



Executive Summary

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

Type	ERC20
Timeline	2023-10-23 through 2023-10-27
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Manifesto Website
Source Code	<ul style="list-style-type: none"> Moai-Labs/vepoch-contracts #b5f3770 Moai-Labs/vepoch-contracts #53ec1b3
Auditors	<ul style="list-style-type: none"> Adrian Koegl Auditing Engineer Roman Rohleder Senior Auditing Engineer Poming Lee Senior Auditing Engineer

Documentation quality	Low 
Test quality	Medium 
Total Findings	18 Fixed: 4 Acknowledged: 12 Mitigated: 2
High severity findings ⓘ	1 Fixed: 1
Medium severity findings ⓘ	2 Fixed: 1 Acknowledged: 1
Low severity findings ⓘ	6 Fixed: 2 Acknowledged: 3 Mitigated: 1
Undetermined severity findings ⓘ	0
Informational findings ⓘ	9 Acknowledged: 8 Mitigated: 1

Summary of Findings

VEPOCH is an ERC-20 veToken contract that allows the deposit and lock of a `depositToken` to receive vEPOCH, representing someone's voting power according to the amount and duration of locked deposits. All current depositors can additionally earn rewards, however, without a well-defined release schedule. Overall, the code is well-written but lacks some documentation. We have not found any significant security vulnerabilities but could identify several edge cases with unexpected and unintended behavior. We recommend incorporating all fixes presented in this report.

ID	DESCRIPTION	SEVERITY	STATUS
vEPO-1	Calling <code>withdrawForfeit()</code> Multiple Times for a Single Deposit Would Cause Loss of Funds	• High ⓘ	Fixed
vEPO-2	Attacker May Reduce Other User Rewards	• Medium ⓘ	Fixed
vEPO-3	ERC20 Token Transfer May Silently Fail	• Medium ⓘ	Acknowledged
vEPO-4	<code>owner</code> May Prevent Any Further Deposits	• Low ⓘ	Acknowledged
vEPO-5	Checks-Effects-Interactions Pattern Violation	• Low ⓘ	Fixed
vEPO-6	Authorised Addresses Can Always Transfer Once Control Is Given Up	• Low ⓘ	Acknowledged
vEPO-7	vEPOCH Token Transferral Can Lead to Assets Being Locked Forever	• Low ⓘ	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
vEPO-8	Unlocked Pragma	• Low ⓘ	Fixed
vEPO-9	Rounding Errors Could Be Intentionally Introduced to Keep the <code>rewardStakingPower</code>	• Low ⓘ	Mitigated
vEPO-10	Logic Missing in <code>transfer()</code> and <code>transferFrom()</code> of the <code>veToken</code>	• Informational ⓘ	Acknowledged
vEPO-11	Lack of Clear Reward Incentive	• Informational ⓘ	Acknowledged
vEPO-12	Gas Improvements	• Informational ⓘ	Mitigated
vEPO-13	Privileged Roles and Ownership	• Informational ⓘ	Acknowledged
vEPO-14	Missing Input Validation	• Informational ⓘ	Acknowledged
vEPO-15	Ownership Can Be Renounced	• Informational ⓘ	Acknowledged
vEPO-16	Application Monitoring Can Be Improved by Emitting More Events	• Informational ⓘ	Acknowledged
vEPO-17	Critical Role Transfer Not Following Two-Step Pattern	• Informational ⓘ	Acknowledged
vEPO-18	<code>deposits</code> Index Will Start at 1	• 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.

i 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.

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.

Scope

Files Included

Repo: [https://github.com/Moai-Labs/vepoch-contracts/\(b5f3770\)](https://github.com/Moai-Labs/vepoch-contracts/(b5f3770))

Files: [tree/master/contracts/Vepoch.sol](#)

Findings

vEPO-1

Calling `withdrawForfeit()` Multiple Times for a Single Deposit Would Cause Loss of Funds

• High ⓘ Fixed

✓ Update

The client fixed this issue in commit `3f95022`. `rewardTokensClaimed` is now updated, so a user will not have to pay back more rewards if forfeited multiple times.

File(s) affected: `Vepoch.sol`

Description: When calling `withdrawForfeit()`, the rewards that need to be paid back to the contract is calculated as follows:

```
uint256 forfeitReward = ((earned[_depositId] + rewardTokensClaimed[_depositId]) * percentage) / 1e18;
```

However, the `rewardTokensClaimed` is never updated after the payback. Thus, when users don't withdraw 100 percent of their deposit, they will repay more rewards than they are obligated to with the following withdrawals. Furthermore, the user won't be able to unlock the entire deposit unless they buy `rewardToken` to pay back.

Recommendation: Update `rewardTokensClaimed[]` based on the repaid amount and also add the modified amount information to `event WithdrawnForfeit` emission.

vEPO-2 Attacker May Reduce Other User Rewards

• Medium ⓘ Fixed

✓ Update

The client removed the functionality entirely in commit `53ec1b3`. All rewards are now transferred back to the contract upon forfeit.

File(s) affected: `Vepoch.sol`

Description: An attacker may be able to reduce the rewards received by other users without any penalty, only incurring gas costs. Such an attack is possible whenever `forfeitRedirectionAddress != address(0)`.

The described exploit scenario will lead to the fact that current deposits will receive fewer reward shares.

In case the attacker has control over `forfeitRedirectionAddress`, they will be able to claim those rewards without any deposits being locked.

Exploit Scenario:

1. Attacker observes the mempool for a call to `addRewardTokens()`. They will proceed with the following steps if the added rewards are sufficiently high.
2. The attacker front-runs this transaction and calls `deposit()` with as many tokens as possible and maximum duration (to mint the maximum amount of vEPOCH tokens).
3. After the attacker has successfully reduced other deposit rewards, they call `withdrawForfeit()` to get back their deposit.

Recommendation: Consider removing the possibility of sending rewards to a different address and always distribute cancelled rewards to current deposits.

vEPO-3 ERC20 Token Transfer May Silently Fail

• Medium ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

This contract will be deployed and used with known ERC20 contracts that adhere to the ERC20 standard

File(s) affected: `Vepoch.sol`

Description: Whenever a `rewardToken` or `depositToken` is transferred, they may fail without reverting. Depending on the context, this may lead to inconsistencies, users losing funds, or the contract losing funds.

The reason is that calls to `transfer()` and `transferFrom()` are not checked for returning true. While many ERC20s will revert upon transfer failure, some ERC20s will return false in adherence to the ERC20 specification. If `rewardToken` or `depositToken` will return false on failure the transfer will silently fail.

The return value is not checked in the following places:

1. `Vepoch.sol#L78` .
2. `Vepoch.sol#L108` .
3. `Vepoch.sol#L130` .
4. `Vepoch.sol#L146` .
5. `Vepoch.sol#L192` .
6. `Vepoch.sol#L210` .
7. `Vepoch.sol#L215` .
8. `Vepoch.sol#L234` .
9. `Vepoch.sol#L252` .

Recommendation: Since some ERC20 tokens do not return a value on above mentioned functions/ behave non-uniformly it is advised to use a secure wrapping interface around said functions, like for example OpenZeppelins `SafeERC20`, which implicitly checks for the different error cases.

vEPO-4 `owner` May Prevent Any Further Deposits

• Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

Design choice to allow for future migrations if necessary. It does not impact existing deposits, only new deposits.

File(s) affected: `Vepoch.sol`

Description: The `owner` may (accidentally) prevent further user deposits by setting `maxDepositDuration < 60` . This would lead to `deposit()` always reverting as `_duration > 59` and `_duration <= maxDepositDuration` cannot hold true at the same time for `maxDepositDuration < 60` .

The root issue is that `setMaxDepositDuration()` does not enforce a lower bound on `maxDepositDuration` .

Furthermore, the `owner` can (accidentally) set a too-low `maxDepositDuration` and lock that value forever by calling `setMaxDepositDurationLocked()` . This would lock out any future calls to `setMaxDepositDuration()` to undo such a setting

Recommendation: Consider enforcing a minimum value for `maxDepositDuration` in `setMaxDepositDuration()` . The lower bound should at least be 60, although the team should consider to set it to a higher value in order to prevent locking in a too low value.

vEPO-5 Checks-Effects-Interactions Pattern Violation

• Low ⓘ Fixed

Update

The CEI pattern was applied to all listed functions in commit `a788632` .

File(s) affected: `Vepoch.sol`

Related Issue(s): [SWC-107](#)

Description: The [Checks-Effects-Interactions](#) coding pattern is meant to mitigate any chance of other contracts manipulating the state of the blockchain in unexpected and possibly malicious ways before control is returned to the original contract. As the name implied, only after checking whether appropriate conditions are met and acting internally on those conditions should any external calls to, or interactions with, other contracts be done.

The following functions are violating the "Checks-Effects-Interactions"-pattern and/or do not employ any reentrancy mitigating modifiers, making them susceptible to reentrancy:

1. `Vepoch.addRewardTokens()` .
2. `Vepoch.claimYield()` .
3. `Vepoch.deposit()` .
4. `Vepoch.withdrawForfeit()` .
5. `Vepoch.withdraw()` .

Recommendation: We recommend refactoring the code so that it conforms to the Checks-Effects-Interactions pattern and/or consider employing the `nonReentrant` -modifier from [OpenZeppelin](#) to prevent any potential reentrancies.

vEPO-6

Authorised Addresses Can Always Transfer Once Control Is Given Up

• Low ⓘ

Acknowledged

i Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

Intentional design choice

File(s) affected: `Vepoch.sol`

Description: Once the `owner` gives up control over authorising addresses by calling `setAuthorisedLocked()` , all authorised addresses will forever be able to transfer vEPOCH tokens. While the `owner` can "unauthorize" addresses, they may not do so for all intended addresses before giving up control.

Recommendation: Consider whether this is intended design. If no one should be able to transfer vEPOCH tokens after `owner` has called `setAuthorisedLocked()` , add a requirement to `transfer()` and `transferFrom()` that `!authorisedLocked` . This will allow the `owner` to not only give up control but disallow transfers for all addresses.

vEPO-7

vEPOCH Token Transferal Can Lead to Assets Being Locked Forever

• Low ⓘ

Acknowledged

i Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

Intentional design choice. Future use cases where vEPOCH can be transferred will require detailed vetting.

File(s) affected: `Vepoch.sol`

Description: If an address or third party is authorized to transfer vEPOCH tokens, they may not be able to recover their locked deposits when they transfer vEPOCH tokens.

Withdrawing entire deposits requires providing the initially minted amount of vEPOCH tokens. If some of them are transferred but cannot be obtained anymore, (parts of) the deposits cannot be recovered.

Recommendation: Consider and elaborate on the scenarios where authorizing token transfers is required. Document these cases or remove the `transfer()` and `transferFrom()` functions.

vEPO-8 Unlocked Pragma

• Low ⓘ

Fixed

✓ Update

The pragma was fixed to 0.8.17 in commit `29b5dda` .

File(s) affected: `Vepoch.sol`

Related Issue(s): [SWC-103](#)

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*` . The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend removing the caret to lock the file onto a specific Solidity version.

vEPO-9

Rounding Errors Could Be Intentionally Introduced to Keep the

• Low ⓘ

Mitigated

rewardStakingPower

✓ Update

The client mitigated this issue in commit `0e30e00`. Deposits are now deleted when 100% is forfeited. However, the second part of the recommendation was not implemented.

File(s) affected: `Vepoch.sol`

Description: In `withdrawForfeit()`, the deposit is never deleted even when `percentage` is 100%. So an attacker could call `withdrawForfeit()` multiple times and pass in engineered numbers of `_depositTokensToRemove` to deliberately introduce rounding errors that allow the contract to return all of the deposited tokens while keeping some dust `rewardStakingPower` inside the contract. If this happens, the attacker can be rewarded without keeping any deposits in the contract.

Recommendation:

- `withdrawForfeit()` should delete the deposit whenever the `percentage` is 100%.
- `withdrawForfeit()` should deduct some fee from users to increase the cost of entering and re-entering the contract.

vEPO-10

Logic Missing in `transfer()` and `transferFrom()` of the veToken

• Informational ⓘ

Acknowledged

i Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

Intentional design choice

File(s) affected: `Vepoch.sol`

Description: Although this issue is only for authorized users, when `veToken` is transferred, the `rewardStakingPower`, `rewardIndexOf` and the deposits are not changed accordingly. This could cause the inability of authorized users in `withdraw()` and `withdrawForfeit()`. This would also lead to an incorrect calculation of rewards.

Recommendation: Either update `rewardStakingPower` and `rewardIndexOf` upon transferal or remove `transfer()` and `transferFrom()` entirely.

vEPO-11 Lack of Clear Reward Incentive

• Informational ⓘ

Acknowledged

i Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

Intentional design choice

File(s) affected: `Vepoch.sol`

Description: The reward mechanism is supposed to be an incentive for users to lock their `depositToken`. However, their received rewards depends on the amount of rewards provided through `addRewardTokens()` during their lock period.

This uncertainty may not pose a stable incentive for users to deposit and lock their tokens.

Recommendation: Either implement a mechanism that allows users to predict the rewards received or communicate a clear reward schedule.

vEPO-12 Gas Improvements

• Informational ⓘ

Mitigated

i Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

File(s) affected: `Vepoch.sol`**Description:** We identified some code sections where gas usage could be improved:

- `Vepoch.sol#L156` and `L160` : Consider caching the return value of the call to `calculateVeTokens(_tokenAmount, _duration)` in a new local variable and re-using it instead of having two calls.
- `deposits` is currently a mapping. As it is used with a strictly incrementing index, an array can also be used. Storing new values in an array is cheaper than in a mapping.

Recommendation: Make adjustments as described above to improve on gas efficiency.

vEPO-13 Privileged Roles and Ownership

• Informational ⓘ Acknowledged

i Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

Intentional design choice

File(s) affected: `Vepoch.sol`**Description:** Specific contracts have state variables, e.g., `owner`, which provide certain addresses with privileged roles. Such roles may pose a risk to end-users.

For a detailed list of roles per contract, see the following list:

The `Vepoch.sol` contract contains the following privileged roles:

- `_owner` (`Ownable` inheritance), as initialized during the execution of `constructor` to `msg.sender` :
 - Renounce the role (**and thereby prevent any future calls to the following listed functions!**) by calling `renounceOwnership()`.
 - Transfer the ownership to an arbitrary other address by calling `transferOwnership()`.
 - Forever prevent any future calls to function `setAuthorised()` for changing transfer-authorized addresses by calling `setAuthorisedLocked()`.
 - Forever prevent any future calls to function `setMaxDepositDuration()` for changing the maximum deposit time (up to 315576000 seconds or 3652,5 days) by calling `setMaxDepositDurationLocked()`.
 - Forever prevent any future calls to function `setForfeitRedirection()` for setting/unsetting a forfeit reward redistribution address by calling `setForfeitRedirectionAddressLocked()`.
 - Add/Remove addresses from the `veToken` transfer whitelist (only addresses in that list may transfer `veToken`s via `transfer()` / `transferFrom()`) as long as this function is not locked via `setAuthorisedLocked()` by calling `setAuthorised()`.
 - Change the maximum deposit duration (up to 315576000 seconds or 3652,5 days) as long as this function is not locked via `setMaxDepositDuration()` by calling `setMaxDepositDuration()`.
 - Set/Unset a forfeit reward redistribution address as long as this function is not locked via `setForfeitRedirectionAddressLocked()` by calling `setForfeitRedirection()`.

A malicious or compromised owner could in a worst-case therefore:

- Arbitrarily allow/disallow `veToken` transfers by adding/removing addresses from the authorized whitelist, as long as `setAuthorisedLocked()` has not been called.
- Denial the service of any future deposits by setting a maximum deposit time (`maxDepositDuration`) of less than 60 seconds, as long as `setMaxDepositDurationLocked()` has not been called.

Recommendation: Clarify the impact of these privileged actions to the end-users via publicly facing documentation.

vEPO-14 Missing Input Validation

• Informational ⓘ Acknowledged

i Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

Deployment of the contract should be performed diligently and deployer must ensure both the deposit token and yield token are not the 0 address. Input validation on setting max deposit duration was a intentional design choice

File(s) affected: `Vepoch.sol`**Description:** It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. The following functions do not have a proper validation of input parameters:

- `Vepoch.constructor()` : `_depositToken` and `_yieldToken` not checked to be different from `address(0x0)`.

2. `Vepoch.setMaxDepositDuration() : _newMaxDepositDuration` not checked to be greater than 59 or any other reasonable lower bound.

Recommendation: Consider adding according input checks.

vEPO-15 Ownership Can Be Renounced

• Informational ⓘ Acknowledged

i Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

Intentional design choice

File(s) affected: `Vepoch.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.

Recommendation: Confirm that this is the intended behavior. If not, override and disable the `renounceOwnership()` function in the affected contracts. For extra security, consider using a two-step process when transferring the ownership of the contract (e.g. `Ownable2Step` from OpenZeppelin).

vEPO-16

Application Monitoring Can Be Improved by Emitting More Events

• Informational ⓘ Acknowledged

i Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

Required values can be read directly from the contract

File(s) affected: `Vepoch.sol`

Description: In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transition can be logged, which is beneficial for monitoring the contract, and also tracking eventual bugs, or hacks. Below we present a non-exhaustive list of events that could be emitted to improve the application management:

1. `Vepoch.addRewardTokens() : rewardIndex`.
2. `Vepoch._updateRewards() : earned[] and rewardIndexOf[]`.
3. `Vepoch.setAuthorisedLocked() : authorisedLocked`.
4. `Vepoch.setMaxDepositDurationLocked() : maxDepositDurationLocked`.
5. `Vepoch.setForfeitRedirectionAddressLocked() : forfeitRedirectionAddressLocked`.
6. `Vepoch.setForfeitRedirection() : forfeitRedirectionAddress`.

Recommendation: Consider emitting the events.

vEPO-17

Critical Role Transfer Not Following Two-Step Pattern

• Informational ⓘ Acknowledged

i Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

The owner address will belong to a gnosis safe controlled by DAO. This additional check is not necessary.

File(s) affected: `Vepoch.sol`

Description: The owner of the contracts can call `transferOwnership()` to transfer the ownership to a new address. If an uncontrollable address is accidentally provided as the new owner address then the contract will no longer have an active owner, and functions with the `onlyOwner` modifier can no longer be executed.

Recommendation: Consider using OpenZeppelin's `Ownable2Step` contract to adopt a two-step ownership pattern in which the new owner must accept their position before the transfer is complete.

i Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

This does not impact the security of the contract.

File(s) affected: Vepoch.sol

Description: The `deposits` mapping will begin at index one upon first `deposit()` call, leaving the zero'th index empty. This is because `depositCount` is incremented before setting the next `deposits` value.

Particularly, if `deposits` is changed to an array for gas efficiency reasons, it is recommended to commence at index zero.

Recommendation: To begin with index zero, increment `depositCount` only after `deposits[depositCount]` is set and return `depositCount - 1`.

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. While some NatSpec inline code documentation is provided, they do not describe function parameters and return values. Consider adding these, at least for all publicly and externally callable functions.
2. Even though in the teams voiced concerns they mention they only plan to work with known ERC20 implementations and therefore intentionally disregard the use of SafeERC20, we nonetheless recommend adding corresponding documentation as inline code comments. This is in order to mitigate any potential future refactorings or forks from using unknown ERC20 implementations that may behave unexpectedly/different, potentially leading to undetected transactions failures and incorrect accounting.
3. It is not documented that authorized users can transfer vEPOCH. 1, The documentation doesn't explicitly state that locking LP tokens will immediately mint vEPOCH tokens.
4. It is not documented how and when rewards are paid out to depositors.

Adherence to Best Practices

1. Vepoch.sol#L19 : For increased readability and maintainability, consider making use of the **solidity time units**. In this regard, consider replacing `63072000` with `730 days`.
2. To improve readability and lower the risk of introducing errors when making code changes, it is advised not to use magic constants throughout code but instead declare them once (as constant and commented) and use these constant variables instead. The following instances should therefore be changed accordingly:
 1. Vepoch.sol#L79 , L85 , L138 , L179 , L182 , L186 , L208 and L247 : `1e18`.
 2. Vepoch.sol#L137 : `11574074074074`.
 3. Vepoch.sol#L145 : `59`.
 4. Vepoch.sol#L325 : `315576000`.

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 .`

Automated Analysis

Slither

Slither identified 103 issues in `Vepoch.sol` during the analysis of the project. However, most of them were false positives and only valid ones have been included in the report.

Test Suite Results

We followed the README, and ran the following commands:

```
pnpm install
pnpm compile
pnpm typechain
pnpm test
```

All 228 tests have successfully passed.

```
root@8c1dc8156e76:~/repo/moai_labs-vepoch-contracts-b5f3770-github# pnpm test
```

```
Admin: ForfeitRewardRedirection Tests
```

```
Yield Token: Epoch EPOCH 18
```

```
Deposit Token: LP Token LP 18
```

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (357ms)
- ✓ mint 1,000,000 yieldTokens into account 0 (133ms)
 - (1) depositTokenBalanceBefore-depositTokenBalanceAfter = 200
 - (1) veTokenBalanceAfter-veTokenBalanceBefore = 145999.999999999065600000
- ✓ (1) account 1 should lock 200 LP tokens for 63072000 (715ms)
 - (2) depositTokenBalanceBefore-depositTokenBalanceAfter = 400
 - (2) veTokenBalanceAfter-veTokenBalanceBefore = 291999.999999998131200000
- ✓ (2) account 1 should lock 400 LP tokens for 63072000 (523ms)
- ✓ set forfeit redirection address to account 3 (290ms)
 - (3) DepositId 1: 0.0
 - (3) DepositId 2: 0.0
- ✓ (3) print reward/yield tokens earned for deposit IDs 1 and 2 (69ms)
- ✓ upload 1,000 yield tokens into contract (218ms)
 - (4) DepositId 1: 333.333333333333210666
 - (4) DepositId 2: 666.666666666666421333
- ✓ (4) print reward/yield tokens earned for deposit IDs 1 and 2 (77ms)
 - (5) depositTokenBalanceAfter-depositTokenBalanceBefore = 200
 - (5) veTokenBalanceBefore-veTokenBalanceAfter = 145999.999999999065600000
 - (5) NOTE: rewardTokenBalance is for account 3
 - (5) rewardTokenBalanceAfter-rewardTokenBalanceBefore = 333.333333333333210666
- ✓ (5) account 1 should unstake ALL 200 LP tokens from deposit Id 1 (366ms)
- ✓ as non owner, FAIL to call setForfeitRedirectionAddressLocked, REVERT (107ms)
- ✓ as owner, call setForfeitRedirectionAddressLocked (66ms)
- ✓ as owner, FAIL TO setForfeitRedirection to 0x0 address, REVERT (66ms)

```
MaxDepositDuration Tests
```

```
Yield Token: Epoch EPOCH 18
```

```
Deposit Token: LP Token LP 18
```

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (252ms)
- ✓ mint 1,000,000 yieldTokens into account 0 (74ms)
 - (1) Max deposit duration: 63072000
- ✓ (1) print maximum deposit duration

- ✓ as owner, set maxDepositDuration to 63072000 seconds (66ms)
- ✓ account 1 should be unable to lock 200 LP tokens for 63072001 (217ms)
- ✓ account 1 should be unable to lock 200 LP tokens for 59 (185ms)
 - (4) depositTokenBalanceBefore–depositTokenBalanceAfter = 200
 - (4) veTokenBalanceAfter–veTokenBalanceBefore = 145999.999999999065600000
- ✓ account 1 should lock 200 LP tokens for 63072000 (303ms)
- ✓ non owner should be unable to maxDepositDuration to 31536000 seconds (92ms)
- ✓ as owner, set maxDepositDuration to 31536000 seconds (95ms)
- ✓ set next block.timestamp to 101% into account 0's deposit (deposit Id 1) (67ms)
 - (4) depositTokenBalanceAfter–depositTokenBalanceBefore = 200
 - (4) veTokenBalanceBefore–veTokenBalanceAfter = 145999.999999999065600000
- ✓ (4) account 1 should unstake ALL 200 LP tokens from deposit (242ms)
- ✓ as non owner, FAIL to call setMaxDepositDurationLocked, REVERT (89ms)
- ✓ as owner, FAIL to setMaxDepositDurationLocked to 315576001, REVERT (84ms)
- ✓ as owner, set setMaxDepositDurationLocked to 315576000
- ✓ as owner, call setMaxDepositDurationLocked (69ms)
- ✓ as owner, FAIL TO set maxDepositDuration to 100000 seconds, REVERT (68ms)

Admin: VeTokenAuthorised Tests

Yield Token: Epoch EPOCH 18

Deposit Token: LP Token LP 18

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (256ms)
- ✓ mint 1,000,000 yieldTokens into account 0 (65ms)

ClaimMultipleYield Tests

Yield Token: Epoch EPOCH 18

Deposit Token: LP Token LP 18

These tests are to ensure a user can claim yield from MULTIPLE deposits at the same time

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (206ms)
- ✓ mint 1,000,000 yieldTokens into account 0 (89ms)
 - (1) depositId = 1
- ✓ (1) account 0 should lock 100 LP tokens for 365 days (715ms)
 - (2) depositId = 2
- ✓ (2) account 0 should lock 200 LP tokens for 365 days (393ms)
 - (3) DepositId 1: 0.0
 - (3) DepositId 2: 0.0
- ✓ (3) print reward/yield tokens earned for deposit IDs 1 and 2 (57ms)
- ✓ upload 1,000 yield tokens into contract (204ms)
 - (4) DepositId 1: 333.333333333333320166
 - (4) DepositId 2: 666.666666666666640333
- ✓ (4) print reward/yield tokens earned for deposit IDs 1 and 2 (68ms)
 - (5) rewardTokenBalanceAfter–rewardTokenBalanceBefore = 999.99999999999960499
- ✓ (5) account 0 should claim all 999.99999999999960499 reward/yield tokens from deposit Ids 1 and 2 (286ms)
 - (6) DepositId 1: 0.0
 - (6) DepositId 2: 0.0
- ✓ (6) print reward tokens earned for deposit IDs 1 and 2 (52ms)
 - (7) rewardTokenBalanceAfter–rewardTokenBalanceBefore = 0
- ✓ (7) account 0 should claim 0 reward tokens from deposit Ids 1 and 2 (186ms)

DepositExtend Tests

Yield Token: Epoch EPOCH 18

Deposit Token: LP Token LP 18

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (195ms)
- ✓ mint 1,000,000 yieldTokens into account 0 (98ms)
 - (1) reward = 36499.999999997664
- ✓ (1) print veTokens minted for 100 depositTokens locked for 365 days (55ms)
- ✓ set max deposit duration to 365 days (76ms)
 - (2) depositTokenBalanceBefore–depositTokenBalanceAfter = 200
 - (2) veTokenBalanceAfter–veTokenBalanceBefore = 72999.999999995328
 - (2) depositId = 1
- ✓ (2) account 0 should lock 200 LP tokens for 365 days (454ms)
 - (3) depositTokenBalanceBefore–depositTokenBalanceAfter = 200
 - (3) veTokenBalanceAfter–veTokenBalanceBefore = 72999.999999995328
 - (3) depositId = 2
- ✓ (3) account 1 should lock 200 LP tokens for 365 days (476ms)
 - (4) depositTokenBalanceBefore–depositTokenBalanceAfter = 200
 - (4) veTokenBalanceAfter–veTokenBalanceBefore = 36499.99999999766400000
 - (4) depositId = 3
- ✓ (4) account 2 should lock 200 LP tokens for 182.5 days (342ms)
 - (5) DepositId 1: 0.0
 - (5) DepositId 2: 0.0

- (5) DepositId 3: 0.0
- ✓ (5) print reward/yield tokens earned for deposit IDs 1 and 2 (96ms)
- ✓ upload 1,000 yield tokens into contract (135ms)
 - (6) DepositId 1: 399.99999999999954999
 - (6) DepositId 2: 399.99999999999954999
 - (6) DepositId 3: 199.99999999999977499
- ✓ (6) print reward/yield tokens earned for deposit IDs 1 and 2 (93ms)
- ✓ account 2 should fail to extend for greater than max deposit duration, REVERT (98ms)
- ✓ expect reported deposit duration for deposit id 3 to be 15768000 (64ms)
 - (7) depositTokenBalanceAfter-depositTokenBalanceBefore = 0
 - (7) veTokenBalanceAfter-veTokenBalanceBefore = 36499.999999999766400000
- ✓ (7) account 2 should extend deposit lock by 182.5 days (233ms)
- ✓ expect reported deposit duration for deposit id 3 to be 31536000 (59ms)
- ✓ upload 1,000 yield tokens into contract (155ms)
 - (8) DepositId 1: 733.33333333333238666
 - (8) DepositId 2: 733.33333333333238666
 - (8) DepositId 3: 533.3333333333261165
- ✓ (8) print reward/yield tokens earned for deposit IDs 1 and 2 (82ms)
- ✓ expect rewards earned by deposit 3 to be 533.3333333333261165

EarlyUnstake: Full Tests

Yield Token: Epoch EPOCH 18

Deposit Token: LP Token LP 18

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (253ms)
- ✓ mint 1,000,000 yieldTokens into account 0 (86ms)
 - (1) reward = 36499.9999999997664
- ✓ (1) print veTokens minted for 100 depositTokens locked for 365 days (82ms)
 - (2) depositId = 1
- ✓ (2) account 0 should lock 100 LP tokens for 365 days (356ms)
 - (3) depositId = 2
- ✓ (3) account 1 should lock 200 LP tokens for 365 days (400ms)
 - (4) DepositId 1: 0.0
 - (4) DepositId 2: 0.0
- ✓ (4) print reward/yield tokens earned for deposit IDs 1 and 2 (73ms)
- ✓ upload 1,000 yield tokens into contract (131ms)
 - (5) DepositId 1: 333.3333333333320166
 - (5) DepositId 2: 666.66666666666640333
- ✓ (5) print reward/yield tokens earned for deposit IDs 1 and 2 (71ms)
- ✓ set next block.timestamp to 40% into account 0's deposit (deposit Id 1) (42ms)
 - (6) depositTokenBalanceAfter-depositTokenBalanceBefore = 100
 - (6) veTokenBalanceBefore-veTokenBalanceAfter = 36499.9999999997664
- ✓ (6) account 0 should unstake ALL 100 LP tokens from deposit (264ms)
 - (7) DepositId 1: 0.0
 - (7) DepositId 2: 999.9999999999923999
- ✓ (7) print reward/yield tokens earned for deposit IDs 1 and 2 (55ms)
- ✓ set next block.timestamp to 101% into account 1's deposit (deposit Id 2) (44ms)
 - (8) rewardTokenBalanceAfter-rewardTokenBalanceBefore = 999.9999999999923999
- ✓ (8) account 1 should claim all 999.9999999999923999 reward/yield tokens from deposit Id 2 (354ms)

EarlyUnstake: Partial Tests

Yield Token: Epoch EPOCH 18

Deposit Token: LP Token LP 18

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (270ms)
- ✓ mint 1,000,000 yieldTokens into account 0 (66ms)
 - (1) reward = 36499.9999999997664
- ✓ (1) print veTokens minted for 100 depositTokens locked for 365 days
 - (2) depositId = 1
- ✓ (2) account 0 should lock 100 LP tokens for 365 days (225ms)
 - (3) depositId = 2
- ✓ (3) account 1 should lock 200 LP tokens for 365 days (346ms)
 - (4) DepositId 1: 0.0
 - (4) DepositId 2: 0.0
- ✓ (4) print reward/yield tokens earned for deposit IDs 1 and 2 (47ms)
- ✓ upload 1,000 yield tokens into contract (221ms)
 - (5) DepositId 1: 333.3333333333320166
 - (5) DepositId 2: 666.66666666666640333
- ✓ (5) print reward/yield tokens earned for deposit IDs 1 and 2 (91ms)
- ✓ set next block.timestamp to 40% into account 0's deposit (deposit Id 1) (92ms)
 - (6) depositTokenBalanceAfter-depositTokenBalanceBefore = 50
 - (6) veTokenBalanceBefore-veTokenBalanceAfter = 18249.9999999998832
- ✓ (6) account 0 should unstake 50 LP tokens from deposit (829ms)
 - (7) DepositId 1: 199.9999999999977499

- (7) DepositId 2: 799.999999999999909999
- ✓ (7) print reward/yield tokens earned for deposit IDs 1 and 2 (134ms)
- ✓ set next block.timestamp to 101% into account 1's deposit (deposit Id 2) (69ms)
 - (8) rewardTokenBalanceAfter-rewardTokenBalanceBefore = 799.999999999999909999
- ✓ (8) account 1 should claim all 799.999999999999909999 reward/yield tokens from deposit Id 2 (531ms)

MaturedWithdrawFull Tests

Yield Token: Epoch EPOCH 18

Deposit Token: LP Token LP 18

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (317ms)
- ✓ mint 1,000,000 yieldTokens into account 0 (104ms)
 - (1) reward = 36499.9999999997664
- ✓ (1) print veTokens minted for 100 depositTokens locked for 365 days (105ms)
 - (2) depositId = 1
- ✓ (2) account 0 should lock 100 LP tokens for 365 days (362ms)
 - (3) depositId = 2
- ✓ (3) account 1 should lock 200 LP tokens for 365 days (421ms)
 - (4) DepositId 1: 0.0
 - (4) DepositId 2: 0.0
- ✓ (4) print reward/yield tokens earned for deposit IDs 1 and 2 (71ms)
- ✓ upload 1,000 yield tokens into contract (189ms)
 - (5) DepositId 1: 333.33333333333320166
 - (5) DepositId 2: 666.66666666666640333
- ✓ (5) print reward/yield tokens earned for deposit IDs 1 and 2 (65ms)
- ✓ account 0 should be unable to call withdraw, revert with DEPOSIT NOT MATURED (82ms)
- ✓ set next block.timestamp to 101% into account 0's deposit (deposit Id 1) (46ms)
- ✓ account 0 should be unable to call withdrawForfeit, revert with DEPOSIT IS MATURED (68ms)
 - (6) depositTokenBalanceAfter-depositTokenBalanceBefore = 100
 - (6) veTokenBalanceBefore-veTokenBalanceAfter = 36499.9999999997664
 - (6) rewardTokenBalanceAfter-rewardTokenBalanceBefore = 333.33333333333320166
 - (6) NOTE: Reward tokens increase because user MUST claim ALL rewards associated with deposit when
 - (6) NOTE: fully withdrawing.
- ✓ (6) account 0 should unstake ALL 100 LP tokens from deposit AND claim rewards (499ms)
 - (7) DepositId 1: 0.0
 - (7) DepositId 2: 666.66666666666640333
- ✓ (7) print reward/yield tokens earned for deposit IDs 1 and 2 (60ms)
 - (8) rewardTokenBalanceAfter-rewardTokenBalanceBefore = 666.66666666666640333
- ✓ (8) account 1 should claim all 666.66666666666640333 reward/yield tokens from deposit Id 2 (240ms)
- ✓ upload 5,000 yield tokens into contract (171ms)
 - (9) DepositId 1: 0.0
 - (9) DepositId 2: 4999.9999999999984999
- ✓ (9) print reward/yield tokens earned for deposit IDs 1 and 2 (76ms)
- ✓ account 1's deposit 2 should have 4999.9999999999984999 yield tokens to claim (39ms)
 - (10) rewardTokenBalanceAfter-rewardTokenBalanceBefore = 4999.9999999999984999
- ✓ (10) account 1 should claim all 4999.9999999999984999 reward/yield tokens from deposit Id 2 (156ms)
 - (11) rewardTokenBalanceAfter-rewardTokenBalanceBefore = 0
- ✓ (11) account 1 should claim and receive 0 yield tokens (147ms)

MaturedWithdrawPartial Tests

Yield Token: Epoch EPOCH 18

Deposit Token: LP Token LP 18

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (305ms)
- ✓ mint 1,000,000 yieldTokens into account 0 (171ms)
 - (1) reward = 36499.9999999997664
- ✓ (1) print veTokens minted for 100 depositTokens locked for 365 days (66ms)
 - (2) depositId = 1
- ✓ (2) account 0 should lock 100 LP tokens for 365 days (352ms)
 - (3) depositId = 2
- ✓ (3) account 1 should lock 200 LP tokens for 365 days (370ms)
 - (4) DepositId 1: 0.0
 - (4) DepositId 2: 0.0
- ✓ (4) print reward/yield tokens earned for deposit IDs 1 and 2 (84ms)
- ✓ upload 1,000 yield tokens into contract (161ms)
 - (5) DepositId 1: 333.33333333333320166
 - (5) DepositId 2: 666.66666666666640333
- ✓ (5) print reward/yield tokens earned for deposit IDs 1 and 2 (48ms)
- ✓ account 0 should be unable to call withdraw, revert with DEPOSIT NOT MATURED (82ms)
- ✓ set next block.timestamp to 101% into account 0's deposit (deposit Id 1) (58ms)
- ✓ account 0 should be unable to call withdrawForfeit, revert with DEPOSIT IS MATURED (75ms)
 - (6) depositTokenBalanceAfter-depositTokenBalanceBefore = 60
 - (6) veTokenBalanceBefore-veTokenBalanceAfter = 21899.99999999985984

- ✓ (6) account 0 should unstake 60 LP tokens from deposit (290ms)
 - (7) DepositId 1: 333.33333333333320166
 - (7) DepositId 2: 666.66666666666640333
- ✓ (7) print reward/yield tokens earned for deposit IDs 1 and 2 (108ms)
 - (8) rewardTokenBalanceAfter-rewardTokenBalanceBefore = 333.33333333333320166
- ✓ (8) account 0 should claim all 333.33333333333320166 reward/yield tokens from deposit Id 2 (143ms)
- ✓ upload 5,000 yield tokens into contract (142ms)
 - (9) DepositId 1: 833.333333333333266
 - (9) DepositId 2: 4833.3333333333306666
- ✓ (9) print reward/yield tokens earned for deposit IDs 1 and 2 (110ms)
- ✓ account 0's deposit 1 should have 833.333333333333266 yield tokens to claim

NonTransferable Tests

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (270ms)
 - (2) deposit id = 1
 - (2) account 0 veTokenBalance = 364999.99999997664
- ✓ (2) account 0 should deposit 1000 depositTokens for 365 days (404ms)
- ✓ 'transfer' of 1 veToken from account 0 to account 1 should revert with NON TRANSFERABLE (80ms)
- ✓ calling 'transferFrom' as account 1 to transfer 1 veToken from account 0 to account 1 should revert with NON TRANSFERABLE (59ms)
 - ✓ calling 'transferFrom' as account 3 to transfer 1 veToken from account 0 to account 1 should revert with NON TRANSFERABLE (70ms)
 - ✓ as non owner (account 1), set account 3 to authorised should revert (113ms)
 - ✓ as owner, set account 3 to authorised (146ms)
 - ✓ calling 'transferFrom' as account 3 to transfer 1 veToken from account 0 to account 1 should REVERT with ERC20: insufficient allowance (69ms)
 - ✓ account 0 should approve account 3 to spend up to 1 veToken (75ms)
 - (3) account 0 token balance before: 364999.99999997664
 - (3) account 0 token balance after: 364998.99999997664
 - (3) account 1 token balance before: 0.0
 - (3) account 1 token balance after: 1.0
 - ✓ (3) calling 'transferFrom' as account 3 to transfer 1 veToken from account 0 to account 1 should succeed (157ms)
 - ✓ calling 'transferFrom' as account 3 to transfer 1 veToken from account 0 to account 1 should REVERT with ERC20: insufficient allowance (82ms)
 - ✓ account 0 should approve account 3 to spend up to 1 veToken (70ms)
 - (3) account 0 token balance before: 364998.99999997664
 - (3) account 0 token balance after: 364997.99999997664
 - (3) account 3 token balance before: 0.0
 - (3) account 3 token balance after: 1.0
 - ✓ (3) calling 'transferFrom' as account 3 to transfer 1 veToken from account 0 to account 3 should succeed (173ms)
 - ✓ calling 'transfer' to transfer 1 veToken from account 3 to account 2 should be succeed (78ms)
 - ✓ should ensure account 5 is NOT authorised to transfer veTokens
 - ✓ as owner, set account 5 to be authorised to transfer veTokens (69ms)
 - ✓ should ensure account 5 IS authorised to transfer veTokens
 - ✓ as owner, set account 5 to NOT be authorised to transfer veTokens (52ms)
 - ✓ should ensure account 5 is NOT authorised to transfer veTokens
 - ✓ as non owner, FAIL to call setAuthorisedLocked, REVERT (57ms)
 - ✓ as owner, call setAuthorisedLocked (56ms)
 - ✓ as owner, FAIL TO set account 3 to non-authorised, REVERT (52ms)

PaybackYield Partial Tests

Yield Token: Epoch EPOCH 18

Deposit Token: LP Token LP 18

These tests are to ensure that even if a user has claimed rewards, upon forfeiting FULLY, they end up paying back the reward using reward tokens from their wallet

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (195ms)
- ✓ mint 1,000,000 yieldTokens into account 0 (77ms)
 - (1) depositId = 1
- ✓ (1) account 0 should lock 100 LP tokens for 365 days (320ms)
 - (2) depositId = 2
- ✓ (2) account 0 should lock 200 LP tokens for 365 days (371ms)
 - (3) DepositId 1: 0.0
 - (3) DepositId 2: 0.0
- ✓ (3) print reward/yield tokens earned for deposit IDs 1 and 2 (71ms)
- ✓ upload 1,000 yield tokens into contract (233ms)
 - (4) DepositId 1: 333.33333333333320166
 - (4) DepositId 2: 666.66666666666640333
- ✓ (4) print reward/yield tokens earned for deposit IDs 1 and 2 (74ms)
 - (5) rewardTokenBalanceAfter-rewardTokenBalanceBefore = 333.33333333333320166
- ✓ (5) account 0 should claim all 333.33333333333320166 reward/yield tokens from deposit Id 1 (171ms)

- (6) DepositId 1: 0.0
- (6) DepositId 2: 666.666666666666640333
- ✓ (6) print reward/yield tokens earned for deposit IDs 1 and 2 (70ms)
- ✓ set next block.timestamp to 40% into account 0's deposit (deposit Id 1) (50ms)
 - (7) depositTokenBalanceAfter-depositTokenBalanceBefore = 100
 - (7) veTokenBalanceBefore-veTokenBalanceAfter = 36499.9999999997664
 - (7) rewardTokenBalanceBefore-rewardTokenBalanceAfter = 333.33333333333320166
 - (7) NOTE: User now has 50 LP tokens staked which is 0% of the total staking power
 - (7) NOTE: which means of the forfeited amount, 0 would be expected to go back to this deposit
- ✓ (7) account 0 should unstake ALL 100 LP tokens from depositId 1 (454ms)
 - (8) DepositId 1: 0.0
 - (8) DepositId 2: 999.99999999999923999
- ✓ (8) print reward/yield tokens earned for deposit IDs 1 and 2 (87ms)
- ✓ upload 1,000 yield tokens into contract (177ms)
 - (9) DepositId 1: 0.0
 - (9) DepositId 2: 1999.9999999999920999
- ✓ (9) print reward/yield tokens earned for deposit IDs 1 and 2 (53ms)

PaybackYield Partial Tests

Yield Token: Epoch EPOCH 18

Deposit Token: LP Token LP 18

This is a duplicate of the Full tests except these cases are to demonstrate that duration into the deposit/stake is irrelevant when it comes to reward forfeits

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (184ms)
- ✓ mint 1,000,000 yieldTokens into account 0 (89ms)
 - (1) depositId = 1
- ✓ (1) account 0 should lock 100 LP tokens for 365 days (314ms)
 - (2) depositId = 2
- ✓ (2) account 0 should lock 200 LP tokens for 365 days (391ms)
 - (3) DepositId 1: 0.0
 - (3) DepositId 2: 0.0
- ✓ (3) print reward/yield tokens earned for deposit IDs 1 and 2 (93ms)
- ✓ upload 1,000 yield tokens into contract (165ms)
 - (4) DepositId 1: 333.33333333333320166
 - (4) DepositId 2: 666.666666666666640333
- ✓ (4) print reward/yield tokens earned for deposit IDs 1 and 2 (105ms)
 - (5) rewardTokenBalanceAfter-rewardTokenBalanceBefore = 333.33333333333320166
- ✓ (5) account 0 should claim all 333.33333333333320166 reward/yield tokens from deposit Id 1 (190ms)
 - (6) DepositId 1: 0.0
 - (6) DepositId 2: 666.666666666666640333
- ✓ (6) print reward/yield tokens earned for deposit IDs 1 and 2 (84ms)
- ✓ set next block.timestamp to 75% into account 0's deposit (deposit Id 1) (73ms)
 - (7) depositTokenBalanceAfter-depositTokenBalanceBefore = 100
 - (7) veTokenBalanceBefore-veTokenBalanceAfter = 36499.9999999997664
 - (7) rewardTokenBalanceBefore-rewardTokenBalanceAfter = 333.33333333333320166
 - (7) NOTE: User now has 50 LP tokens staked which is 0% of the total staking power
 - (7) NOTE: which means of the forfeited amount, 0 would be expected to go back to this deposit
- ✓ (7) account 0 should unstake ALL 100 LP tokens from depositId 1 (753ms)
 - (8) DepositId 1: 0.0
 - (8) DepositId 2: 999.99999999999923999
- ✓ (8) print reward/yield tokens earned for deposit IDs 1 and 2 (106ms)
- ✓ upload 1,000 yield tokens into contract (162ms)
 - (9) DepositId 1: 0.0
 - (9) DepositId 2: 1999.9999999999920999
- ✓ (9) print reward/yield tokens earned for deposit IDs 1 and 2 (80ms)

PaybackYield Partial Tests

Yield Token: Epoch EPOCH 18

Deposit Token: LP Token LP 18

These tests are to ensure that even if a user has claimed rewards, upon forfeiting PARTIALLY, they end up paying back the reward using reward tokens from their wallet

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (241ms)
- ✓ mint 1,000,000 yieldTokens into account 0 (99ms)
 - (1) depositId = 1
- ✓ (1) account 0 should lock 100 LP tokens for 365 days (516ms)
 - (2) depositId = 2
- ✓ (2) account 0 should lock 200 LP tokens for 365 days (287ms)
 - (3) DepositId 1: 0.0
 - (3) DepositId 2: 0.0
- ✓ (3) print reward/yield tokens earned for deposit IDs 1 and 2 (58ms)
- ✓ upload 1,000 yield tokens into contract (143ms)
 - (4) DepositId 1: 333.33333333333320166

- (4) DepositId 2: 666.666666666666640333
- ✓ (4) print reward/yield tokens earned for deposit IDs 1 and 2 (87ms)
 - (5) rewardTokenBalanceAfter-rewardTokenBalanceBefore = 333.33333333333320166
- ✓ (5) account 0 should claim all 333.33333333333320166 reward/yield tokens from deposit Id 1 (157ms)
 - (6) DepositId 1: 0.0
 - (6) DepositId 2: 666.666666666666640333
- ✓ (6) print reward/yield tokens earned for deposit IDs 1 and 2 (81ms)
- ✓ set next block.timestamp to 40% into account 0's deposit (deposit Id 1) (61ms)
 - (7) depositTokenBalanceAfter-depositTokenBalanceBefore = 50
 - (7) veTokenBalanceBefore-veTokenBalanceAfter = 18249.9999999998832
 - (7) rewardTokenBalanceBefore-rewardTokenBalanceAfter = 166.66666666666660083
 - (7) NOTE: User now has 50 LP tokens staked which is 20% of the total staking power
 - (7) NOTE: which means of the forfeited amount, ~33.33 would be expected to go back to this

deposit

- ✓ (7) account 0 should unstake 50 LP tokens from depositId 1 (454ms)
 - (8) DepositId 1: 33.33333333333317416
 - (8) DepositId 2: 799.99999999999909999
- ✓ (8) print reward/yield tokens earned for deposit IDs 1 and 2 (75ms)
- ✓ upload 1,000 yield tokens into contract (167ms)
 - (9) DepositId 1: 233.33333333333313166
 - (9) DepositId 2: 1599.99999999999892999
- ✓ (9) print reward/yield tokens earned for deposit IDs 1 and 2 (255ms)

PaybackYield Partial2 Tests

Yield Token: Epoch EPOCH 18

Deposit Token: LP Token LP 18

This is a duplicate of the Partial tests except these cases are to demonstrate that duration into the deposit/stake is irrelevant when it comes to reward forfeits

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (1263ms)
- ✓ mint 1,000,000 yieldTokens into account 0 (236ms)
 - (1) depositId = 1
- ✓ (1) account 0 should lock 100 LP tokens for 365 days (375ms)
 - (2) depositId = 2
- ✓ (2) account 0 should lock 200 LP tokens for 365 days (408ms)
 - (3) DepositId 1: 0.0
 - (3) DepositId 2: 0.0
- ✓ (3) print reward/yield tokens earned for deposit IDs 1 and 2 (114ms)
- ✓ upload 1,000 yield tokens into contract (257ms)
 - (4) DepositId 1: 333.33333333333320166
 - (4) DepositId 2: 666.666666666666640333
- ✓ (4) print reward/yield tokens earned for deposit IDs 1 and 2 (76ms)
 - (5) rewardTokenBalanceAfter-rewardTokenBalanceBefore = 333.33333333333320166
- ✓ (5) account 0 should claim all 333.33333333333320166 reward/yield tokens from deposit Id 1 (237ms)
 - (6) DepositId 1: 0.0
 - (6) DepositId 2: 666.666666666666640333
- ✓ (6) print reward/yield tokens earned for deposit IDs 1 and 2 (88ms)
- ✓ set next block.timestamp to 75% into account 0's deposit (deposit Id 1) (72ms)
 - (7) depositTokenBalanceAfter-depositTokenBalanceBefore = 50
 - (7) veTokenBalanceBefore-veTokenBalanceAfter = 18249.9999999998832
 - (7) rewardTokenBalanceBefore-rewardTokenBalanceAfter = 166.66666666666660083
 - (7) NOTE: User now has 50 LP tokens staked which is 20% of the total staking power
 - (7) NOTE: which means of the forfeited amount, ~33.33 would be expected to go back to this

deposit

- ✓ (7) account 0 should unstake 50 LP tokens from depositId 1 (423ms)
 - (8) DepositId 1: 33.33333333333317416
 - (8) DepositId 2: 799.99999999999909999
- ✓ (8) print reward/yield tokens earned for deposit IDs 1 and 2 (79ms)
- ✓ upload 1,000 yield tokens into contract (241ms)
 - (9) DepositId 1: 233.33333333333313166
 - (9) DepositId 2: 1599.99999999999892999
- ✓ (9) print reward/yield tokens earned for deposit IDs 1 and 2 (93ms)

SimpleRewardTest Tests

Stack traces engine could not be initialized. Run Hardhat with --verbose to learn more.

Yield Token: Epoch EPOCH 18

Deposit Token: LP Token LP 18

The goal of this test is to ensure same token amounts with different deposit lock durations result in a different number of reward tokens earned

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (235ms)
- ✓ mint 1,000,000 yieldTokens into account 0 (78ms)
 - (1) reward = 36499.9999999997664
- ✓ (1) print veTokens minted for 100 depositTokens locked for 365 days (66ms)

- (2) depositId = 1
- ✓ (2) account 0 should lock 100 LP tokens for 365 days (416ms)
 - (3) depositId = 2
- ✓ (3) account 1 should lock 100 LP tokens for 730 days (429ms)
 - (3.1) depositId = 3
- ✓ (3.1) account 2 should lock 500 LP tokens for 730 days (364ms)
 - (4) DepositId 1: 0.0
 - (4) DepositId 2: 0.0
 - (4) DepositId 3: 0.0
 - (4) DepositId 1 (stakingPower): 3649999999999766400000
 - (4) DepositId 2 (stakingPower): 7299999999999532800000
 - (4) DepositId 3 (stakingPower): 36499999999997664000000
- ✓ (4) print reward/yield tokens earned for deposit IDs 1 and 2 (269ms)
- ✓ upload 1,000 yield tokens into contract (180ms)
 - (5) DepositId 1: 76.923076923076903192
 - (5) DepositId 2: 153.846153846153806384
 - (5) DepositId 3: 769.230769230769031923
 - (5) DepositId 1 (stakingPower): 3649999999999766400000
 - (5) DepositId 2 (stakingPower): 7299999999999532800000
 - (5) DepositId 3 (stakingPower): 36499999999997664000000
- ✓ (5) print reward/yield tokens earned for deposit IDs 1 and 2 (194ms)
- ✓ set next block.timestamp to 40% into account 0's deposit (deposit Id 1) (77ms)
 - (6) depositTokenBalanceAfter-depositTokenBalanceBefore = 100
 - (6) veTokenBalanceBefore-veTokenBalanceAfter = 36499.9999999997664
- ✓ (6) account 0 should unstake ALL 100 LP tokens from deposit (544ms)
 - (7) DepositId 1: 0.0
 - (7) DepositId 2: 166.666666666666605333
 - (7) DepositId 3: 833.333333333333026666
 - (7) DepositId 1 (stakingPower): 0
 - (7) DepositId 2 (stakingPower): 7299999999999532800000
 - (7) DepositId 3 (stakingPower): 36499999999997664000000
- ✓ (7) print reward/yield tokens earned for deposit IDs 1 and 2 (183ms)

TransferDeposit Tests

Stack traces engine could not be initialized. Run Hardhat with --verbose to learn more.

Yield Token: Epoch EPOCH 18

Deposit Token: LP Token LP 18

- ✓ mint 1,000 depositTokens into accounts 1, 2 and 3 (1259ms)
- ✓ mint 1,000,000 yieldTokens into account 0 (429ms)
- ✓ as owner, set maxDepositDuration to 63072000 seconds (221ms)
 - (1) depositId: 1
 - (1) account 1 veToken balance: 145999.9999999990656
- ✓ (1) account 1 should lock 200 LP tokens for 63072000 (520ms)
- ✓ account 0 should be unable to transfer deposit ownership of deposit Id 1, REVERT with NOT OWNER (83ms)
 - (1) account 0 veToken balance: 0.0
 - (1) account 1 veToken balance: 145999.9999999990656
 - Ownership transferred
 - (1) account 0 veToken balance: 145999.9999999990656
 - (1) account 1 veToken balance: 0.0
- ✓ account 1 should transfer deposit ownership of deposit Id 1 to account 0 (177ms)
- ✓ account 1 should be unable to call withdrawForfeit, withdraw, transferDepositOwnership and claimReward (yield) (441ms)
- ✓ set next block.timestamp to 40% into account 0's deposit (deposit Id 1) (154ms)
- ✓ account 0 should be able to call withdrawForfeit for deposit id 1 (static call) (166ms)
- ✓ set next block.timestamp to 101% into account 0's deposit (deposit Id 1) (87ms)
- ✓ account 0 should be able to call withdraw for deposit id 1 (static call) (150ms)
- ✓ account 0 should be able to call transferDepositOwnership for deposit id 1 (static call) (169ms)
- ✓ account 0 should be able to call claimReward for deposit id 1 (static call) (93ms)

veTokenQuotes Tests

Stack traces engine could not be initialized. Run Hardhat with --verbose to learn more.

Yield Token: Epoch EPOCH 18

Deposit Token: LP Token LP 18

These tests are to ensure the correct number of veTokens are minted given a specific token amount and duration. This does NOT test maximum deposit durations

- (1) Max deposit duration: 63072000
- ✓ (1) print maximum deposit duration
- ✓ quoting 1 depositToken for 365 days should mint 364.99999999997664 veTokens (77ms)
- ✓ quoting 100 depositToken for 365 days should mint 36499.9999999997664 veTokens (53ms)
- ✓ quoting 1 depositToken for 182.5 days should mint 18249.9999999998832 veTokens (42ms)
- ✓ quoting 1 depositToken for 1 days should mint 0.9999999999993600 veTokens (40ms)

228 passing (2m)

Code Coverage

We followed the README, and ran the following command:

```
pnpm coverage
```

The test suite achieves 100% branch coverage. We would like to note that branch coverage is not a metric for test quality and recommend [mutation testing](#) in addition.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
Vepoch.sol	100	100	100	100	
All files	100	100	100	100	

Changelog

- 2023-10-30 - Initial Report
- 2023-11-03 - Final 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' 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.



Quantstamp