100.00% (**1/**1)
test/mocks/SharesTimeLockMock.sol	88.89% (**16/**18)	89.47% (**17/**19)	50.00% (**5/**10)	60.00% (**3/**5)

File	% Lines	% Statements	% Branches	% Funcs
test/mocks/Token.sol	100.00% (**4/**4)	100.00% (**4/**4)	100.00% (**0/**0)	100.00% (**2/**2)
test/rewards/MerkleDistributorReentrant.t.sol	100.00% (**4/**4)	100.00% (**4/**4)	100.00% (**0/**0)	100.00% (**4/**4)
test/rewards/MerkleTreeInitializer.sol	2.04% (**1/**49)	1.96% (**1/**51)	100.00% (**0/**0)	11.11% (**1/**9)
test/rewards/MerkleTreeUpgrades.t.sol	50.00% (**1/**2)	50.00% (**1/**2)	100.00% (**0/**0)	50.00% (**1/**2)
test/utils.sol	100.00% (**15/**15)	95.00% (**19/**20)	50.00% (**1/**2)	75.00% (**3/**4)
Total	31.27% (**632/**2021)	31.00% (**752/**2426)	24.36% (**191/**784)	43.10% (**181/**420)

Changelog

- 2023-03-31 - Initial report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no

responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.