Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Pendle Token
Website:	pendle.finance
Platform:	Ethereum
Language:	Solidity
Date:	May 8th, 2024

Table of contents

Email: audit@EtherAuthority.io

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Pendle token smart contract from pendle.finance was audited extensively. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on May 8th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- This Solidity contract is for a token called PENDLE, which implements the IPENDLE interface. Here's a breakdown of the contract:
    - **Constructor:** The contract constructor initializes various parameters including the token distribution to different addresses, emission rates, start time, and other configuration parameters.
    - **Token Transfer Functions:** Functions for transferring tokens (transfer, transferFrom, approve, increaseAllowance, decreaseAllowance) are implemented.
    - **Delegation Functions:** Functions for delegate voting power (delegate, delegateBySig), and related internal functions are implemented.
    - **Vote Calculation Functions:** Functions for calculating vote balance (getCurrentVotes, getPriorVotes), and related internal functions are implemented.
    - **Internal Transfer and Approval Functions:** Internal functions for transferring tokens, approving spending, and updating voting power are implemented (_transfer, _approve, _moveDelegates, _writeCheckpoint).
    - **Utility Functions:** Utility functions like safe32, getChainId, and _getCurrentWeek are implemented.
    - **Config Changes Functions:** Functions for initiating and applying configuration changes (initiateConfigChanges, applyConfigChanges) are implemented.

- **Liquidity Emission Functions:** Functions for claiming liquidity emissions (claimLiquidityEmissions, _mintLiquidityEmissions) are implemented.
- **Token Minting and Burning Functions:** Functions for minting and burning tokens (_mint, _burn) are implemented.

- This contract governs the behavior of the PENDLE token including transfers, voting, delegation, configuration changes, and emission of additional tokens for liquidity incentives.

Audit scope

Name	**Code Review and Security Analysis Report for Pendle Token Smart Contract**
Platform	**Ethereum**
File	PENDLE.sol
Smart Contract Code	0x808507121b80c02388fad14726482e061b8da827
Audit Date	May 8th, 20234

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Pendle • Symbol: PENDLE • Decimals: 18	**YES, This is valid.**
Ownership control: • The Governance owner can initiate Configuration changes. • The liquidity incentives recipient owner can claim liquidity emissions. • Allows governance to withdraw Ether in a Pendle contract in case of accidental ETH transfer into the contract. • Allows governance to withdraw all IERC20 compatible tokens in a Pendle contract in case of accidental token transfer into the contract. • Allows the pending governance address to finalize the change governance process by the governance owner. • Allows the current governance to set the pending governance address by the governance owner.	**YES, This is valid. We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **“Secured”**. Also, these contracts contain owner control, which does not make them fully decentralized.

We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low, and 5 very low level issues.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Business Risk Analysis

Category	Result
Buy Tax	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	Not Detected
Fee Check	No
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	No
Pause Transfer?	No
Max Tax?	No
Is it Anti-whale?	No
Is Anti-bot?	Not Detected
Is it a Blacklist?	Not Detected
Blacklist Check	No
Can Mint?	No
Is it Proxy?	No
Can Take Ownership?	No
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. Smart contract contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Pendle Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Pendle Token.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Pendle Token smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are well commented on. and the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Another source of information was its official website: https://www.pendle.finance which provided rich information about the project architecture and tokenomics.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure that are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	approve	external	Passed	No Issue
3	transfer	external	Passed	No Issue
4	transferFrom	external	Passed	No Issue
5	increaseAllowance	write	Public functions should be declared external	Refer Audit Findings
6	decreaseAllowance	write	Public functions should be declared external	Refer Audit Findings
7	burn	write	Public functions should be declared external	Refer Audit Findings
8	allowance	external	Public functions should be declared external	Refer Audit Findings
9	balanceOf	external	Passed	No Issue
10	getCurrentVotes	external	Passed	No Issue
11	delegate	write	Public functions should be declared external	Refer Audit Findings
12	delegateBySig	write	Public functions should be declared external	Refer Audit Findings
13	getPriorVotes	read	Public functions should be declared external	Refer Audit Findings
14	_delegate	internal	Passed	No Issue
15	_transfer	internal	Passed	No Issue
16	_approve	internal	Passed	No Issue
17	_moveDelegates	internal	Passed	No Issue
18	_writeCheckpoint	internal	Passed	No Issue
19	safe32	internal	Passed	No Issue
20	getChainId	internal	Passed	No Issue
21	initiateConfigChanges	external	access only Governance	No Issue
22	applyConfigChanges	external	Passed	No Issue
23	claimLiquidityEmissions	external	Passed	No Issue
24	_mintLiquidityEmissions	internal	Passed	No Issue
25	_getCurrentWeek	internal	Passed	No Issue
26	_mint	internal	Passed	No Issue
27	_burn	internal	Passed	No Issue
28	withdrawEther	external	Low-Level Calls, Missing Zero Address Validation	Refer Audit Findings
29	withdrawToken	external	access only Governance	No Issue
30	initialized	modifier	Passed	No Issue
31	onlyGovernance	modifier	Passed	No Issue
32	claimGovernance	write	Passed	No Issue
33	transferGovernance	write	access only Governance	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium-severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Informational / Best Practices:

(1) Low-Level Calls:

Withdrawable.sol

```
function withdrawEther(uint256 amount, address payable sendTo) external onlyGovernance {
    (bool success, ) = sendTo.call{value: amount}("");
    require(success, "WITHDRAW_FAILED");
    emit EtherWithdraw(amount, sendTo);
}
```

This contract uses low-level calls, which may be unsafe.

Resolution: Enhance safety by reviewing low-level calls, considering high-level alternatives, and consulting security experts. Prioritize code security and integrity.

(2) Floating Pragma: **IPENDLE.sol, PENDLE.sol, Permissions.sol, Withdrawable.sol**

```
pragma solidity 0.7.6
```

This contract may not function as expected due to inconsistent solidity compiler versions being specified.

Resolution: Specify and lock the Solidity pragma version to a fixed and known version in your smart contract to avoid potential issues caused by unexpected compiler behavior in future versions.

(3) Public functions should be declared external: **PENDLE.sol**

Some functions in this contract should be declared as external in order to save gas.

- increaseAllowance
- decreaseAllowance
- burn
- delegate
- delegateBySig
- getPriorVotes

Resolution: External functions can be called more efficiently, reducing gas costs compared to public functions. Update the function declarations accordingly to enhance gas efficiency.

(4) Division Before Multiplication: **PENDLE.sol**

```
lastWeeklyEmission = lastWeeklyEmission.mul(emissionRateMultiplierNumerator).div(
    CONFIG_DENOMINATOR
);
```

```
lastWeeklyEmission = totalSupply.mul(terminalInflationRateNumerator).div(
    CONFIG_DENOMINATOR
);
```

The order of operations used may result in a loss of precision.

Resolution: Ensure precision by prioritizing multiplication before division in calculations. Review and adjust the order of operations accordingly, and rigorously test the updated code for accuracy.

(5) Missing Zero Address Validation: **Withdrawable.sol**

```
function withdrawEther(uint256 amount, address payable sendTo) external onlyGovernance {
    (bool success, ) = sendTo.call{value: amount}("");
    require(success, "WITHDRAW_FAILED");
    emit EtherWithdraw(amount, sendTo);
}
```

Some functions in this contract may not appropriately check for zero addresses being used.

Resolution: Implement zero address validation checks in relevant functions to prevent unintended behavior and enhance the contract's security.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble. The following are Admin functions:

PENDLE.sol

- initiateConfigChanges: The Governance owner can initiate Configuration changes.
- claimLiquidityEmissions: The liquidity incentives recipient owner can claim liquidity emissions.

Withdrawable.sol

- withdrawEther: Allows governance to withdraw Ether in a Pendle contract in case of accidental ETH transfer into the contract.
- withdrawToken: Allows governance to withdraw all IERC20 compatible tokens in a Pendle contract in case of accidental token transfer into the contract.

Permissions.sol

- claimGovernance: Allows the pendingGovernance address to finalize the change governance process by the governance owner.
- transferGovernance: Allows the current governance to set the pendingGovernance address by the governance owner.

To make the smart contract 100% decentralized, we suggest renouncing ownership of the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of Etherscan web links. And we have used all possible tests based on given objects as files. We observed 5 Informational issues in the smart contracts. but those are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

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 white box penetration testing. We look at the project's website 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.

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, and 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.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

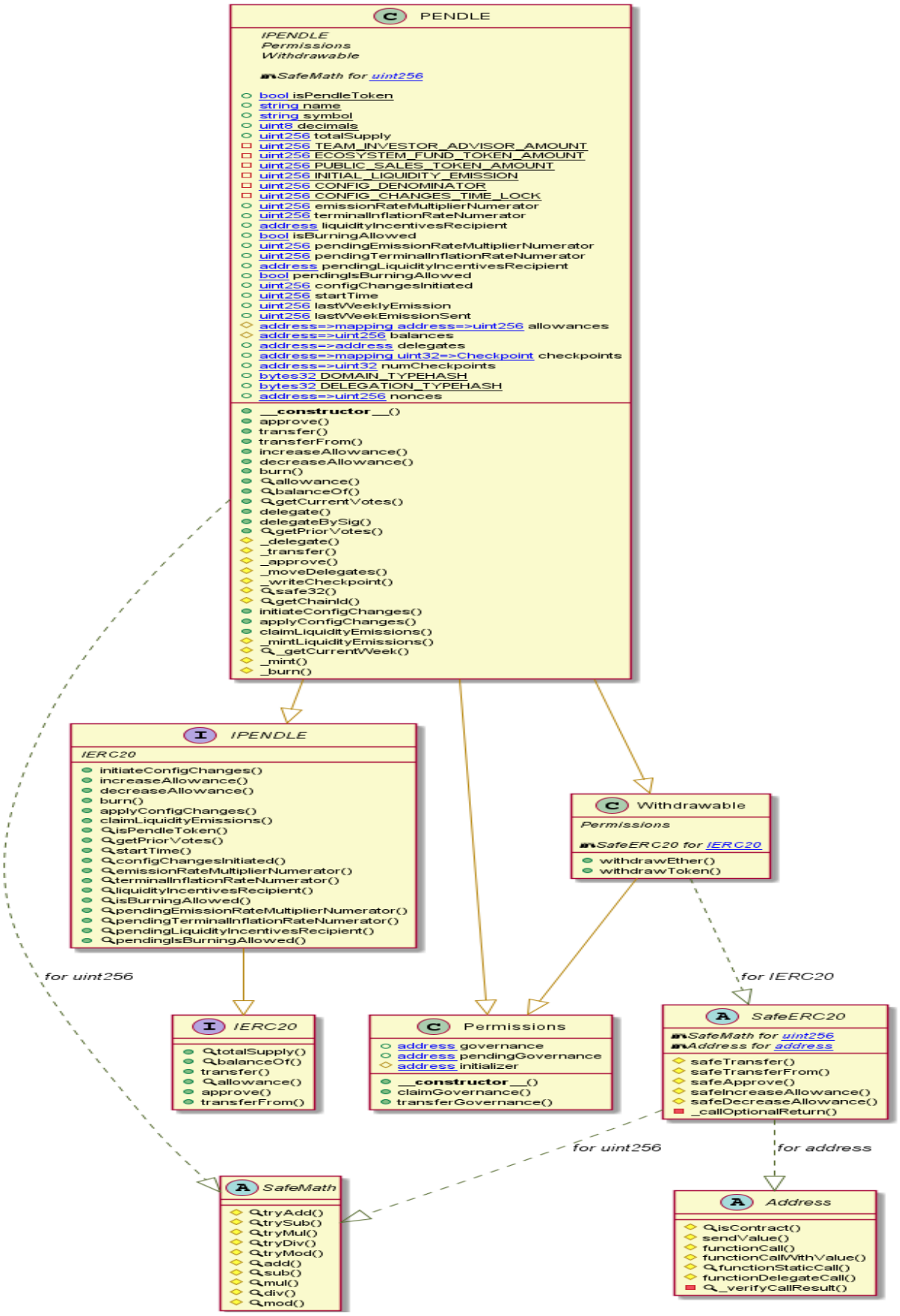
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - PENDLE Token



Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

Slither Log >> PENDLE.sol

```
PENDLE._mintLiquidityEmissions() (PENDLE.sol#1107-1126) performs a multiplication on the result of a division:
        - lastWeeklyEmission = lastWeeklyEmission.mul(emissionRateMultiplierNumerator).div(CONFIG_DENOMINATOR) (PENDLE.sol#1114-
1116)
        - lastWeeklyEmission = totalSupply.mul(terminalInflationRateNumerator).div(CONFIG_DENOMINATOR) (PENDLE.sol#1118-1120)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

PENDLE._writeCheckpoint(address,uint32,uint256,uint256) (PENDLE.sol#1027-1045) uses a dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (PENDLE.sol#1036)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Withdrawable.withdrawEther(uint256,address).sendTo (PENDLE.sol#623) lacks a zero-check on :
                - (success) = sendTo.call{value: amount}() (PENDLE.sol#624)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in Withdrawable.withdrawEther(uint256,address) (PENDLE.sol#623-627):
        External calls:
        - (success) = sendTo.call{value: amount}() (PENDLE.sol#624)
        Event emitted after the call(s):
        - EtherWithdraw(amount,sendTo) (PENDLE.sol#626)
Reentrancy in Withdrawable.withdrawToken(IERC20,uint256,address) (PENDLE.sol#636-643):
        External calls:
        - token.safeTransfer(sendTo,amount) (PENDLE.sol#641)
        Event emitted after the call(s):
        - TokenWithdraw(token,amount,sendTo) (PENDLE.sol#642)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

PENDLE.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (PENDLE.sol#899-918) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= expiry,SIGNATURE_EXPIRED) (PENDLE.sol#916)
PENDLE.applyConfigChanges() (PENDLE.sol#1080-1100) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(configChangesInitiated != 0,UNINITIATED_CONFIG_CHANGES) (PENDLE.sol#1081)
        - require(bool,string)(block.timestamp > configChangesInitiated + CONFIG_CHANGES_TIME_LOCK,TIMELOCK_IS_NOT_OVER) (PENDLE
.sol#1082-1085)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
Address.isContract(address) (PENDLE.sol#26-35) uses assembly
        - INLINE ASM (PENDLE.sol#33)
Address._verifyCallResult(bool,bytes,string) (PENDLE.sol#171-188) uses assembly
        - INLINE ASM (PENDLE.sol#180-183)
PENDLE.getChainId() (PENDLE.sol#1052-1058) uses assembly
        - INLINE ASM (PENDLE.sol#1054-1056)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Address.functionCall(address,bytes) (PENDLE.sol#79-81) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (PENDLE.sol#104-106) is never used and should be removed
Address.functionDelegateCall(address,bytes) (PENDLE.sol#153-155) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (PENDLE.sol#163-169) is never used and should be removed
Address.functionStaticCall(address,bytes) (PENDLE.sol#129-131) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (PENDLE.sol#139-145) is never used and should be removed
Address.sendValue(address,uint256) (PENDLE.sol#53-59) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (PENDLE.sol#479-488) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (PENDLE.sol#495-498) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (PENDLE.sol#490-493) is never used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (PENDLE.sol#468-470) is never used and should be removed
SafeMath.div(uint256,uint256,string) (PENDLE.sol#434-437) is never used and should be removed
SafeMath.mod(uint256,uint256) (PENDLE.sol#396-399) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (PENDLE.sol#454-457) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (PENDLE.sol#268-272) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (PENDLE.sol#304-307) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (PENDLE.sol#314-317) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (PENDLE.sol#289-297) is never used and should be removed
SafeMath.trySub(uint256,uint256) (PENDLE.sol#279-282) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version>=0.6.2<0.8.0 (PENDLE.sol#3) is too complex
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
Low level call in Address.sendValue(address,uint256) (PENDLE.sol#53-59):
        - (success) = recipient.call{value: amount}() (PENDLE.sol#57)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (PENDLE.sol#114-121):
        - (success,returndata) = target.call{value: value}(data) (PENDLE.sol#119)
Low level call in Address.functionStaticCall(address,bytes,string) (PENDLE.sol#139-145):
        - (success,returndata) = target.staticcall(data) (PENDLE.sol#143)
Low level call in Address.functionDelegateCall(address,bytes,string) (PENDLE.sol#163-169):
        - (success,returndata) = target.delegatecall(data) (PENDLE.sol#167)
Low level call in Withdrawable.withdrawEther(uint256,address) (PENDLE.sol#623-627):
        - (success) = sendTo.call{value: amount}() (PENDLE.sol#624)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter Permissions.transferGovernance(address)._governance (PENDLE.sol#603) is not in mixedCase
Parameter PENDLE.initiateConfigChanges(uint256,uint256,address,bool)._emissionRateMultiplierNumerator (PENDLE.sol#1061) is not i
n mixedCase
Parameter PENDLE.initiateConfigChanges(uint256,uint256,address,bool)._terminalInflationRateNumerator (PENDLE.sol#1062) is not in
 mixedCase
Parameter PENDLE.initiateConfigChanges(uint256,uint256,address,bool)._liquidityIncentivesRecipient (PENDLE.sol#1063) is not in m
ixedCase
Parameter PENDLE.initiateConfigChanges(uint256,uint256,address,bool)._isBurningAllowed (PENDLE.sol#1064) is not in mixedCase
Constant PENDLE.isPendleToken (PENDLE.sol#660) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

PENDLE.slitherConstructorConstantVariables() (PENDLE.sol#651-1151) uses literals with too many digits:
        - INITIAL_LIQUIDITY_EMISSION = 1200000 * 1e18 (PENDLE.sol#669)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

PENDLE.startTime (PENDLE.sol#681) should be immutable
Permissions.initializer (PENDLE.sol#567) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
PENDLE.sol analyzed (8 contracts with 84 detectors), 44 result(s) found
```

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

PENDLE.sol

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Address.functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 114:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Withdrawable.withdrawEther(uint256,address payable): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 623:4:

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

more

Pos: 1054:8:

Email: audit@EtherAuthority.io

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 1130:18:

Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
more
Pos: 624:27:

Gas costs:

Gas requirement of function PENDLE.claimLiquidityEmissions is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1102:4:

Similar variable names:

PENDLE._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 1137:50:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 984:8:

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

PENDLE.sol

```
Compiler version >=0.6.2 <0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:2
Error message for require is too long
Pos: 9:57
Error message for require is too long
Pos: 9:114
Error message for require is too long
Pos: 9:139
Error message for require is too long
Pos: 9:163
Error message for require is too long
Pos: 9:362
Error message for require is too long
Pos: 9:483
Error message for require is too long
Pos: 13:513
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:572
Contract has 19 states declarations but allowed no more than 15
Pos: 1:650
Constant name must be in capitalized SNAKE_CASE
Pos: 5:659
Constant name must be in capitalized SNAKE_CASE
Pos: 5:660
Constant name must be in capitalized SNAKE_CASE
Pos: 5:661
Constant name must be in capitalized SNAKE_CASE
Pos: 5:662
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:736
Avoid making time-based decisions in your business logic
Pos: 21:757
Avoid making time-based decisions in your business logic
Pos: 17:915
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:1053
Avoid making time-based decisions in your business logic
Pos: 34:1076
Avoid making time-based decisions in your business logic
Pos: 13:1082
Avoid making time-based decisions in your business logic
```

```
Pos: 19:1129
```

Software analysis result:

This software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io