Least Authority

PRIVACY MATTERS

Pendle Protocol Smart Contracts

Security Audit Report

Pendle Finance

Final Report Version: 4 June 2021

Table of Contents

Overview

Background

Pendle Finance requested that Least Authority perform a security audit of the Pendle protocol smart contracts.

The Pendle protocol leverages the base lending layer created by other Decentralized Finance (DeFi) protocols (i.e. Aave and Compound) by separating the future cash flow from the lending protocols' yield tokens and tokenizing that cash flow. As a result, this allows future yield to be traded without impacting ownership of the underlying asset.

The following components are considered in-scope:

- Yield Tokenization
- Future yield trading via Automated Market Makers (AMM)

Project Dates

- **March 2 - April 13**: Code review *(Completed)*
- **April 16**: Delivery of Initial Audit Report *(Completed)*
- **May 20 - 21:** Verification *(Completed)*
- **May 24:** Delivery of Final Audit Report *(Completed)*
- **June 4**: Delivery of Updated Final Audit Report *(Completed)*

Review Team

- Dominc Tarr, Security Researcher and Engineer
- Nathan Ginnever, Security Researcher and Engineer
- Rai Yang, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Pendle protocol Smart Contracts followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

The following code repositories were considered in-scope for the review:

- Pendle protocol Smart Contracts: https://github.com/benchmark-finance/contracts/tree/5116f44cb0828d74b77ebfb14f394d61103112bb

The following files were considered out of scope for the review:

- Core
  - `PendleGovernance.sol`
  - `PendleTreasury.sol`
- Interfaces
  - `IPendleGovernance.sol`
  - `IPendleTreasury.sol`
- Periphery
  - `Timelock.sol`

Specifically, we examined the Git revisions for our initial review:

`5116f44cb0828d74b77ebfb14f394d61103112bb`

For the verification, we examined the Git revision:

`3380ebff814ae4a5e1241bfa5ca39036754e7160`

For the review, this repository was cloned for use during the audit and for reference in this report:

https://github.com/LeastAuthority/Benchmark-Pendle-Smart-Contracts

All file references in this document use Unix-style paths relative to the project's root directory.

Supporting Documentation

The following documentation was available to the review team:

- README.md: https://github.com/benchmark-finance/contracts/blob/audit_quotation/README.md
- Pendle Technical Specification (pdf shared with Least Authority via Telegram on February 26, 2021)
- Pendle AMM Design (pdf shared with Least Authority via Telegram on March 2, 2021)
- Audit Change Log: https://gist.github.com/mrenoon/746d1a9284cdedab07b8f6cf6e62463c

In addition, this audit report references the following documents:

- Article, "Missing return value bug — At least 130 tokens affected": https://medium.com/coinmonks/missing-return-value-bug-at-least-130-tokens-affected-d67bf08521ca
- Article, "Improving front running resistance of x*y=k market makers": https://ethresear.ch/t/improving-front-running-resistance-of-x-y-k-market-makers/1281
- K. Qin, L. Zhou, B. Livshits, A. Gervais, 2021, "Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit." *arXiv* 2003.03810v4 [QZL+21]

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the contracts;
- Potential misuse and gaming of the smart contracts;
- Attacks that impacts funds, such as the draining or the manipulation of funds;
- Mismanagement of funds via transactions;
- Economic incentives: ensure token economics (monetary incentives to punish bad behavior and reward good behavior) are included and functional;
- Denial of Service (DoS) and other security exploits that would impact the contracts intended use or disrupt the execution of the contract;
- Vulnerabilities in the smart contracts code;
- Protection against malicious attacks and other ways to exploit contracts;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The Pendle protocol offers liquidity providers the tokenization of future yields from a yield generating asset into the protocol token XYT. Liquidity providers are then able to provide liquidity to Pendle's time-decaying AMMs, allowing traders to take positions on the future yield. In the process, liquidity providers earn fees and incentives and provide liquidity to the pools.

System Design

The Pendle protocol is adapted from and builds on the base lending layer created by Aave and Compound, which are commonly used, regularly audited, and well maintained DeFi protocols. As previously noted, the Pendle protocol has been designed to provide two primary functionalities. First, the Pendle protocol provides yield tokenization and, second, it enables the trading of future yield token through AMMs.

Pendle AMM and Impermanent Loss

The Pendle team has implemented an AMM variant based on Balancer protocol's constant mean function, which is intended to mitigate time-dependent impermanent loss for token classes with a time decay model (i.e. XYT). This AMM model is intended to reduce the impact of time decaying tokens, which lose value over time as the token contract approaches maturity, by reducing volatility on tokens with lowering its weight over time (shift of the price curve). This is intended to reduce the risk of impermanent loss on the more price volatile assets in the pair and ultimately reduce barriers for liquidity providers. The Pendle protocol AMM model has been modified such that the price curve shift is responsive to change in time, implemented by changing the weight of the token pair in the constant product model. In particular, at time = 0, the weight of both tokens is equal and initiated at 0.5. Overtime, the weight of XYT decreases from 0.5 to 0 while the weight of the non-time decaying token increases from 0.5 to 1.

While we identified no security vulnerabilities with Pendle's AMM variant, the system design and code is extremely complex, which may result in hidden errors that may lead to critical security issues. As a result, we strongly recommend reducing the complexity of the system design and coded implementation. In addition, we suggest that additional and regular security audits of the Pendle protocol be conducted by different teams (Suggestion 6).

Governance Model

The Pendle protocol governance model was out of scope for our security review. It is important to note, however, that governance will be represented by an `M` of `N` multi-signature smart contract for Pendle's initial launch. This is meant as an interim solution until the Pendle team completes the implementation of a Pendle governance DAO smart contract, based on Compound's DAO implementation. According to Pendle's *Technical Specification*, the Pendle protocol will ultimately implement a DAO that will replace the multi-signature, in order to provide token holders voting and delegating rights. We strongly suggest that both the multi-signature interim solution and the final DAO implementations be followed up with a security review to investigate and analyze the design and functionality for potential security vulnerabilities (Suggestion 7).

Gas Expensive Computations

We identified several gas expensive computations, which affect liquidity providers by increasing gas costs. This has the potential to increase gas costs prohibitively during times of high network congestion. For example, the function `claimLpInterests`, which is implemented in a loop, may cause the block of the function call or the gas cost of the transaction to be greater than the amount claimed. As a result, we recommend limiting the number of expiries from which a user can claim interest or implementing a method for users to claim interest from specific expiries (Issue B).

Types of Attacks

DeFi smart contracts on the Ethereum blockchain are inherently vulnerable to flash loan attacks [QZL+21] and sandwich attacks, resulting in the price manipulation of underlying assets in liquidity pools. Pendle implemented curve shift as the first transaction of the block in order to reduce the profit of flashloan and sandwich attacks due to curve shift, however, traditional sandwich attacks may still occur. There is currently no known remediation for these types of vulnerabilities, however, we recommend that the Pendle team stay informed of the latest research and conduct further investigation into the exposure to these types of attack and possible mitigations.

Consideration of Security

The Pendle team has demonstrated a consideration for security in the design of the Pendle protocol. This is demonstrated by the modifiers limiting access to sensitive functions (e.g. fee changes, add/remove liquidity) and the reentrancy guards implemented in all cases where the liquidity provider is not whitelisted. However, as previously noted, the Pendle protocol is characterized by a high degree of complexity. The smart contracts are expansive and near the limits of what can be reasonably deployed. Systems with complex inheritance are particularly vulnerable due to a large attack surface resulting in errors hidden by the complexity, which can be difficult to detect. The Pendle team has stated that the complexity is necessary to implement the required features, however, we strongly recommend reducing the complexity of the system, which increases the ability for auditors to review the code more easily and reduces overall risk to the system (Suggestion 6).

Code Quality

The Pendle protocol code base is well written, organized, and logically structured. The code adheres to the Solidity style guide and best practices, and implements up-to-date compilers, which made the code easy to navigate.

The code base includes some test coverage, however, the existing tests are not exhaustive and do not comprehensively capture all failure scenarios, which would aid in identifying potential edge cases and errors. In addition, executing tests for failure cases can help determine if error conditions function as expected. We recommend expanding test coverage to include all error cases (Suggestion 3).

Documentation

The Pendle protocol code base is well documented and the available documentation, including the *Technical Specification* and the *Pendle AMM Design* document, sufficiently describes the intended functionality of the system. The *Technical Specification* includes a Trust Model (Section 6), which is particularly useful in evaluating the security considerations of the system. Furthermore, it includes important detailed information on the theory and application of the Curve Shifting Algorithm (Section 4.5.2) that the Pendle team has implemented. Additionally, we found the code base to be generally well commented, which facilitates a clear understanding of the intended functionality of the code.

The Pendle protocol's detailed documentation of the system design and components demonstrates a thorough and organized approach to both development and security. We commend the Pendle team's efforts to provide thorough documentation, which is a notable and important step in security due diligence.

Scope

We found that the scope of the security audit for the current implementation was sufficient and included all security-critical components, which allowed a comprehensive security review of the Pendle protocol. Furthermore, our team did not identify any third-party dependency concerns.

However, we strongly recommend that the governance model, which was out of scope for our review, be followed up with a security audit upon completion of its implementation. The addition of new and complex features may pose critical security risks (e.g. if the governance account is compromised) and should be closely evaluated for potential vulnerabilities (Suggestion 7). More generally, given the system's complexity, extensive code base, and the frequent changes to the implementation, we recommend that the Pendle protocol undergo regular security audits by multiple independent auditing teams.

Verification & New Issues

Verification

Following the delivery of the Initial Audit Report (16 April 2021), the Pendle team provided a response (9 May 2021) to the *Issues* and *Suggestions* reported by the Least Authority team. Our team has verified the *Issues* and *Suggestions* against the provided commit and updated the status of each *Issue* and *Suggestion*, as detailed below (See Specific Issues and Suggestions). However, in verifying the *Issues* and *Suggestions*, we discovered a number of potential areas of concern due to the continued significant complexity of the smart contracts system and recommend a follow up security audit (Suggestion 8).

New Issues

Following the delivery of the *Initial Audit Report*, the Pendle team provided Least Authority with the following:

- A list of major changes and features (9 May 2021) introduced by the Pendle team following the *Initial Audit Report* delivery;
- A List_of_bugs.pdf document (10 May 2021) containing new bugs and fixes identified by the Pendle team following the *Initial Audit Report* delivery; and
- An Overview of issues and changes - Pendle.pdf document (20 May 2021) containing new security issues and resolutions identified by the Pendle team following the *Initial Audit Report* delivery.

The Pendle team has informed Least Authority that there are ongoing efforts by other whitehat teams to review the Pendle smart contracts, and that many of the issues and bugs in the above documentation were identified by those teams. The Pendle team also noted their intent to have the new issues and resolutions outlined in the above documentation further verified by those teams. Least Authority has not been involved in and is unable to confirm those efforts and we recommend publishing results from those security reviews once they have been completed.

The issues and resolutions in the above documentation have not been verified by the Least Authority team, as review of the newly identified issues and resolutions resulting from code refactors and protocol changes are considered out of scope for this security audit.

Conclusions

Due to the continued complexity of the smart contracts system, the number of new issues discovered by Pendle and other teams, and the ongoing changes being made to the implementation, we strongly recommend against deploying the Pendle Protocol until further security review and verification of the smart contracts has been conducted and concluded. We recommend conducting and publishing a follow up security audit by an independent third-party team to review any remaining known issues and verify the suggested resolutions, in addition to identifying unknown potential security issues introduced through recent major code changes, code refactors, and the introduction of new features (Suggestion 8). Finally, we advise that a security audit only be conducted once all development work has been finalized, the smart contracts system is feature complete, and a stable target can be provided to a security auditing team.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Check For WETH Address In The Receive Fallback	Resolved
Issue B: claimLpInterests Potentially Blocked by High Gas Cost	Partially Resolved
Issue C: No Return Value Check for Transfer of Random ERC-20 Token	Resolved
Issue D: updateParamL is Called Twice Within _settleLpInterests in _beforeTokenTransfer	Resolved
Issue E: updateParamL is Missed in addMarketLiquidityDual (Known Issue)	Resolved
Suggestion 1: Remove Unnecessary/Unwarranted Reentrancy Guards	Partially Resolved
Suggestion 2: Create View Function For Loop In claimRewards	Resolved
Suggestion 3: Improve Tests	Partially Resolved
Suggestion 4: Remove Complicated Custom Reentrancy Guard	Unresolved
Suggestion 5: Remove Instances of Redundancy	Resolved
Suggestion 6: Reduce Contract Complexity	Unresolved
Suggestion 7: Conduct Security Audit of Governance Model	Unresolved
Suggestion 8: Conduct a Follow Up Security Audit Prior to Launch [NEW]	Unresolved

Issue A: Check For WETH Address In The Receive Fallback

Location

`contracts/core/PendleRouter.sol#L55`

Synopsis

The receive fallback has no implementation in `PendleRouter.sol`. In Uniswap, this is utilized to ensure that if ETH is being sent to the router, it must come from the wrapped ETH contract. All ETH must be wrapped in order to work and it may be reasonable that users might not wrap their ETH and attempt to send it to the router.

Impact

Without a fallback implementation, ETH could be locked on the router.

Feasibility

Given that wrapped ETH is intended to work with AMMs, this could be feasible.

Remediation

We recommend providing a check that if ETH is sent to the smart contract that it is coming from the WETH contract only. If this is not required, we recommend removing the unnecessary code.

```
assert(msg.sender == WETH); // only accept ETH via fallback from the
WETH contract
```

Status

The check of `msg.sender == WETH` has been implemented.

Verification

Resolved.

Issue B: `claimLpInterests` Potentially Blocked by High Gas Cost

Location

[core/abstract/PendleLiquidityMiningBase.sol#L288]()

[core/abstract/PendleLiquidityMiningBase.sol#L236]()

Synopsis

Too many expiries are added every time a liquidity provider stakes LP tokens,causing the function to be potentially blocked by high gas cost

Impact

Too many expiries may cause the loop in `claimLpInterests` to increase the gas cost to the block gas limit, blocking the call and preventing the liquidity provider from claiming `interests`.

Feasibility

This is feasible, however, it is unlikely that a liquidity provider creates greater than ~100 expiries

Technical Details

A liquidity provider stakes LP tokens and expiries are added:

```
newLpHoldingContract = _addNewExpiry(expiry, xyt, marketAddress);
```

When calling `claimLpInterests()`, interests are settled in a loop across all expiries:

```
for (uint256 i = 0; i < userExpiries[msg.sender].expiries.length; i++) {
_interests =
_interests.add(_settleLpInterests(userExpiries[msg.sender].expiries[i],
msg.sender)); }
```

Mitigation

We recommend limiting the number of expiries from which a user can claim `interests`, based on the gas cost of claiming one expiry interest.

Remediation

A possible remediation is to implement a functionality allowing the liquidity provider to claim `interests` from specific expiries. The liquidity provider can then clear expiries one by one until they can claim all remaining expiries with the usual method.

Status

The Pendle team changed `redeemLpInterests` to be claiming interests for a single expiry.

However, in `_settleLpInterests`, there is no validity check of expiry data (only available in `_updateParamL`).. As a result, we recommend moving the validity check to `_settleLpInterests`, continue to refactor, and look for redundancies and errors.

Verification

Partially Resolved.

Issue C: No Return Value Check for Transfer of Random ERC-20 Token

Location

`contracts/periphery/Withdrawable.sol#L56`

Synopsis

ERC-20 token transfer calls do not have a wrapper to anticipate the event that a token interface does not adhere to the standard of reverting in the case of failure. A missing return value bug may arise leading to older tokens becoming locked on the contract.

Impact

Tokens not adhering to the ERC-20 specification may be locked on any contract utilizing `Withdrawable.sol`.

Preconditions

A token that does not conform to the standard that Pendle protocol expects is deposited and cannot be later withdrawn.

Feasibility

Some tokens that are in use still adhere to a standard that does not return a value. A list of them is provided in the Medium article on the missing return value bug.

Remediation

We recommend using a standard safe transfer wrapper, such as the one used by Uniswap or provided by OpenZeppelin, which will anticipate interfaces with no return value and still allow their utilization.

Status

The Pendle team has implemented OpenZeppelin's `safeTransfer` function for token withdrawal, which has also been added to other transfer functions throughout the system for further protection.

Verification

Resolved.

Issue D: `updateParamL` is Called Twice Within `_settleLpInterests` in `_beforeTokenTransfer`

Location

[core/abstract/PendleMarketBase.sol#L795]

[core/abstract/PendleMarketBase.sol#L756]

Synopsis

`_settleLpInterests` is called twice in `_beforeTokenTransfer`, in which `_updateParamL` is called twice. `updateParamL` is to update interests data when interests are settled, and it is called first in the first `_settleLpInterests`, and then is called again in the second `_settleLpInterests`, before any Pendle token transfer.

Impact

Redundant code is executed increasing gas costs of the operation and decreasing readability.

Preconditions

Any Pendle token is transferred including XYT, OT, and the Lp token.

Mitigation

We recommend setting a time interval limit between two `_updateParamL` calls. If time between the two calls is less than the limit, `_updateParamL` will return no error.

Remediation

We recommend separating update interests logic from the settle interests function (`_settleLpInterests`).

Status

The Pendle team has implemented a mechanism to only `updateParamL()` after the interest has increased by a %, so `_updateParamL` is called only once in `_beforeTokenTransfer`.

Verification

Resolved.

Issue E: `updateParamL` is Missed in `addMarketLiquidityDual` (Known Issue)

Location

[core/abstract/PendleMarketBase.sol#L262]

Synopsis

When called, `updateParamL` updates `interests` data and should be called whenever the total amount of liquidity provision changes. `addMarketLiquidityDual` changes the total amount of liquidity provision, but does not call `updateParamL`.

Impact

`Interests` data is not updated when the total amount of liquidity provision changes.

Preconditions

`addMarketLiquidityDual` is called.

Remediation

We recommend adding `UpdateParamL` in `addMarketLiquidityDual`.

Status

This issue was identified and resolved by the Pendle team during the security audit. In resolving the issue, the Pendle team made a change such that `updateParamL()` must be called in the beginning of `addMarketLiquidityDual` (while minting protocol fees in `_mintProtocolFee()`).

Verification

Resolved.

Suggestions

Suggestion 1: Remove Unnecessary & Unwarranted Reentrancy Guards

Location

Example: addForge and newYieldContracts do not call to untrusted contracts and cannot be part of a reentrancy attack.

Synopsis

The possibility for reentrancy can be used to attack a smart contract, if it calls an attacker controlled smart contract before updating internal state. A correctly designed function can avoid the need for reentrancy guards by avoiding patterns that can be exploited. For example, `addForge` and `newFieldContracts` are two functions that do not call untrusted contracts, so they simply cannot provide an opportunity for a reentrancy attack, and do not require guards as a result.

The use of unnecessary code contributes to complexity, which increases security risks. Furthermore, while minimal, the reentrancy guards require some computation and if a function is to be called frequently, this will result in wasted gas consumption.

Mitigation

We recommend identifying the functions that do not require reentrancy guards and remove them.

Status

The Pendle team has removed reentrancy guards where only trusted sources will be calling, thus making them unnecessary. However, the Pendle team has created a new and custom reentrancy guard that is non-standard and more complex than the existing standard. As a result, we recommend instead using the standard guard provided by OpenZeppelin.

Verification

Partially Resolved.

Suggestion 2: Create View Function For Loop In `claimRewards`

Location

[/core/abstract/PendleLiquidityMiningBase.sol#L282-L284]

Synopsis

A UINT256 array is filled during claiming rewards and passed back to the caller. This is intended to provide a better view of rewards. This does not update any state and includes gas costs everytime rewards are claimed.

Mitigation

We recommend considering further gas optimization on the liquidity mining contract, such as creating a seperate view function to provide the caller with reward data.

Status

`claimRewards()` has been removed entirely from the smart contract.

Verification

Resolved.

Suggestion 3: Improve Tests

Location

/test

Synopsis

The Pendle protocol test suite covers many simple and happy path test cases. However, the code base would benefit from increasing test coverage with specific attention to failure cases. The existing tests are not exhaustive in that they do not comprehensively capture all failure scenarios, which may lead to potentially missing edge cases and errors. Executing tests for failure cases can help determine if error conditions operate and catch problems as expected. Furthermore, comprehensive test coverage helps to identify simple errors and prevents functionality from breaking when new code changes are introduced.

Mitigation

We recommend expanding test coverage to include all success and failure cases.

Status

The Pendle team has expanded their test suite and has stated their intent to continue introducing additional test coverage. However, the smart contracts implementation has changed significantly due to recent major code changes, code refactors, and the introduction of new features. As a result, we are unable to verify if the test suite has been expanded sufficiently to provide comprehensive coverage. We recommend expanding test coverage so that it encompasses the entire implementation.

Verification

Partially Resolved.

Suggestion 4: Remove Complicated Custom Reentrancy Guard

Location

contracts/periphery/PendleNonReentrant.sol

Synopsis

The Pendle protocol uses an unusual reentrancy guard that contains a whitelist of smart contracts that skip the reentrancy check, which is then updated by governance to whitelist the core Pendle smart contracts. The Pendle smart contracts do not call arbitrary contracts except for `IERC20.transfer`.

Mitigation

The Pendle team has notified us they intend to implement a one-state reentrancy check, such that functions on the Pendle router may not be called again while executing. The only exception is for the `redeemDueInterests` function, which may only be called by `pendleMarket` instances. However, this mitigation has not been implemented at the time of our audit and has not yet been verified by our team.

Status

The Pendle team ha used a reentrancy guard from OpenZepplin, in addition to custom, non-standard reentrancy code (e.g. PendleRouterNonReentrant.sol) for a special case `redeemDueInterests` in PendleRouter. However, it is also used on other functions in the router. We recommend using the standard guard provided by OpenZeppelin and remove unnecessary use for reentrancy guard (including standard and custom).

In addition, there are duplicated lines of import (e.g. PendleLiquidityMiningBase.sol#L38-L40). We recommend removing all instances of duplicate code

Verification

Unresolved.

Suggestion 5: Remove Instances of Redundancy

Location

core/abstract/PendleLiquidityMiningBase.sol#L521

core/abstract/PendleMarketBase.sol#L753

Synopsis

Function `_settleLpInterests` in `PendleLiquidMiningBase` calls the other function `_settleLpInterests` in `PendleMarketBase`. The near identical nature of the functions creates confusion and redundancy.

Mitigation

We recommend removing instances of redundancy from the code base.

Status

The Pendle team indicated that the two functions in question calculate two different types of interests: one for the interests accrued for the LP stakers in the liquidity mining contract and the other for interests accrued for the LP holders in the Market. As a result, they cannot be merged.

Verification

Resolved.

Suggestion 6: Reduce Contract Complexity

Synopsis

The Pendle protocol system design and implementation is considerably complex. The coded implementation consists of many lines of code that are interwoven in intricate ways. Furthermore, the inheritance system that has been designed is difficult to comprehend due to a high level of complexity.

Impact

Simple errors are more difficult to identify in a complex system, which may lead to severe security vulnerabilities that have devastating outcomes in smart contracts containing significant value. Thus, increased complexity in a system's design and implementation results in a greater probability that mistakes will go undetected.

Mitigation

We recommend that the Pendle protocol team find ways to reduce the complexity of the code. In addition, we strongly recommend additional security audits of the protocol conducted by different teams in order to identify potential security issues.

Status

The Pendle team has responded that they have refactored the codebase to be less complex, thus making the system less prone to error. While reduced complexity has been introduced to some areas of the code base, we find that the complexity of the system is still too large and widespread, thus prohibiting our ability to verify and confirm that complexity has been sufficiently reduced.

In addition, the significant amount of newly introduced changes (i.e. both upgrades and in response to issues found by both the Least Authority and the Pendle teams since the delivery of the *Initial Audit Report*) have left the codebase in a state that requires a full follow up security audit in order to verify the claim that system complexity has been sufficiently reduced. We have also identified areas in the code base where additional complexity has been newly introduced (e.g. the reentrance guard mentioned in Suggestion 4).

We strongly recommend that the complexity of the code be further reduced and that a follow up security audit be conducted by an independant, third-part team following the completion of all development.

Verification

Unresolved.

Suggestion 7: Conduct Security Audit of Governance Model

Synopsis

The Pendle protocol governance model was out of scope for our security review. The addition of new, complex features may pose critical security risks (e.g. if the governance account is compromised) and should be closely evaluated for potential vulnerabilities.

Mitigation

We recommend that a security audit of the governance model be conducted following the implementation of both the interim Multi-Signature smart contract and the final DAO smart contract, in order to analyze the design and functionality for potential security vulnerabilities.

Status

The Pendle team responded they will conduct an audit of the governance module prior to its launch. To the best of our knowledge, the security audit has not been completed at the time of this verification. However, the Pendle team has stated that a Gnosis Safe Multi-Signature smart contract will be used as the governance role before the governance module is launched.

Verification

Unresolved.

Suggestion 8: Conduct a Follow Up Security Audit Prior to Launch [NEW]

Synopsis

As a result of the continued significant complexity of the smart contracts system and the number of new issues discovered by the Pendle team, we strongly recommend against deploying the Pendle Protocol without further security review and modifications, as we consider it to be unsafe in its current state.

Mitigation

We strongly recommend conducting a follow up security audit by an independent third-party team to review known issues and verify the suggested resolutions, in addition to identifying unknown potential security issues introduced through recent major code changes, code refactors, and the introduction of new features. We strongly advise that a security audit only be conducted once all development work has been finalized, the smart contracts system is feature complete, and a stable target can be provided to the security auditing team.

Status

The Pendle team has responded that there are ongoing efforts by three whitehat teams to review the Pendle smart contracts. Least Authority has not been involved in those efforts and we recommend publishing results from those security reviews once they have been completed.

Verification

Unresolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. Although we are a small team, we believe that we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.