Pendle V2 - Liquidity Mining

Audit Report

Prepared by Christoph Michel

July 29, 2022.

Contents

1 Introduction

Pendle Finance is a yield tokenisation and yield trading protocol. This audit is about Pendle's liquidity mining and vested Pendle (`vePendle`) implementations. The documentation for auditors can be found at https://pendle.notion.site.

1.1 Scope of Work

The auditors were provided with a GitHub repository at commit hash 6b7ec5e. The scope of the audit is the liquidity mining contracts.

The task was to audit the contracts, consisting of the following files with their `sha1` hashes:

File	SHA1
CelerReceiverUpg.sol	`79d1665464f595da56d0aeb9f7efebb6a7a845e4`
CelerSenderUpg.sol	`2924c2a7aac70f8c343e99ee8738a8b7748ed72f`
PendleGaugeControllerBaseUpg.sol	`951ba01976051e2aa1629cc15333064037b6bd15`
PendleGaugeControllerMainchainUpg.sol	`7a862797f0340113bb713790ec925c6f034fd007`
PendleGaugeControllerSidechainUpg.sol	`a43662316d99fdf3dced90119f5374ddbe6af323`
PendleGauge.sol	`581266d7cff588bda8d41f7c0acdb810a7821d0c`
PendleVotingControllerUpg.sol	`61738433193e8df7bc3ee81ead99f9177a139c43`
VotingControllerStorageUpg.sol	`f2f9fe4c551d16ddb7577deae2e03c20ecf655d1`
VotingEscrowPendleMainchain.sol	`b2bc694b5b78cb83366e580f9a343e471dabb97b`
VotingEscrowPendleSidechain.sol	`83c41c548cedc8670e66fd6259c0a79360c65559`
VotingEscrowTokenBase.sol	`7aacd7495a7b06f0644af28deadc41f620f84397`
VeBalanceLib.sol	`f2b42e687c335049ae8a1f54cb75c041978f74a4`
VeHistoryLib.sol	`d5ef97c5ee33d7417d5932c38429648bf899e4b6`

The rest of the repository was out of the scope of the audit.

1.2 Security Assessment Methodology

The smart contract's code is scanned both manually and automatically for known vulnerabilities and logic errors that can lead to potential security threats. The conformity of requirements (e.g., specifications, documentation, White Paper) is reviewed as well on a consistent basis.

1.3 Auditors

Christoph Michel

2 Severity Levels

We assign a risk score to the severity of a vulnerability or security issue. For this purpose, we use 4 *severity levels* namely:

MINOR

Minor issues are generally subjective in nature or potentially associated with topics like “best practices” or “readability”. As a rule, minor issues do not indicate an actual problem or bug in the code. The maintainers should use their own judgment as to whether addressing these issues will improve the codebase.

LOW

Low-severity issues are generally objective in nature but do not represent any actual bugs or security problems. These issues should be addressed unless there is a clear reason not to.

MEDIUM

Medium-severity issues are bugs or vulnerabilities. These issues may not be directly exploitable or may require certain conditions in order to be exploited. If unaddressed, these issues are likely to cause problems with the operation of the contract or lead to situations that make the system exploitable.

HIGH

High-severity issues are directly exploitable bugs or security vulnerabilities. If unaddressed, these issues are likely or guaranteed to cause major problems or, ultimately, a full failure in the operations of the contract.

3 Discovered issues

3.1 Voting might temporarily cross the 100% threshold and revert (medium)

Context: PendleVotingControllerUpg.sol#L69

The `vote` function splits the votes into vote reductions and additions. The reductions should be processed first such that there are no reverts when the total vote weight temporarily crosses 100%.

However, the additions are processed first and the function might therefore revert.

> Note that this splitting of votes leads to unexpected behavior when duplicate pool parameters are specified. The user might expect the last weight of the `weights` array to be applied for the duplicate pool but it might not if it is a reduction.

Recommendation

Swap the iterations such that the vote weight reductions (`oldWeight[i] > weights[i]`) are performed first.

3.2 Bridge DoS by depleting `CelerSenderUpg` balance (medium)

Context: PendleVotingControllerUpg.sol#L144

The `CelerSenderUpg` computes a `fee` that Celer charges for broadcasting the message to other chains. It's possible for users to trigger broadcasts either through:

- repeatedly broadcasting the current week's voting results
- repeatedly calling `broadcastUserPosition` with repeated `chainIds` to trigger many messages and fee payments in a single call

Once the contract's balance is depleted, no more messages will be broadcast.

Recommendation

Ensure that the contract has a large enough balance to handle broadcasts and come up with a way to stop griefers from depleting this balance. While broadcasting the current week's voting results could use a flag to indicate that they have already been broadcasted and revert, preventing mass broadcasting of user balances is non-trivial.

3.3 Gauge controller must have large enough reward balance (low)

Context: PendleGaugeControllerBaseUpg.sol#L72

The `PendleGaugeControllerBaseUpg` must be manually funded through `fundPendle` calls to pay out the rewards. It could be that rewards should be paid out (through `epochRewardReceived`) but the contract was not funded with a large enough balance and claiming the rewards will fail.

Recommendation

Ensure that the `PendleGaugeControllerBaseUpg` always has a large enough balance to cover rewards for a certain period for all markets.

3.4 Pools (markets) on different chains must not have the same address (informational)

Context: VotingControllerStorageUpg.sol#L138

The `VotingControllerStorageUpg._addPool` function globally disallows duplicate pool (market) addresses. This means, the market factory must be deployed at different address on sidechains to derive a different `create2` market address in case any market is created on a sidechain with the same parameters as on another chain.

Recommendation

Ensure the market addresses are unique even across sidechains when creating markets. Either through deploying the market *factory* at different addresses on all chains, or using the chain ID as the `salt` `create2` parameter that is currently unused.

3.5 Miscellaneous (minor)

- The VotingEscrowPendleSidechain.delegatorOf mapping is used to boost rewards on sidechains from a delegator contract address on mainnet. It's possible that a single delegator delegates to multiple receivers, all of which will inherit the delegator's mainnet balance. Ensure that a delegator can only delegate to a single receiver when governance calls the `setDelegatorFor` function.
- Gas: VotingEscrowPendleSidechain.balanceOf could use the cached `delegator` value for the `if` condition.
- `broadcastResults` only broadcasts the current week. Missed weeks cannot be broadcast by users and governance needs to force-broadcast any missed weeks which leaves room for error as the voting results are specified manually.
- Consider renaming `VotingControllerStorageUpg.chainPools` to `activeChainPools` as inactive pools are removed from it.
- VotingEscrowPendleMainchain.increaseLockPosition reverts with an underflow if the `newExpiry` is before the existing `expiry`. Consider reverting with a more specific error message in this case.
- VotingEscrowPendleMainchain._applySlopeChange could return cached `(supply, wTime)` instead of reading `lastSlopeChangeAppliedAt` from storage again.
- VotingEscrowPendleMainchain._broadcast encodes `wTime` as a `uint256` type but `_executeMessage` decodes it as a `uint128`. This does not lead to issues as `abi.encode` pads all values to 32 bytes and `wTime` should never exceed a `uint128`. We still recommend encoding it as a `uint128` such that the encoding and decoding types match.

4 Conclusion

A potential denial of service issue with sending cross-chain updates has been found. No other major issues have been found in the liquidity mining contracts. We'd like to stress that the reward system requires frequent updates to be triggered to work accurately, both to accurately represent the decaying vePendle LP reward bonus for markets, as well as for broadcasting market reward results and vePendle user balance changes to sidechains. Some of these actions are supposed to be triggered by users themselves and the game-theoretic incentives are described in WhitePaper section 6.3 but they rely on simplifying assumptions such as "assuming gas is not a concern". The incentives might not play out as stated in practice.

Overall, the documentation and the codebase are of high quality. No judgement can be made on the test suite as it was not available to the auditors.

Disclaimer

This audit is based on the scope and snapshot of the code mentioned in the introduction. The contracts used in a production environment may differ drastically. Neither did this audit verify any deployment steps or multi-signature wallet setups. Audits cannot provide a guarantee that all vulnerabilities have been found, nor might all found vulnerabilities be completely mitigated by the project team. An audit is not an endorsement of the project or the team, nor guarantees its security. No third party should rely on the audit in any way, including for the purpose of making any decisions about investing in the project.