Quantstamp

Auxo Governance

Executive Summary

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

Timeline	2023-02-08 through 2023-02-15
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Documentation
Source Code	• Alexintosh/auxo-governance #0deb98e • Alexintosh/auxo-governance #2b9c6ef • Alexintosh/auxo-governance #4031436
Auditors	• Mostafa Yassin Auditing Engineer • Ed Zulkoski Senior Auditing Engineer • Cameron Biniamow Auditing Engineer • Ruben Koch Auditing Engineer

Documentation quality	High
Test quality	High
Total Findings	19 Fixed: 10 Acknowledged: 8 Mitigated: 1
High severity findings ⓘ	2 Fixed: 2
Medium severity findings ⓘ	2 Fixed: 2
Low severity findings ⓘ	7 Fixed: 3 Acknowledged: 3 Mitigated: 1
Undetermined severity findings ⓘ	1 Fixed: 1
Informational findings ⓘ	7 Fixed: 2 Acknowledged: 5

Summary of Findings

Auxo Governance protocol allows the staking of `AUXO` tokens in order to gain voting power in the form of `veAUXO`. The amount of `veAUXO` depends on the amount of `AUXO` staked and the duration of the lock, which has a minimum duration of 6 months.

It is also possible to deposit `AUXO` tokens permanently and gain the `xAUXO` which is a liquid staking derivative.

Initial audit:
We have raised 19 issues, ranging from high to undetermined severity. We recommended fixing these issues before deployment.

Update after the first re-audit:
Client addressed all of the findings by either fixing or acknowledging them, however the fix for `QSP 4 – Voting Multiplier Can Be Gamed Using increaseAmountFor` caused two new vulnerabilities.

Update after the second re-audit:
Client provided a fix for `QSP 4 – Voting Multiplier Can Be Gamed Using increaseAmountFor`.

Also, in this re-audit round, the following file names were changed:

- `veAUXO.sol` to `ARV.sol`
- `xAUXO.sol` to `PRV.sol`

All related variable names were also changed.

ID	DESCRIPTION	SEVERITY	STATUS
QS-1	**Incorrect Burn Operation in `eject()` Function**	● High ⓘ	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
QS-2	Funds May Be Stolen if Two Merkle Windows Use Different Tokens	High	Fixed
QS-3	Reentrancy During Sending Rewards	Medium	Fixed
QS-4	Voting Multiplier Can Be Gamed Using `increaseAmountFor`	Medium	Fixed
QS-5	Privileged Roles and Ownership	Low	Acknowledged
QS-6	Missing Input Validation	Low	Mitigated
QS-7	Ownership and Roles Can Be Renounced/Revoked	Low	Acknowledged
QS-8	Events Not Emitted	Low	Fixed
QS-9	Unintuitive User-Experience when `feeBeneficiary` Is Not Set	Low	Fixed
QS-10	Eject Functionality May Harm Interacting Contracts	Low	Acknowledged
QS-11	Incorrect Semantics for `Bitfield.activateFrom()`	Low	Fixed
QS-12	Unclear Economic Incentive for `xAUXO` Liquid Staking	Informational	Acknowledged
QS-13	Only the Rollstaker Admin Can Activate the Next Epoch	Informational	Acknowledged
QS-14	The `StakingManager` Should Not Be Ejectable	Informational	Acknowledged
QS-15	Unnecessary `COMPOUNDER_ROLE`	Informational	Acknowledged
QS-16	Funds Locked by `StakingManager` Are Not 1:1 Pegged to `xAuxo`	Informational	Acknowledged
QS-17	Unresolved `TODOs` in Code	Informational	Fixed
QS-18	Quorum Denominator Cannot Be Adjusted Later	Informational	Fixed
QS-19	Potential "Snowball Effect" Regarding the `StakingManager.delegateto()` Function	Undetermined	Fixed

Assessment Breakdown

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

Disclaimer

If the final commit hash provided by the client contains features that are not within the scope of the audit or an associated fix review, those features are excluded from consideration in this report.

This audit is only concerned with the following files:

- src/AUXO.sol
- src/veAUXO.sol
- src/modules/governance/EarlyTermination.sol
- src/modules/governance/Governor.sol
- src/modules/governance/IncentiveCurve.sol

- src/modules/governance/Migrator.sol
- src/modules/governance/TokenLocker.sol
- src/modules/LSD/bitfield.sol
- src/modules/LSD/RollStaker.sol
- src/modules/LSD/StakingManager.sol
- src/modules/LSD/xAUXO.sol
- src/modules/rewards/MerkleDistributor.sol

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

QS-1 Incorrect Burn Operation in `eject()` Function

● High ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `6daf6a0a70faef0208ca3aa3e16414d6bbee14a7`.

Recommendation implemented.

File(s) affected: `TokenLocker.sol`

Description: The function `eject()` iterates through a list of accounts removing a given account's stake if its lock has expired. This involves burning each account's `veTokens` and transferring them their corresponding depositTokens. However, the loop instead burns from `msg.sender`:

```
veToken.burn(account, veToken.balanceOf(_msgSender()));
```

This has the effect of incorrectly clearing the `veToken` balance of the caller, thus removing their voting power, while also not removing the voting power for any of the ejected accounts. As long as these ejected accounts do not create a new lock, they will permanently retain their voting power while having zero stakes.

Recommendation: Change the above line to:

```
veToken.burn(account, veToken.balanceOf(account));
```

QS-2 Funds May Be Stolen if Two Merkle Windows Use Different Tokens

● High ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `29182b640a367be84f865374d8acceac405c071b`.

Recommendation implemented, the token is now included in the Merkle verification.

Description: The `MerkleDistributor` contract does not restrict which tokens can be used as rewards, and different windows can use different rewards tokens. However, `verifyClaim()` suggests that the leaves of the Merkle tree are hashes of the data `<account,accountIndex,windowIndex,amount>`. Importantly, note that the `Claim.token` is not included in this data. This allows the user to choose which token to be rewarded in, which may not be the same as intended by the window.

Exploit Scenario: Suppose there are two reward windows:

1. Window 1 has in total 1,000 DAI. Alice has a reward in this window for 10 DAI (equivalent to $10 USD).
2. Window 2 has in total 100 WETH. Alice has zero rewards in this window.

Alice now invokes `claim()` with the following data:

```
 {
     windowIndex = 1; // Window 1
     accountIndex = "Alice's account index in the Window 1 Merkle tree";
     amount = 10e18; // multiplied by 1e18 due to decimals
     token = WETH; // exploit occurs here
     merkleProof = "proof traversing the Window 1 Merkle tree";
     account = "Alice's address";
}
```

The result is that Alice is rewarded 10 WETH instead of 10 DAI.

Recommendation: Include the intended token address in the Merkle tree data.

QS-3 Reentrancy During Sending Rewards

● Medium ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `7cc81d7d4a68fcc8478fe7c40fad27ec319e35df`.

Recommendation implemented.

File(s) affected: `MerkleDistributor.sol`

Description: The `claim()` & `claimMulti()` functions accept a `_claim` object and verify that it is valid through Merkle Tree. However, the `_claim.rewardToken` parameter is not included in the Merkle Tree verification.

This means that a malicious user could use a valid `_claim` object and pass a `rewardToken` address that he controls.

When the `_processClaim()` functions call `safeTransfer()` the attacker's malicious code will be able to re-enter into the contract again in the same transaction.

Recommendation: Use a re-entrancy guard, and include the `rewardToken` in the Merkle Tree verification

QS-4 Voting Multiplier Can Be Gamed Using `increaseAmountFor`

● Medium ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `1152d2927d896e00974330d071930f10bd6f2c85`.

This fix introduced two potential Issues:

1 - There is now a griefing attack vector where a user can deposit a tiny amount of new tokens for a different user, resetting their timer.

2 - Depositing tokens can cause inaccuracies because of how solidity rounds integers down. This can cause the require statement:

`require(veToken.balanceOf(_msgSender()) == veShares)` in `boostToMax()`

to fail because the token balance will not be equal to shares due to rounding errors.

For the first issue, it is recommended to remove the function `increaseAmountFor()`.

For the second issue, it is recommended to remove the `require` statement in `boostToMax()`.

Update after the second re-audit: Recommendation implemented.

File(s) affected: `TokenLocker.sol`

Description: The function `_increaseAmountFor()` allows a user to increase their staked amount without changing their `lockDuration`. However, the lock multiplier associated with the new deposit is the same as the original deposit. If a user deposits new tokens near the end of their staking period, they may have a large voting multiplier without a long stake on their tokens.

Exploit Scenario:

1. Suppose at month 0, the user deposits the minimal amount of tokens for the maximal 36-month period (thus having the largest lock multiplier). For simplicity in our example, assume `minLockAmount == 1`.
2. At month 35, 1 month before their unlock, an important vote is about to occur that they wish to manipulate. The user then invokes `increaseAmount("1 billion tokens")`. The multiplier on the one billion tokens is still maximal, even though they will be able to withdraw in 1 month instead of 36 months.
3. The user votes on the topic.
4. The user withdraws all tokens 1 month later.

Exploit scenario after the commit `1152d2` (accounting inaccuracies):

1. Suppose the lock duration is 6 months, so the lock multiplier is 83333333333300000, i.e., 0.083 ∗ 10^18.
2. Suppose our initial deposit is 15 tokens (not 15e18, just 15). This will result in the user being awarded 15 ∗ 83333333333300000 / 1e18 =~= 1.25 → `1 veToken` (due to integer arithmetic truncation).
3. Now suppose we increase the amount by 23 new tokens. In `_increaseAmountFor()`, the newly awarded amount is 23 ∗ 83333333333300000 / 1e18 =~= 1.91 → `1 veToken`.

In total, the user will receive `2 veTokens`, but will have deposited 38 depositTokens.

4. Now the require check will fail because 38 ∗ 83333333333300000 / 1e18 =~= 3.16 → `3 veTokens`.

Recommendation: Consider the implications of users increasing their staked amounts late in their lock period and revise as needed. It is generally recommended to reset the lock duration.

QS-5 Privileged Roles and Ownership

• Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.

The client provided the following explanation:

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:

- EarlyTermination.sol
- TokenLocker.sol
- MerkleDistributor.sol
- RollStaker.sol
- AUXO.sol
- xAUXO.sol
- StakingManager.sol.

File(s) affected: `EarlyTermination.sol`, `TokenLocker.sol`, `MerkleDistributor.sol`, `RollStaker.sol`, `AUXO.sol`, `xAUXO.sol`, `StakingContract.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.

For instance, the owner of the `RollStaker.sol` can withdraw all the staked amount through a call to `emergencyWithdraw()`. The owner can also pause rewards withdrawing/depositing.

The following are more examples:

1. `EarlyTermination.setPenalty()` allows the owner to set the early withdraw penalty percentage (up to 100%).
2. `TokenLocker.setWhitelisted()` allows the owner to whitelist arbitrary contracts that can interact with the system.
3. `TokenLocker.setxAUXO()` allows the owner to change the xAUXO address.
4. The owner of `MerkleDistributor` can lock out rewards indefinitely.
5. The admin of `RollStaker` can withdraw all `stakingTokens` in the contract at any time.

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.

QS-6 Missing Input Validation

● Low Mitigated

Update

Marked as "Mitigated" by the client. Addressed in: `6c2cd78ac298f52516c64d0d48e7db706d0cb12b`.

The client provided the following explanation:

Quantstamp correctly notes that some variables do not have validations for null checks or known erroneous values. We subdivide these errors into:

• Initialization validation: missing validations during contract deployment and/or initialization

• Runtime validation: missing validations for values that can be changed over the contract lifetime

• For initialization validation, we acknowledge the risks, instead opting for runtime, pre and post deploy health checks to ensure all variables are set as expected. These health checks are in the form of foundry scripts, although they have not themselves been subject to a formal audit. The scripts cover the following issues raised:

• StakingManager.sol: all values

• Governor.sol: all values

• RollStaker.sol: all values

• TokenLocker: min duration, max duration, min lock amount

• For runtime validation, here are the specific comments:

• Migrator.sol.setMigrationEnabled: covered by deploy scripts mentioned above

• TokenLocker.sol.setXAuxo: covered by deploy scripts and not expected to change

• xAuxo.sol.setEntryFee/constructor: risk of setting fee without beneficiary is bourne by operator

• EarlyTermination.sol.setPenaltyBeneficiary: risk of setting incorrect address is bourne by operator

• MerkleDistribution.sol.setLock: risk of setting incorrect block number is bourne by operator

• The following 3 cases were flagged for missing validation but have reverts further down the call stack:

• TokenLocker.sol.depositByMonths: veAUXO reverts on mint to Zero Address

• TokenLocker.sol: getDuration(months) is called in depositByMonts, and increaseByMonths. Both these functions validate the passed months inside getLockMultiplier.

• xAUXO._depositAndStake: _account will revert if minting to the zero address as it is OpenZeppelin

File(s) affected: `StakingManager.sol`, `EarlyTermiantion.sol`, `Migrator.sol`, `Governor.sol`, `xAUXO.sol`, `MerkleDistributor.sol`, `RollStaker.sol`

Description: The following inputs need to be checked

• In `StakingManager.sol`, check `initialize._auxo` against the `0x0` address.

• In `StakingManager.sol`, check `initialize._veAuxo` against the `0x0` address.

• In `StakingManager.sol`, check `initialize.governor` against the `0x0` address.

• In `StakingManager.sol`, check `initialize.tokenLocker` against the `0x0` address.

• In `EarlyTermination.sol`, check `setPenaltyBeneficiary.penaltyBeneficiary` against the `0x0` address.

• In `Migrator.sol`, check `setMigrator._migrator` against the `0x0` address.

• In `Migrator.sol`, check the `migrator` against the `0x0` address in `setMigrationEnabled()`.

• In `Governor.sol`, `constructor` is missing validation for all inputs.

• In `TokenLocker.sol`, `initialize._minLockAmount` should be > 0.

• In `TokenLocker.sol`, `initialize._maxLockDuration` should be <= `getDuration(maxRatioArray.length)`.

• In `TokenLocker.sol`, `initialize.minLockDuration` should be >= `getDuration(6)`, or else the multiplier results will revert.

• In `TokenLocker.sol`, `depositByMonths()` should check that `_receiver` is not the `0x0` address; and `getDuration(_months)` is >= `minLockDuration`.

• In `TokenLocker.sol`, check `setxAUXO._xAUXO` against the `0x0` address.

• In `xAUXO.sol`, `constructor()` should check that `entryFee` and `feeBeneficiary` are only settable if `_entryFees` is not zero and `_feeBeneficiary` is not the `0x0` address.

• In `xAUXO.sol`, `setEntryFee()` should revert if the `feeBeneficiary` is the `0x0` address. Or else, fees can potentially be sent off to the `0x0` address. Additionally, `setFeePolicy()` would then need to swap the function calls.

- In `xAUXO.sol`, check `_depositAndStake._account` against the `0x0` address.
- In `MerkleDistributor.sol`, `setLock()` should check that `_lock` is greater than or equal to `block.number`.
- In `RollStaker.sol`, check `constructor.stakingToken` against the `0x0` address.

Recommendation: Consider adding the suggested missing input validations.

QS-7 Ownership and Roles Can Be Renounced/Revoked

• Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.

The client provided the following explanation:

In line with Quantstamp recommendations, we acknowledge that the following contracts have access controls that can be renounced or revoked, potentially leaving some functions unable to be executed:

- EarlyTermination.sol
- TokenLocker.sol
- MerkleDistributor.sol
- RollStaker.sol
- xAUXO.sol
- StakingManager.sol
- Migrator.sol
- AUXO.sol

File(s) affected: `EarlyTermination.sol`, `TokenLocker.sol`, `MerkleDistributor.sol`, `RollStaker.sol`, `xAUXO.sol`, `StakingContract.sol`, `Migrator.sol`, `AUXO.sol`

Description: If the owner renounces their ownership, all ownable contracts will be left without an owner. Consequently, any function guarded by the `onlyOwner` modifier will no longer be able to be executed.

The contracts also use the `AccessControl` library, which allows the function `revokeRole()` to remove a given role. In the case of the `DEFAULT_ADMIN_ROLE` or `MINTER_ROLE` getting revoked, this will prevent the protocol from functioning as intended.

Recommendation: Double-check if this is the intended behavior.

If not, consider overriding the `renounceOwnership()` for the `Ownable` contract and the `revokeRole()` for the `AccessControl` contract to always revert. Or, in the case of `revokeRole()`, at least require the existence of one role, of the type being revoked, after revoking.

QS-8 Events Not Emitted

• Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `de72dd6b7e8030b03b6f00286c8929d313ad0cde`.

Recommendation implemented.

File(s) affected: `xAUXO.sol`, `TokenLocker.sol`

Description: During deployment, `xAUXO` sets `entryFee` and `feeBeneficiary`, but the events `EntryFeeSet` and `FeeBeneficiarySet` are not emitted. Similarly in `TokenLocker.initialize()`, `minLockAmount` and `ejectBuffer` are set, but the events `MinLockAmountChanged` and `EjectBufferUpdated` are not emitted.

Exploit Scenario: Emit the respective events after the aforementioned contract-level state variables are assigned.

QS-9 Unintuitive User-Experience when `feeBeneficiary` Is Not Set

• Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `f648eb0550afa68901f42918719cb88140223b23`.

Recommendation implemented.

File(s) affected: `xAUXO.sol`

Description: In `_chargeFee()`, the fee is first calculated and only sent to the `feeBeneficiary` if its address is non-zero. However, when the `feeBeneficiary == address(0)`, the `amountMinFee` will be equal to `_amount - feeAmount`, but the `feeAmount` will not be transferred from the user's account; resulting in the user depositing less than what was intended.

For example, suppose the entry fee is 10%, the `feeBeneficiary` is not set, and the user invokes `depositFor()` with `_amount = 1e18.` This will result in `9e17` tokens being deposited, with `1e17` tokens remaining in their account. The user would be unable to deposit their whole balance of `AUXO` into the contract without many calls to the contract (e.g., after one call `1e17` tokens would remain in their wallet, then `1e16`, then `1e15`, ...).

Recommendation: If the `feeBeneficiary` is equal to the `0x0` address, return `amount` in the `_chargeFee()` function. This will allow the user to deposit the amount intended.

QS-10 Eject Functionality May Harm Interacting Contracts

• Low ⓘ Acknowledged

Update
Marked as "Fixed" by the client. Addressed in: `368371f68489ad6d7c2835bcaac89134bf7ec559`.

The client provided the following explanation:

Quantstamp notes that the TokenLocker has an eject function that allows anyone to force a user to exit their stake, after a grace period. It is argued that this behaviour may be unexpected for veAUXO holders that are also smart contracts.

Acknowledging this, we have added a note to the ITokenLocker interface and to the veAUXO contract, to remind developers. Furthermore, smart contracts must be whitelisted before they can deposit into the TokenLocker, so it is expected that the AUXO team will have a reasonable chance of conveying this behaviour during development.

File(s) affected: `TokenLocker.sol`

Description: The `eject()` function allows any user to remove another user's stake if the eject buffer period has passed. If the user being ejected is a smart contract, then its functionality may be dependent on invoking `withdraw()` itself. For example, suppose the contract implements a function of the following form:

```
 function withdrawAuxoAndRepayBeneficiary() {
     tokenLocker.withdraw();
     auxoToken.safeTransfer(beneficiary, auxoToken.balanceOf(address(this)));
}
```

If a different user invokes `eject()` first, this function would fail due to `withdraw()` reverting (since the lock would no longer exist).

Recommendation: Ensure that whitelisted contracts that interact with the `TokenLocker` contract are aware of ejection scenarios.

QS-11 Incorrect Semantics for `Bitfield.activateFrom()`

• Low ⓘ Fixed

Update
Marked as "Fixed" by the client. Addressed in: `5e6c5e40466b151d96b6ab1f2301800dda0cf7d2`.

Recommendation implemented.

File(s) affected: `Bitfield.sol`

Description: The function `activateFrom()` "takes an existing bitfield, and sets all values starting at _epochFrom to one". However, since the function uses `self._value ^= activator` (using XOR), if there are already existing bits that are set to `1` beyond index `_epochFrom`, they will instead be zeroed.

While this may not be problematic for the current usage in `RollStaker.sol`, if future projects rely upon the library, there may be unforeseen consequences.

Recommendation: Either revise the function name and documentation or use the OR (|) operator instead of XOR.

QS-12
Unclear Economic Incentive for `xAUXO` Liquid Staking

• Informational ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client.

The client provided the following explanation:

QSP-3 was raised as high severity due to the fact that the xAUXO does not have a withdraw mechanism, and, consequently, there is no guarantee of price parity between AUXO. The comparison was made between staking derivatives between ETH and stETH, where the price is dictated by the ability of the token to be redeemed for ETH at some later date.

xAUXO is easier to compare with other staking derivatives, with significant TVL and application across DeFi, that utilise one-way staking derivatives and no guarantee of price pegging. Examples of such protocols include veCRV/cvxCRV (Curve and ConvexCurve), and Balancer/AURA finance.

The Auxo team feel that, because such protocols have been successful without offering a price peg, this particular issue is incorrectly flagged as high severity when it should be considered 'informational', and communicated clearly to users. Additionally, the DAO has plans to allocate resources to xAUXO buybacks, which, while still to be confirmed, are aiming to give xAUXO holders opportunities to exit their positions.

File(s) affected: `xAUXO.sol` , `StakingManager.sol`

Description: Typically, liquid staking protocols (e.g., Lido or Coinbase's cbETH) propose liquid staking so that if users do not wish to actively participate in the protocol (e.g., by running an ETH2 validator node), they can still earn a portion of the staking rewards (effectively by delegating stake). In these two example systems, while users hold the Lido `stETH` tokens or Coinbase `cbETH` tokens, they will accrue rewards, which are some fraction of the rewards earned by the underlying validator nodes. The prices of cbETH and stETH typically follow the same trend as ETH (with some deviation due to rewards or general market fluctuations).

However, a key reason for this price parity between `cbETH<>ETH` and `stETH<>ETH` is that **eventually, it will be possible to withdraw ETH from the validator network**. Suppose this were not the case, i.e., users deposited ETH with no chance of ever recovering their tokens. If, for example, the user deposited 1 ETH valued at $2000 at time of stake, then they would need to expect at least $2000 return in rewards, otherwise there would be no economic incentive to stake in the first place.

This possibility of withdrawal does not appear to exist in `xAUXO` , and a comment explicitly states "tokens are [locked in] stakingManager in perpetuity, no coming back". As such, there does not appear to be any reason for price parity between `AUXO<>xAUXO` , making the utility and incentive of the `xAUXO` system unclear.

Recommendation: Revise the economic incentive system behind `xAUXO` .

QS-13 Only the Rollstaker Admin Can Activate the Next Epoch

● Informational ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client.

The client provided the following explanation:

Quantstamp notes that the RollStaker does not automatically activate the next epoch nor does it have any sort of time range for how long an epoch is. Further, only the admin of the RollStaker can activate the next epoch.

We confirm that it is intended behaviour not to encode specific epoch timings into the RollStaker.

This was a deliberate design decision made to align the RollStaker's concept of epochs with existing processes for distributing rewards in the MerkleDistributor.

File(s) affected: `RollStaker.sol` ,

Description: RollStaker does not automatically activate the next epoch nor does it have any sort of time range for how long an epoch is. Further, only the admin of `RollStaker` can activate the next epoch. Per inline comments in `RollStaker` , the duration of an epoch is one month.

If the `RollStaker` admin fails to call `RollStaker.activateNextEpoch()` , the next epoch will never be activated. Alternatively, the `RollStaker` admin could activate the next epoch at any time they want since there is no minimum epoch time.

Recommendation: Verify that this is intended behavior and inform users that the `RollStaker` admin has complete control of if and when the next epoch is activated.

QS-14 The `StakingManager` Should Not Be Ejectable

● Informational ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client.

The client provided the following explanation:

Quantstamp recommends that the StakingManager should be blacklisted from ejection from the TokenLocker, noting:

The StakingManager contract holds the lock of AUXO in the TokenLocker on behalf of the xAUXO contract, the liquid staking version of the AUXO token. Given that the token is intended to be an irreversible conversion of the token, this contract should not be ejectable, as else the depositors would hold both the token and the token from their deposit.

Quantstamp also notes that the option to renew the stakingManager's lock is available to any user, and that, therefore it is a "very unlikely scenario that the lock will run out unnoticed" over the course of 36 months.

We therefore acknowledge this minor risk of forgetting to re-boost the staking manager, additionally we note that the staking manager should potentially be ejectable if it is decided to decommission the manager - a code upgrade would be needed here to remove the public boostToMax function.

File(s) affected: `TokenLocker.sol`

Description: The `StakingManager` contract holds the lock of `AUXO` in the `TokenLocker` on behalf of the `xAUXO` contract, the liquid staking version of the `AUXO` token.

Given that the `xAUXO` token is intended to be an irreversible conversion of the `AUXO` token, this contract should not be ejectable, as else the depositors would hold both the `xAUXO` token and the `AUXO` token from their deposit.

The developers thought of this and created an external function that is callable by anyone for the `StakingManager` contract that calls the `boostToMax()` function for it in the `TokenLocker` contract, which renews the lock to the maximum duration.

While it is a very unlikely scenario that the lock will run out unnoticed, we still recommend properly blacklisting the `StakingManager` contract from the ejecting mechanism.

Recommendation: Add a check in `TokenLocker.eject()` that an account is not equal to the `StakingManager` contract address.

QS-15 Unnecessary `COMPOUNDER_ROLE`

● Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.

The client provided the following explanation:

Quantstamp notes that the TokenLocker contract has a COMPOUNDER_ROLE, which is used only to restrict increaseAmountsForMany to the compounder, but asks why the role exists, given that same functionality can be achieved with repeated calls to increaseAmountFor.

increaseAmountsForMany was envisioned to be a utility method used by a to-be-developed compounding vault for ARV/veAUXO. Initially, we saw no harm in making it public and adding the same modifier functionality inside the for-loop.

However, static analysis highlighted that there is a potential reentrancy attack due to state changes in between looped calls to veToken.mint() and the final depositToken.safeTransferFrom after the loops had completed. If, somehow, an attacker is able manipulate the control flow before the final safeTransferFrom is called, then they could have additional reward tokens without having to pay the deposit tokens.

While we couldn't define a specific exploit scenario, we decided to make the function permissioned as a precautionary measure, especially as we see a low likelihood of regular users needing it.

File(s) affected: `TokenLocker.sol`

Description: The role `COMPOUNDER_ROLE` in the `TokenLocker` contract seems to be unnecessary. It only guards access to the `increaseAmountsForMany()` function, whose functionality can also be achieved without access to the role by repeatedly calling `increaseAmountFor()`, as it is a completely unrestricted method.

Exploit Scenario: Consider removing the `COMPOUNDER_ROLE` or adding the modifier for the `increaseAmountsForMany()` function if that was forgotten.

QS-16 Funds Locked by `StakingManager` Are Not 1:1 Pegged to `xAuxo`

● Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.

The client provided the following explanation:

Quantstamp acknowledges that the AUXO balance of the Staking Manager is not a reliable indicator of the xAUXO balance of the Staking Manager, and that the Staking Manager's AUXO balance may stray from the total xAUXO in the system due to the fact that the Staking Manager may have received AUXO from ERC20 transfers.

As recommended we formally acknowledge this risk. A comment has been added to the staking manager to make it clear.

File(s) affected: `StakingManager.sol`

Description: The `StakingManager` contract deposits its whole `AUXO` balance into the `TokenLocker` contract. While it is expected that that token balance only comes from transfers from the `xAUXO` contract, anyone could send `AUXO` tokens to the `StakingManager` contract which would get locked away.

Such direct `AUXO` transfers would become inaccessible to the sender and no `xAUXO` tokens would be minted for them. Therefore, there is no 1:1 peg between `veAUXO` of the `StakingManager` and the StakingManager's locked `AUXO` multiplied by the corresponding multiplier.

This could only result in problems if some part of the system relies on such a 1:1 peg in an invariant, which is currently not the case. This issue is simply intended to raise awareness for the developers.

Recommendation: There is no course of action, the developers should be aware of the potential pitfall in case an assumption is made about the `xAUXO` token in circulation by looking at the `veAUXO` owned by the `StakingManager` contract.

QS-17 Unresolved `TODOs` in Code

● Informational ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `6597ff095fd4f4f253cc37f49d34b16e3578edbd`.

The client provided the following explanation: Unnecessary TODOs removed.

File(s) affected: `bitfield.sol`, `PolicyManager.sol`

Description: The following `TODOs` annotations still remain in the code:
1. In `Bitfield.sol`: "TODO: experiment with a 'pure' version of the library for (possible) gas savings"
2. In `PolicyManager.sol`: "TODO Check for duplicated in queue"

Recommendation: Resolve all `TODOs` before using the code in production.

QS-18 Quorum Denominator Cannot Be Adjusted Later

● Informational ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `032bbfa6a82b42a0a99533376fff63d1ac28d807`.

The client provided the following explanation:

Quantstamp correctly identified a misunderstanding on the contract author's part: that OpenZeppelin governance's quorum denominator is not adjustable.

Having re-reviewed the documentation, we are happy to leave the default behaviour in place, with the deonomicator set to 100 - we have adjusted the comment to reflect this.

File(s) affected: `Governor.sol`

Description: The `constructor()` comment mentions that the "quorum percentage is initially set as x/100, the denominator can be adjusted later". However, the denominator can only be adjusted from 100 if the function `GovernorVotesQuorumFraction.quorumDenominator()` is overridden, as discussed here.

Recommendation: If the denominator does not need to change from 100, consider updating the comment. Otherwise, override the `quorumDenominator()` function and add a setter function for a denominator variable.

QS-19 Potential "Snowball Effect" Regarding the `StakingManager.delegateto()` Function

● Undetermined ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `a72cd091b3e10e2a36319555bdcabf0c43a051ec`.

It is recommended to modify the `AUDITORS-READ-ME.md` file to inform future auditors of this change.

File(s) affected: `StakingManager.sol`

Description: The function `StakingManager.delegateTo()` allows governance to vote on which representative will receive the voting power of the `xAUXO` users. Suppose that `xAUXO` accounts for a significant fraction of the voting power. Then the user that has been delegated power can continually vote for themselves, and the remaining voters may not have enough power to renounce that delegation.

Recommendation: Ensure that the amount staked in `xAUXO` is not enough to irreversibly over-allocate voting power.

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).

Adherence to Best Practices

1. The event `ITokenLockerEvents.IncreasedLock()` is not used anywhere. It should likely be emitted by `_increaseUnlockDuration()`. **Update:** Fixed in `4031436`
2. The function `Migrator.migrate()` is expected to be overridden, but includes an implementation (simply emitting an event). It may be better to not include the implementation (i.e., change to `function migrate(address Staker) external virtual;`), ensuring that classes that extend Migrator must actually override the function.
3. In the function `TokenLocker.isLockExpired()`, the comment mentions "// upcasting is safer than downcasting", however in `canEject()`, the opposite approach occurs by downcasting `block.timestamp.toUint32()`. **Update:** Fixed in `4031436`
4. In `TokenLocker.getLockMultiplier()`, the error message "GML: Duration incorrect" should instead be "GLM: Duration incorrect". **Update:** Fixed in `4031436`
5. In `veAUXO.burn()`, the parameter `to` should instead be named `from`. **Update:** Fixed in `4031436`
6. For the function `StakingManager.transferGovernance()`, `grantRole()` should be used instead of `_setupRole()`. **Update:** Fixed in `4031436`
7. In `RollStaker.sol`, the errors `Deprecated()` and `DepositReverted()` are not used and could be removed. **Update:** Fixed in `4031436`
8. The current configuration of `TokenLocker`, `xAUXO`, and `StakingManager` results in unnecessary transferring of `AUXO` when `terminateEarly()` is called. Consider refactoring the contracts to avoid excessive gas costs.
9. Functions marked as `public` that are not accessed within the contract can be marked as `external`, e.g. `TokenLocker.increaseAmount()` (or alternatively use it in `TokenLocker.sol#L290`). **Update:** Fixed in `4031436`
10. For-loops can be gas-optimized by caching the `array.length` in a memory variable.
11. Generally, it is more gas-efficient to increment via `++i` instead of `i++`. If it is in fixed bounds, i.e. smaller than some array length, an unchecked block can be put around it `(unchecked {++i;})` for further gas cost improvements.
12. Contract names should match the file name. That is not the case for `EarlyTermination.sol` and `Migration.sol`, `Governor.sol`, `Bitfield.sol`.
13. Since the features that the `AccessControl` import provides are a superset of the ones from `Ownable`, consider removing the redundant `Ownable` import in `TokenLocker.sol` **Update:** Fixed in `205d6c215`.
14. `TokenLocker._increaseUnlockDuration()` is slightly misnamed, we would recommend changing it to `_increaseLockDuration()`.
15. Max-approvals are being used in the `StakingManager` contract, which is a pattern we generally discourage. Consider setting the allowance appropriately equal to the contract's balance before desired transfers instead.
16. `IERC20Permit` is no longer in the draft, which is why OpenZeppelin provides the file also without the draft prefix in newer versions. Consider updating the import path to "@oz/token/ERC20/extensions/IERC20Permit.sol". **Update:** Fixed in `12d5d63d`
17. `RollStaker.sol#L261` can be `else if` instead of `if`. **Update:** Fixed in `7f974a64d2`
18. When dealing with `unsigned integer types (uint)`, comparisons with `!= 0` are more gas efficient than with > 0.
19. `StakingManager.approveAuxo()` does not have any access control and should therefore not be listed in the section of admin functions. **Update:** Fixed in `7f974a64`

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

- `c3f...12d ./src/AUXO.sol`
- `a99...4d9 ./src/veAUXO.sol`
- `6c4...9c1 ./src/modules/rewards/MerkleDistributor.sol`
- `31a...79a ./src/modules/LSD/bitfield.sol`
- `39c...4ee ./src/modules/LSD/RollStaker.sol`
- `98a...ec0 ./src/modules/LSD/StakingManager.sol`
- `c78...c17 ./src/modules/LSD/xAUXO.sol`
- `dd9...d21 ./src/modules/governance/EarlyTermination.sol`
- `234...e2a ./src/modules/governance/Governor.sol`
- `f25...846 ./src/modules/governance/IncentiveCurve.sol`
- `237...fca ./src/modules/governance/Migrator.sol`
- `e25...447 ./src/modules/governance/TokenLocker.sol`
- `cc2...52f ./src/interfaces/IERC20MintableBurnable.sol`
- `976...dfa ./src/interfaces/ILiquidStakingDerivative.sol`
- `df9...1f6 ./src/interfaces/ILSD.sol`
- `bf5...c1a ./src/interfaces/IRollStaker.sol`
- `8ee...2c4 ./src/interfaces/IStakingManager.sol`
- `e8a...6f2 ./src/interfaces/ITokenLocker.sol`

Toolset

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

Setup

Tool Setup:
- Slither v0.8.3

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

Automated Analysis

Slither

Slither found nothing of severity

Test Suite Results

Run `make test-unit`

```
 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 27.23ms

Running 3 tests for test/Upgradoor.sol/Getters.t.sol:TestGetters
[PASS] testGetAmountAndLongestDuration() (gas: 239142)
[PASS] testGetMonthsNewLock(uint32) (runs: 256, μ: 12589, ~: 12608)
[PASS] testGetOldLock() (gas: 47871)
Test result: ok. 3 passed; 0 failed; finished in 32.47ms

Running 1 test for test/Auxo.sol/MintBurn.t.sol:TestMintBurn
[PASS] testFuzz_RestrictedMint(address,address,uint256) (runs: 256, μ: 132618, ~: 134329)
Test result: ok. 1 passed; 0 failed; finished in 232.05ms

Running 1 test for test/Governor/GovSetup.t.sol:GovSetupTest
[PASS] testSettingControllerRoles(uint256,address,address) (runs: 256, μ: 2070075, ~: 2070386)
Test result: ok. 1 passed; 0 failed; finished in 445.40ms
```

```
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 479.73ms

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: 116463, ~: 117425)
Test result: ok. 2 passed; 0 failed; finished in 283.79ms

Running 2 tests for test/TokenLocker.sol/Boost.t.sol:TestlockerBoost
[PASS] testFuzz_BoostToMax(address,uint128,uint8) (runs: 256, μ: 183776, ~: 184062)
[PASS] testSmallQtyIncreaseDoesNotBrickBoost(address,uint128,uint128,uint8) (runs: 256, μ: 227209, ~:
227288)
Test result: ok. 2 passed; 0 failed; finished in 621.37ms

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:
804023, ~: 805360)
Test result: ok. 1 passed; 0 failed; finished in 785.34ms

Running 1 test for test/TokenLocker.sol/Access.t.sol:TestlockerAccess
[PASS] testAdminGetter() (gas: 20849)
Test result: ok. 1 passed; 0 failed; finished in 3.05ms

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: 29138, ~: 23853)
Test result: ok. 5 passed; 0 failed; finished in 593.95ms

Running 8 tests for test/TokenLocker.sol/IncreaseAmountsFor.t.sol:TestlockerIncreaseAmountFor
[PASS] testFuzz_AllReceiversNeedALock(uint120,uint184[5]) (runs: 256, μ: 678936, ~: 678954)
[PASS] testFuzz_CanIncreaseAmountForMany(uint120,uint128[5],uint8[5]) (runs: 256, μ: 743166, ~: 743107)
[PASS] testFuzz_CannotHaveAZeroAmount(uint120,uint128[5]) (runs: 256, μ: 697415, ~: 699807)
[PASS] testFuzz_CannotHaveDifferentLenghtParam(address[],uint192[]) (runs: 256, μ: 163869, ~: 163819)
[PASS] testFuzz_CannotIncreaseAmountForExpiredLock(address,uint128,uint8) (runs: 256, μ: 279175, ~:
279175)
[PASS] testFuzz_CannotIncreaseAmountForManyBelowMin(address,uint128,uint8) (runs: 256, μ: 282006, ~:
282006)
[PASS] testFuzz_DepositorCanUseIncreaseAmount(address,uint128,uint8) (runs: 256, μ: 199437, ~: 199429)
[PASS] testFuzz_DepositorCannotUseIncreaseAmountForMany(address,uint128,uint8) (runs: 256, μ: 262749, ~:
262759)
Test result: ok. 8 passed; 0 failed; finished in 3.23s

Running 9 tests for test/TokenLocker.sol/Deposits.t.sol:TestlockerDeposits
[PASS] testFuzz_CannotDepositOnBehalfOfAnotherUnlessWhitelisted(address,uint128,uint8,address) (runs:
256, μ: 189707, ~: 189717)
[PASS] testFuzz_CannotDepositToContractUnlessWhitelisted(address,uint128,uint8) (runs: 256, μ: 380288, ~:
380288)
[PASS] testFuzz_CannotDepositTwice(address,uint128,uint8) (runs: 256, μ: 154813, ~: 154823)
[PASS] testFuzz_DepositRevertsBelowMin(address,uint192,uint8,uint192) (runs: 256, μ: 101260, ~: 105920)
[PASS] testFuzz_DepositRevertsOutOfRangeMonths(address,uint128,uint8,uint8,uint8) (runs: 256, μ: 5016800,
~: 5016854)
[PASS] testFuzz_DepositWithSignature(uint128,uint128,uint8,uint24,uint256) (runs: 256, μ: 181026, ~:
181026)
[PASS] testFuzz_DepositWithSignatureToExternalReceiver(uint128,address,uint128,uint8,uint24,uint256)
(runs: 256, μ: 214490, ~: 214490)
[PASS] testFuzz_HasLock(address,uint128,uint8) (runs: 256, μ: 155498, ~: 155508)
[PASS] testFuzz_SuccessfulDeposit(address,uint128,uint8,uint24) (runs: 256, μ: 159755, ~: 159755)
Test result: ok. 9 passed; 0 failed; finished in 2.59s

Running 1 test for test/TokenLocker.sol/EarlyTermination.sol:TestlockerTerminateEarly
[PASS] testFuzz_TerminateEarly(address,uint128,uint8,address,uint256) (runs: 256, μ: 325960, ~: 322069)
Test result: ok. 1 passed; 0 failed; finished in 384.38ms
```

```
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 413.52ms

Running 2 tests for test/TokenLocker.sol/Eject.t.sol:TestlockerEject
[PASS] testFuzz_CanEjectGetter(address,uint128,uint8,uint32,address) (runs: 256, μ: 180357, ~: 180357)
[PASS] testFuzz_canEject(address[2],uint128[2],uint8[2],uint32) (runs: 256, μ: 341713, ~: 341777)
Test result: ok. 2 passed; 0 failed; finished in 4.89s

Running 2 tests for test/TokenLocker.sol/EmergencyWithdraw.t.sol:TestlockerEmergencyWithdraw
[PASS] testEmergencyReverts() (gas: 32878)
[PASS] testEmergencyWithdraw(address,uint128,uint8) (runs: 256, μ: 187512, ~:
187511)
Test result: ok. 2 passed; 0 failed; finished in 260.13ms

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: 1294177, ~: 1572105)
[PASS] testFuzz_PreviewSingleLockXAuxoNoExpiry(address,address[5],uint8[5],uint128[5],uint256,address)
(runs: 256, μ: 1626122, ~: 1631174)
Test result: ok. 2 passed; 0 failed; finished in 3.13s

Running 1 test for
test/Upgradoor.sol/integration/GetAmountAndLongestDuration.t.sol:TestGetAmountAndLongestDuration
[PASS] testFuzz_GetAmountAndLongestDuration(address,uint8[5],uint128[5]) (runs: 256, μ: 644463, ~:
644232)
Test result: ok. 1 passed; 0 failed; finished in 435.39ms

Running 8 tests for test/rewards/RewardsDelegation.t.sol:TestDistributorDelegate
[PASS] testCanAddRemoveDelegate(address,address) (runs: 256, μ: 64526, ~: 64526)
[PASS] testCannotClaimIfInvalidDelegate(address) (runs: 256, μ: 443409, ~: 443409)
[PASS] testCannotClaimIfSomeoneElsesDelegate(address,address) (runs: 256, μ: 521194, ~: 521194)
[PASS] testCannotDelegateUnlessWhiteListed(address,address) (runs: 256, μ: 74065, ~: 74065)
[PASS] testDelegatedClaim(address) (runs: 256, μ: 498066, ~: 498066)
[PASS] testInvalidDelegatedMultiClaim(address,address) (runs: 256, μ: 537422,
~: 537422)
[PASS] testNoEmptyClaims(address) (runs: 256, μ: 69080, ~: 69080)
[PASS] testSuccessfulDelegatedMultiClaim(address) (runs: 256, μ: 569001, ~: 569001)
Test result: ok. 8 passed; 0 failed; finished in 895.33ms

Running 8 tests for test/PRV/StakingManager.t.sol:TestStakingManager
[PASS] testBoostToMax(uint256,uint32) (runs: 256, μ: 80731, ~: 80851)
[PASS] testCantCallIncreaseWithoutLock(uint192) (runs: 256, μ: 11451269, ~: 11451269)
[PASS] testChangeGovernor(address) (runs: 256, μ: 52337, ~: 52337)
[PASS] testIncrease(uint184) (runs: 256, μ: 106242, ~: 106242)
[PASS] testInitialState() (gas: 26875)
[PASS] testOnlyGovernance(address,uint256) (runs: 256, μ: 90876, ~: 90876)
[PASS] testStake(uint256) (runs: 256, μ: 11592507, ~: 11592507)
[PASS] testStakeRevertsIfDepositTooLow(uint256) (runs: 256, μ: 11463157, ~: 11464401)
Test result: ok. 8 passed; 0 failed; finished in 1.58s

Running 3 tests for test/TokenLocker.sol/InitialState.t.sol:TestlockerInitialState
[PASS] testInitialStateRatioArray(address,address,uint32,uint32,uint192) (runs: 256, μ: 5049074, ~:
5049074)
[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 773.35ms

Running 2 tests for test/Auxo.sol/TransferAdmin.t.sol:TestTransferAdmin
[PASS] testFuzz_ChangingAuxoAdmin(address,address) (runs: 256, μ: 90756, ~: 90764)
[PASS] testFuzz_TransferOfAdminRole(address) (runs: 256, μ: 44794, ~: 44794)
Test result: ok. 2 passed; 0 failed; finished in 179.95ms

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 237.55ms

Running 7 tests for test/PRV/rollStaker/invariant/RollStaker.invariant.t.sol:RollStakerInvariantTest
[PASS] invariantTestActiveBalanceEqActive() (runs: 256, calls: 3840, reverts:
2924)
[PASS] invariantTestActivePlusPendingIsAlwaysEqDeposited() (runs: 256, calls:
3840, reverts: 2924)
[PASS] invariantTestEpochPendingGEPendingDeposits() (runs: 256, calls: 3840, reverts: 2924)
[PASS] invariantTestNotActiveThisEpochActiveNextEqPendingBalanceEqTotal() (runs: 256, calls: 3840,
reverts: 2924)
[PASS] invariantTestTokenBalanceContactGEContractLocked() (runs: 256, calls: 3840, reverts: 2924)
[PASS] invariantTestTotalInContractAlwaysGeUserTotal() (runs: 256, calls: 3840, reverts: 2924)
[PASS] invariantTestZeroBalanceEqInactive() (runs: 256, calls: 3840, reverts:
2924)
Test result: ok. 7 passed; 0 failed; finished in 954.22ms

Running 12 tests for test/TokenLocker.sol/IncreaseLock.t.sol:TestlockerIncreaseLock
[PASS] testCannotGovernanceAttack(address) (runs: 256, μ: 221380, ~: 221390)
[PASS] testFuzz_CanIncreaseLockDuration(address,uint128,uint8,uint8,uint32) (runs: 256, μ: 186023, ~:
186023)
[PASS] testFuzz_CanIncreaseLockQty(address,uint128,uint8,uint128,uint32) (runs: 256, μ: 211561, ~:
211592)
[PASS] testFuzz_CanIncreaseQtyWithSig(uint128,uint128,uint128,uint8,uint256,uint32) (runs: 256, μ:
237926, ~: 237968)
[PASS] testFuzz_CannotIncreaseLockMoreThanMax(address,uint128,uint8,uint8) (runs: 256, μ: 155475, ~:
155485)
[PASS] testFuzz_CannotIncreaseLockQtyBelowMin(address,uint128,uint8) (runs: 256, μ: 205758, ~: 205748)
[PASS] testFuzz_CannotIncreaseLockWithZeroChange(address,uint128,uint8) (runs: 256, μ: 158524, ~: 158524)
[PASS] testFuzz_CannotIncreaseLockWithoutDeposit(address,uint128,uint8) (runs: 256, μ: 44106, ~: 44106)
[PASS] testFuzz_TerminateEarlyRevertsWithoutxAUXO(address,uint128,uint8) (runs: 256, μ: 157653, ~:
157653)
[PASS] testFuzz_repeatedIncreasesDoNotCauseRoundingErrors(address,uint128,uint8,uint128[50]) (runs: 256,
μ: 1027957, ~: 1028019)
[PASS] testFuzz_repeatedIncreasesDurationDoNotCauseRoundingErrors(address,uint128) (runs: 256, μ: 696438,
~: 696438)
[PASS] testFuzz_repeatedIncreasesInBothDoesntCauseIssues(address,uint128,uint128[50],bool[50]) (runs:
256, μ: 1008902, ~: 1000441)
Test result: ok. 12 passed; 0 failed; finished in 7.38s

Running 3 tests for test/TokenLocker.sol/Withdraw.t.sol:TestlockerWithdraw
[PASS] testFuzz_CanWithdrawOtherwise(address,uint128,uint8,uint32) (runs: 256, μ: 180384, ~: 180372)
[PASS] testFuzz_CannotMakeEmptyWithdraw() (gas: 15950)
[PASS] testFuzz_CannotWithdrawEarly(address,uint128,uint8,uint32) (runs: 256,
μ: 153257, ~: 153267)
Test result: ok. 3 passed; 0 failed; finished in 715.16ms

Running 6 tests for test/Upgradoor.sol/sharesTimelock/Migrate.t.sol:TestSharesTimelock
[PASS] testCannotMigrateTwice(address,uint8[10],uint128[10]) (runs: 256, μ: 1280863, ~: 1284444)
[PASS] testMigrate(address,uint8[10],uint128[10]) (runs: 256, μ: 1671142, ~: 1676365)
[PASS] testMigrateMany(address,uint8[10],uint128[10],uint8[]) (runs: 256, μ: 1151955, ~: 1134109)
[PASS] testMigrateManyReverts(address,uint8[10],uint128[10],uint8[]) (runs: 256, μ: 1142662, ~: 1138808)
[PASS] testMigrateReverts(address,address,uint256,uint8[10],uint128[10]) (runs: 256, μ: 1084405, ~:
1084819)
[PASS] testOwnable(address) (runs: 256, μ: 35482, ~: 35512)
Test result: ok. 6 passed; 0 failed; finished in 8.02s

Running 3 tests for test/TokenLocker.sol/invariant/Locker.Invariant.t.sol:RollStakerInvariantTest
[PASS] invariantTestNobodyWithoutALockHasARewardBalanceAndViceVersa() (runs: 256, calls: 3840, reverts:
3390)
[PASS] invariantTestRewardAlwaysEqExpected() (runs: 256, calls: 3840, reverts: 3390)
[PASS] invariantTestRewardTotalSupplyLEDepositTokenLocked() (runs: 256, calls: 3840, reverts: 3390)
Test result: ok. 3 passed; 0 failed; finished in 1.53s

Running 1 test for test/rewards/MerkleDistributorReentrant.t.sol:TestDistributor
[PASS] testNoReentrant() (gas: 149057)
Test result: ok. 1 passed; 0 failed; finished in 1.98ms
```

```
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 2.08ms

Running 7 tests for test/Upgradoor.sol/MigrateToARV.t.sol:MigrateToARV
[PASS] testAggregateAndBoost() (gas: 510334)
[PASS] testPreviewAggregateAndBoost() (gas: 578034)
[PASS] testPreviewupgradeSingleLockARV() (gas: 503952)
[PASS] testUpgradeFailIfReceiverHasVeDOUGH() (gas: 61349)
[PASS] testaggregateToARV() (gas: 591405)
[PASS] testpreviewAggregateARV() (gas: 738331)
[PASS] testupgradeSingleLockARV() (gas: 5944085)
Test result: ok. 7 passed; 0 failed; finished in 38.15ms

Running 6 tests for test/TokenLocker.sol/Migrate.t.sol:TestlockerMigrate
[PASS] testFuzz_CannotMigrateByDefault(address) (runs: 256, μ: 23875, ~: 23875)
[PASS] testFuzz_CannotMigrateIfAmountIsZero(address) (runs: 256, μ: 67734, ~:
67734)
[PASS] testFuzz_CannotMigrateIfLockIsExpired(address) (runs: 256, μ: 190884, ~: 190884)
[PASS] testFuzz_CannotMigrateIfMigratorIsNotSet(address) (runs: 256, μ: 185790, ~: 185790)
[PASS] testFuzz_Migrate(address,uint128,uint8) (runs: 256, μ: 215955, ~: 215948)
[PASS] testFuzz_OnlyMigratorCanCallMigrate(address) (runs: 256, μ: 186957, ~:
186957)
Test result: ok. 6 passed; 0 failed; finished in 610.94ms

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: 900640, ~:
903513)
[PASS] testFuzz_ExitToXAuxoWithExpiry(address,uint8[5],uint128[5],uint64,uint256,address) (runs: 256, μ:
869207, ~: 925950)
Test result: ok. 2 passed; 0 failed; finished in 1.34s

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 408.15ms

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 29.42ms

Running 1 test for test/PRV/rollStaker/RollStaker.scenario.t.sol:RollStakerScenarioTest
[PASS] testKitchenSink(uint8,uint96[10],uint96[10]) (runs: 256, μ: 2120859, ~: 2129877)
Test result: ok. 1 passed; 0 failed; finished in 2.93s

Running 5 tests for test/Upgradoor.sol/integration/MigrateToARV.t.sol:TestMigrateToARV
[PASS] testFuzz_ExitToVeAuxoBoosted(address,uint8[5],uint128[5]) (runs: 256, μ: 799645, ~: 799429)
[PASS] testFuzz_ExitToVeAuxoBoostedWithExpiry(address,uint8[5],uint128[5],uint64) (runs: 256, μ: 753030,
~: 812197)
[PASS] testFuzz_ExitToVeAuxoNonBoosted(address,uint8[5],uint128[5]) (runs: 256, μ: 822631, ~: 822497)
[PASS] testFuzz_ExitToVeAuxoNonBoostedWithExpiry(address,uint8[5],uint128[5],uint64) (runs: 256, μ:
787809, ~: 835302)
[PASS] testFuzz_migrateNoLocksRevertsGracefully() (gas: 368234)
Test result: ok. 5 passed; 0 failed; finished in 2.20s

Running 15 tests for test/PRV/PRV.t.sol:PRVTest
[PASS] testDepositFor(uint256,uint184,address,address) (runs: 256, μ: 218107,
~: 212943)
[PASS] testDepositForSignatureInValid(uint128,address,uint128,uint256,uint8,bytes32,bytes32) (runs: 256,
μ: 85632, ~: 86504)
[PASS] testDepositForSignatureValid(uint128,address,uint128,uint256) (runs: 256, μ: 206657, ~: 206663)
[PASS] testFeeIsCharged(uint256,uint184) (runs: 256, μ: 169042, ~: 163112)
[PASS] testFeeNotChargedIfNoBeneficiarySet(uint256,uint128) (runs: 256, μ: 157310, ~: 158200)
[PASS] testInitialBalances() (gas: 29700)
[PASS] testInitialState(address,address,address,uint256,address) (runs: 256, μ: 1741666, ~: 1743377)
[PASS] testInitialStateReverts(address,address,address,uint256,address) (runs: 256, μ: 335071, ~: 335071)
[PASS] testOnlyGovernance(address) (runs: 256, μ: 35579, ~: 35579)
[PASS] testSetFeeBeneficiary(address) (runs: 256, μ: 22113, ~: 22113)
[PASS] testSetFeeBeneficiaryRevertsZeroAddr() (gas: 13329)
[PASS] testSetFeePolicy(address,uint256) (runs: 256, μ: 44442, ~: 46075)
```

```
[PASS] testSetFeeRevertAboveMax(uint256) (runs: 256, μ: 14845, ~: 14845)
[PASS] testSetGovernor(address) (runs: 256, μ: 20179, ~: 20179)
[PASS] testSetGovernorRevertsZeroAddr() (gas: 13275)
Test result: ok. 15 passed; 0 failed; finished in 1.20s

Running 6 tests for test/Upgradoor.sol/MigrateToPRV.t.sol:TestAggregateToPRV
[PASS] testPreviewaggregateToPRV() (gas: 597057)
[PASS] testaggregateToPRV() (gas: 520790)
[PASS] testaggregateToPRVAndStake() (gas: 701085)
[PASS] testpreviewUpgradeSingleLockPRV() (gas: 498115)
[PASS] testupgradeSingleLockPRV() (gas: 4763304)
[PASS] testupgradeSingleLockPRVAndStake() (gas: 7633345)
Test result: ok. 6 passed; 0 failed; finished in 62.54ms

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: 1535959,
~: 1535984)
Test result: ok. 1 passed; 0 failed; finished in 2.00s

Running 5 tests for test/PRV/PRVRouter.t.sol:PRVTestRouter
[PASS] testFuzz_CanConvertAndStake(address,uint120) (runs: 256, μ: 340193, ~:
340188)
[PASS] testFuzz_CanConvertAndStakeWithFee(address,uint120,uint256,address) (runs: 256, μ: 375437, ~:
370658)
[PASS] testFuzz_CanConvertAndStakeWithReceiver(address,address,uint120) (runs: 256, μ: 342508, ~: 342489)
[PASS]
testFuzz_convertAndStakeWithSignatureInValid(uint128,address,uint120,uint256,uint8,bytes32,bytes32)
(runs: 256, μ: 86825, ~: 87710)
[PASS] testFuzz_convertAndStakeWithSignatureValid(uint128,address,uint120,uint256) (runs: 256, μ: 373201,
~: 373213)
Test result: ok. 5 passed; 0 failed; finished in 977.93ms

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: 791588, ~: 793545)
Test result: ok. 1 passed; 0 failed; finished in 998.84ms

Running 20 tests for test/PRV/rollStaker/RollStaker.t.sol:RollStakerTest
[PASS] testActivateNewEpoch(uint128,uint128,uint8,uint256) (runs: 256, μ: 313778, ~: 153977)
[PASS] testCanWithdrawAcrossMultipleEpochs(uint120,address) (runs: 256, μ: 268473, ~: 268468)
[PASS] testCannotWithdrawMoreThanDeposited(uint120,address,uint256) (runs: 256, μ: 190754, ~: 190749)
[PASS] testDepositFor(uint120,address,address) (runs: 256, μ: 288627, ~: 288627)
[PASS] testDepositPermit(uint120,uint256,uint128) (runs: 256, μ: 185399, ~: 185399)
[PASS] testDepositRollStaker(uint120,address) (runs: 256, μ: 279108, ~: 279108)
[PASS] testEmergencyWithdraw(address,uint120) (runs: 256, μ: 212763, ~: 212757)
[PASS] testGettersComputeCorrectlyAfterWithdraw(uint120,address) (runs: 256, μ: 239269, ~: 239264)
[PASS] testGettersDoNotRevert(uint8,address) (runs: 256, μ: 27181, ~: 27446)
[PASS] testLastEpochUserWasActive(address,uint8,uint8,uint8) (runs: 256, μ: 603522, ~: 724378)
[PASS] testOperatorRestrictedFunctions(address) (runs: 256, μ: 208748, ~: 208748)
[PASS] testPendingDepositsUpdateCorrectlyMultipleWithdrawal(address,address,uint96,uint96) (runs: 256, μ:
344843, ~: 344820)
[PASS] testPublicFunctionsPaused(uint120,address) (runs: 256, μ: 75880, ~: 75880)
[PASS] testQuit(uint120,address,uint256) (runs: 256, μ: 260304, ~: 260302)
[PASS] testRevertDeposit(uint120,address,uint256) (runs: 256, μ: 202782, ~: 202835)
[PASS] testRevertDepositActiveUserStaysActive(uint120,address) (runs: 256, μ:
250700, ~: 250691)
[PASS] testRevertDepositInActiveUserGoesBackToInactive(uint120,address) (runs: 256, μ: 240256, ~: 240249)
[PASS] testUserStaysActive(address,uint8,uint8,uint8) (runs: 256, μ: 1200441,
~: 1025555)
[PASS] testWithdraw(uint120,address,uint256) (runs: 256, μ: 235770, ~: 235796)[PASS]
testZeroAmountReverts() (gas: 69337)
Test result: ok. 20 passed; 0 failed; finished in 2.95s

Running 3 tests for test/PRV/bitfields.t.sol:TestBitfields
[PASS] testBitFieldLib(uint8,uint8,uint8) (runs: 256, μ: 5828043, ~: 6443378)
[PASS] testLastActiveDoesNotModifyLocalVariable(uint8,uint8,uint8) (runs: 256, μ: 34282, ~: 31219)
[PASS] testRepeatedActivationsAndDeactivationsAreIdempotent(uint8,uint8,uint8) (runs: 256, μ: 36213, ~:
33842)
Test result: ok. 3 passed; 0 failed; finished in 8.60s
```

```
Running 17 tests for test/rewards/MerkleDistributor.t.sol:TestDistributor
[PASS] testBadMultiClaim((uint256,uint256,uint256,address,bytes32[],address)[],address,address) (runs:
256, μ: 9200246, ~: 7885364)
[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: 9174674, ~: 7535799)[PASS]
testCannotMultiClaimForSomeoneElse(address) (runs: 256, μ: 373982, ~: 373982)
[PASS] testCannotMultiClaimWithPaddedArray() (gas: 374650)
[PASS] testClaim() (gas: 512175)
[PASS] testDeleteWindow(bytes32,string,uint256) (runs: 256, μ: 743476, ~: 746749)
[PASS] testEmergencyWithdraw(uint256,uint256) (runs: 256, μ: 775183, ~: 775261)
[PASS] testInvalidClaims((uint256,uint256,uint256,address,bytes32[],address))
(runs: 256, μ: 114256, ~: 113847)
[PASS] testLockedClaims(uint256,(uint256,uint256,uint256,address,bytes32[],address)) (runs: 256, μ:
91055, ~: 90890)
[PASS] testNoEmptyClaims() (gas: 38701)
[PASS] testOwnable(address) (runs: 256, μ: 33586, ~: 33586)
[PASS] testPausable(address) (runs: 256, μ: 61194, ~: 61194)
[PASS] testSetWindow(bytes32,string,uint256) (runs: 256, μ: 851462, ~: 864299)[PASS]
testSuccessfulMultiClaim() (gas: 461981)
Test result: ok. 17 passed; 0 failed; finished in 21.79s
```

Code Coverage

Run `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 branch coverage.

Update after the second re-audit : Branch coverage for all in-scope contracts is 100%, except for `TokenLocker.sol` (80%), `RollStaker.sol` (59.09%), and `MerkleDistributor.sol` (86.36%).

File	% Lines	% Statements	% Branches	% Funcs
script/Deploy. s.sol	0.00% (**0**/39)	0.00% (**0**/43)	0.00% (**0**/6)	0.00% (**0**/1)
script/DeployFork. s.sol	0.00% (**0**/56)	0.00% (**0**/61)	0.00% (**0**/8)	0.00% (**0**/1)
script/ DeployOracle.sol	0.00% (**0**/4)	0.00% (**0**/4)	0.00% (**0**/2)	0.00% (**0**/1)
script/DeployRewards. s.sol	0.00% (**0**/12)	0.00% (**0**/13)	100.00% (**0**/0)	0.00% (**0**/5)
script/ EjectVeDOUGHFreeriders. sol	0.00% (**0**/19)	0.00% (**0**/31)	0.00% (**0**/2)	0.00% (**0**/1)
src/ AUXO.sol	100.00% (**1**/1)	100.00% (**1**/1)	100.00% (**0**/0)	100.00% (**1**/1)
src/modules/LSD/ LsdRouter.sol	100.00% (**16**/16)	100.00% (**16**/16)	100.00% (**0**/0)	100.00% (**3**/3)
src/modules/LSD/ RollStaker.sol	84.62% (**66**/78)	84.27% (**75**/89)	64.29% (**9**/14)	88.89% (**24**/27)
src/modules/LSD/ StakingManag er.sol	84.21% (**16**/19)	85.71% (**18**/21)	100.00% (**0**/0)	85.71% (**6**/7)
src/modules/LSD/ bitfield.sol	100.00% (**13**/13)	100.00% (**15**/15)	100.00% (**2**/2)	100.00% (**7**/7)
src/modules/LSD/ xAUXO.sol	90.32% (**28**/31)	91.89% (**34**/37)	100.00% (**8**/8)	90.91% (**10**/11)
src/modules/governance/ EarlyT ermination.sol	83.33% (**5**/6)	83.33% (**5**/6)	100.00% (**2**/2)	66.67% (**2**/3)

File	% Lines	% Statements	% Branches	% Funcs
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	100.00% (**128**/128)	99.32% (**147**/148)	95.65% (**44**/46)	96.43% (**27**/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)
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/** Upgradoor.sol	100.00% (**104**/104)	99.33% (**149**/150)	82.35% (**28**/34)	100.00% (**19**/19)
src/ veAUXO.sol	100.00% (**7**/7)	100.00% (**7**/7)	100.00% (**0**/0)	100.00% (**7**/7)
test/Auxo.**sol/** Setup.sol	0.00% (**0**/1)	0.00% (**0**/1)	100.00% (**0**/0)	0.00% (**0**/1)
test/LSD/LsdBase. t.sol	0.00% (**0**/16)	0.00% (**0**/20)	100.00% (**0**/0)	0.00% (**0**/2)
test/LSD/rollStaker/ RollStakerTestInitializer.sol	53.33% (**8**/15)	44.44% (**8**/18)	100.00% (**0**/0)	80.00% (**4**/5)
test/LSD/rollStaker/RollStakerUpgrades. t.sol	0.00% (**0**/2)	0.00% (**0**/2)	100.00% (**0**/0)	0.00% (**0**/2)
test/LSD/rollStaker/invariant/RollStaker.invariant. t.sol	0.00% (**0**/61)	0.00% (**0**/82)	0.00% (**0**/30)	0.00% (**0**/8)
test/LSD/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	8.00% (**2**/25)	7.69% (**2**/26)	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**/44)	0.00% (**0**/58)	0.00% (**0**/22)	0.00% (**0**/5)
test/TokenLocker.**sol/invariant/** LockerNonUpgradeable.sol	0.00% (**0**/124)	0.00% (**0**/144)	0.00% (**0**/46)	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)

File	% Lines	% Statements	% Branches	% Funcs
test/ UpgradeDeployer.sol	0.00% (**0**/39)	0.00% (**0**/54)	100.00% (**0**/0)	0.00% (**0**/9)
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**/32)	0.00% (**0**/34)	0.00% (**0**/4)	0.00% (**0**/2)
test/Upgradoor.**sol/integration/**Setup. t.sol	0.00% (**0**/107)	0.00% (**0**/132)	0.00% (**0**/20)	0.00% (**0**/12)
test/fork/ SharesTimeLock.sol	36.92% (**48**/130)	37.18% (**58**/156)	25.00% (**17**/68)	21.88% (**7**/32)
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)
test/mocks/ Token.sol	100.00% (**4**/4)	100.00% (**4**/4)	100.00% (**0**/0)	100.00% (**2**/2)
test/rewards/ MerkleTreeInitializer.sol	2.13% (**1**/47)	2.08% (**1**/48)	100.00% (**0**/0)	14.29% (**1**/7)
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)
test/veAuxo.**sol/** Setup.sol	0.00% (**0**/1)	0.00% (**0**/1)	100.00% (**0**/0)	0.00% (**0**/1)
Total	39.32% (**558**/1419)	39.17% (**667**/1703)	36.63% (**137**/374)	46.13% (**155**/336)

Second re-audit

File	% Lines	% Statements	% Branches	% Funcs
script/Deploy-v1. s.sol	0.00% (**0**/144)	0.00% (**0**/156)	0.00% (**0**/60)	0.00% (**0**/12)
script/ HealthCheck.sol	0.00% (**0**/121)	0.00% (**0**/127)	0.00% (**0**/200)	0.00% (**0**/16)
script/Simulation. s.sol	0.00% (**0**/353)	0.00% (**0**/447)	0.00% (**0**/124)	0.00% (**0**/22)
script/old/Deploy-v0. s.sol	0.00% (**0**/38)	0.00% (**0**/43)	0.00% (**0**/6)	0.00% (**0**/1)
script/old/DeployFork. s.sol	0.00% (**0**/56)	0.00% (**0**/61)	0.00% (**0**/8)	0.00% (**0**/1)
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	90.62% (**29**/32)	92.31% (**36**/39)	100.00% (**10**/10)	90.91% (**10**/11)

File	% Lines	% Statements	% Branches	% Funcs
src/modules/PRV/ PRVRouter.sol	100.00% (**19**/19)	100.00% (**22**/22)	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	82.35% (**14**/17)	84.21% (**16**/19)	100.00% (**0**/0)	83.33% (**5**/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	100.00% (**129**/129)	99.33% (**148**/149)	80.00% (**48**/60)	96.43% (**27**/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)
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/PRVBase. t.sol	5.56% (**1**/18)	4.76% (**1**/21)	100.00% (**0**/0)	33.33% (**1**/3)
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	7.69% (**2**/26)	7.41% (**2**/27)	100.00% (**0**/0)	16.67% (**1**/6)

File	% Lines	% Statements	% Branches	% Funcs
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**/44)	0.00% (**0**/58)	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**/49)	0.00% (**0**/65)	100.00% (**0**/0)	0.00% (**0**/11)
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**/32)	0.00% (**0**/34)	0.00% (**0**/4)	0.00% (**0**/2)
test/Upgradoor.**sol/integration/**Setup. t.sol	0.00% (**0**/109)	0.00% (**0**/135)	0.00% (**0**/20)	0.00% (**0**/12)
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)
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	29.00% (**598**/2062)	29.15% (**720**/2470)	20.72% (**162**/782)	42.89% (**169**/394)

Changelog

- 2023-02-17 - Initial report
- 2023-03-07 - First re-audit
- 2023-03-24 - Second re-audit

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, Aave, 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