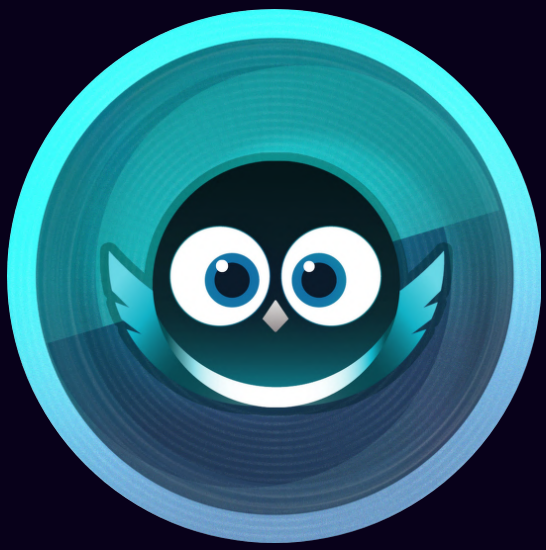

SMART CONTRACT AUDIT

zokyo

June 22nd 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

TECHNICAL SUMMARY

This document outlines the overall security of the Penpie smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Penpie smart contracts codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the audit. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contracts but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Penpie team put in place a bug bounty program to encourage further active analysis of the smart contracts.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Penpie repository:
https://github.com/magpiexyz/pendleMagpie

Last commit - 0097c836a61ffe869519143076b346f664b2ade9

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- MasterPenpie.sol
- PendleStaking.sol
- mPendleConvertor.sol
- mPendleOFT.sol
- PendleMarketDepositHelper.sol
- PenpieReceiptToken.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Penpie smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contracts by industry leaders.
03	Thorough manual review of the codebase line by line.

Executive Summary

The codebase was subjected to a security audit by the Zokyo team. The provided contracts for the audit exhibit are well written and structured. The results of the audit, including all findings, are detailed in the "Complete Analysis" section. No critical issues were discovered during the audit. However, there were identified issues with varying levels of severity, including high, low, and informational issues.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Penpie team and the Penpie team is aware of it, but they have chosen to not solved it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Lack of access control in increaseLockTime function enables unauthorized position lock	High	Acknowledged
2	Execution flow in _onlyWhiteListed modifier leaves modified function unexecuted	High	Resolved
3	Missing Reentrancy modifier in depositVlPenPieFor(...) and withdrawVlPenPierFor(...) methods	Low	Resolved
4	Unrestricted Reactivation of Markets with Null Rewarder Address in PendleMarketDepositHelper Contract	Low	Resolved
5	Missing length check for rewarders array in harvestVePendleReward function	Low	Resolved
6	Unused removePoolInfo function in PendleStaking contract	Low	Resolved
7	Unlimited emission rate can be set for penPiePerSec	Low	Acknowledged
8	Unutilized bonus reward tokens functionality	Low	Resolved
9	Upgradeable contracts missing method to disable initializers for implementation contracts	Low	Resolved
10	Method convert(...) does not validate parameter `_mode`	Low	Acknowledged
11	Zero address validation missing for setPendleStaking	Low	Resolved
12	Lack of Input Validation in addBonusRewardForAsset function	Low	Resolved
13	Missing Reentrancy checks in internal methods	Low	Resolved
14	Missing check for the `newMinter` parameter if it's an EOA	Low	Acknowledged

#	Title	Risk	Status
15	Unused imports, events, custom errors and modifiers	Informational	Resolved
16	Unbounded iteration in massUpdatePools function (MasterPenPie.sol)	Informational	Acknowledged
17	Missing initialization of Reentrancy Guard Upgradeable Contract	Informational	Resolved
18	Missing initialization of PausableUpgradeable Contract	Informational	Acknowledged
19	Reuse code by adding a modifier	Informational	Resolved
20	Missing events in methods called only by Operator and Admin	Informational	Acknowledged
21	Wrong NatSpec comments.	Informational	Resolved
22	Use specific Solidity compiler version	Informational	Resolved
23	Upgradeable contracts not using upgradeable smart contract libraries	Informational	Acknowledged
24	Redundant use of `this` keyword in the function invocations	Informational	Resolved

HIGH | ACKNOWLEDGED

Lack of access control in increaseLockTime function enables unauthorized position lock

The increaseLockTime() function in the PendleStaking contract allows anyone to lock the position of a PendleStaking for up to 104 weeks. The function takes an _unlockTime parameter, which represents the duration in seconds, to increase the lock time. Then, it invokes the increaseLockPosition function from the IPVotingEscrowMainchain contract, passing 0 as a pendle amount to be pulled in from user to lock and the calculated unlockTime as the lock duration. The vulnerability arises from the lack of access control or permission checks in this function. As a result, any address can call this function and increase the lock time for the position without any restrictions.

Recommendation:

Consider implementing proper access controls or permission checks in the increaseLockTime() function.
Comment from the client team: I don't think, this needs to be changed because we want the lock period to increase eventually and, the Pendle side is already checking this for us, if the lock time is not greater than last time then it will set the old lock time and if we want to change it, then we have to add timestamp at the time of deployment and do the calculation of unlock time with that in convertpendle, so timestamp will not increase, I haven't touched this

HIGH | RESOLVED

Execution flow in _onlyWhiteListed modifier leaves modified function unexecuted

The MasterPenPIe._onlyWhiteListed() modifier checks whether the sender is part of the AllocationManagers, PoolManagers, or is the contract owner. If any of these conditions are satisfied, the modifier executes a return statement and immediately exits the function, bypassing the execution of the modified function entirely. As a result, the _; symbol after the return statements are unreachable, and the modified function does not execute.
As a result, in the MasterPenPie.updatePoolsAlloc() function, which utilizes the _onlyWhiteListed() modifier, if the calling address is whitelisted, the modifier will prevent the execution of the function. This means that the allocation points for staking tokens will not be updated.

```
        it("Invalid modifier execution flow", async () => {
            let stakingToken1: StandardTokenMock;
            stakingToken1 = await setup.newToken(20, 50)
            let allocPoint1 = ether(200)

            let totalAllocBefore = await masterPenPie.totalAllocPoint();

            await deposit(player1);
            await masterPenPie.updatePoolsAlloc([stakingToken1.address],[allocPoint1]);

            let totalAllocAfter = await masterPenPie.totalAllocPoint()

            expect((await
masterPenPie.tokenToPoolInfo(stakingToken1.address)).allocPoint).to.eq(0)
            expect(totalAllocBefore).to.eq(totalAllocAfter);
        });
```

Recommendation:

Modifying the _onlyWhiteListed() modifier as follows so as the execution flow will correctly proceed to the modified function:

```
    modifier _onlyWhiteListed() {
        require(
            AllocationManagers[msg.sender] ||
            PoolManagers[msg.sender] ||
            msg.sender == owner(),
            "OnlyWhiteListedAllocaUpdator"
        );
        _;
}
```

Fixed: Issue fixed in commit 0470a8c

LOW | RESOLVED

Missing Reentrancy modifier in depositVlPenPieFor(...) and withdrawVlPenPierFor(...) methods

In contract MasterPenPie, the methods depositVlPenPieFor(...) and withdrawVlPenPierFor(...) are missing non-reentrant modifier unlike other deposit and withdraw methods in the same contract, and they all are using the same internal method logic.

Recommendation:
Add the non-reentrant modifier to the depositVlPenPieFor(...) and withdrawVlPenPierFor(...) methods.

Fix: Issue fixed in commit 4a8a2

LOW | RESOLVED

Unrestricted Reactivation of Markets with Null Rewarder Address in PendleMarketDepositHelper Contract

The `PendleMarketDepositHelper` contract currently possesses a potential vulnerability, where a removed market could be reactivated by an operator with a null rewarder address. The potential loophole lies in the combination of `removePoolInfo` and `setPoolInfo` functions.

When `removePoolInfo` is invoked, it doesn't just toggle the `isActive` flag to false, but it completely eliminates the `PoolInfo` structure, resetting both `rewarder` and `isActive` to their default values (which are `address(0)` and `false` respectively).
Subsequently, an operator or malicious actor with operator privileges can call `setPoolInfo` and set `isActive` to true, even if the `rewarder` address is null. This could potentially lead to unforeseen behavior or loss of funds, given that a null address is incapable of functioning as a rewarder.

Proof of Concept:
Consider a scenario where a market has an address referred to as 'marketAddress'.

1. The operator executes `removePoolInfo(marketAddress)`.
2. The `PoolInfo` for 'marketAddress' is thereby removed, setting the `rewarder` to `address(0)` and `isActive` to false.
3. The operator or a potential attacker with operator privileges then executes `setPoolInfo(marketAddress, address(0), true)`.
4. Consequently, the `PoolInfo` for 'marketAddress' is reactivated but with a null `rewarder` address.

Recommendation:
Prevent the possibility of markets being reactivated with a null rewarder address. Amend the `setPoolInfo` function to revert the process if the `rewarder` is the null address. This ensures that a market cannot be reactivated unless there's a valid `rewarder`

Fixed: Issue fixed in commit f2e27ad

LOW | RESOLVED

Missing length check for rewarders array in harvestVePendleReward function

The purpose of the harvestVePendleReward function is to distribute rewards to pools and perform fee transfers, but it fails to validate if the _rewarders array has the same length as the _pools array, potentially leading to out-of-bounds array access. The function expects both arrays to have the same length, assuming that each pool in the _pools array corresponds to a specific rewarder in the _rewarders array. However, there is no explicit check to ensure that the lengths of the two arrays match.

Recommendation:
Add a validation check at the beginning of the function to ensure that the _pools and _rewarders arrays have the same length.

LOW | RESOLVED

Unused removePoolInfo function in PendleStaking contract

The PendleMarketDepositHelper contract has the removePoolInfo and setPoolInfo functions, which are intended to be called only by an operator, as enforced by the _onlyOperator modifier. Inside the initializer of this contract, the PendleStaking contract is designated as the first operator. Its function to register a new pool (registerPool()) makes a call to the setPoolInfo function to add a rewarder and indicate that the pool is active. However, no function within the PendleStaking contract actually calls the removePoolInfo function. Consequently, the removePoolInfo function cannot be executed by the intended operator, which can lead to a discrepancy between the contract's design and its implementation.

Recommendation:
Consider updating a relevant functionality within the PendleStaking contract to call the removePoolInfo function when necessary. This may involve adding a new function specifically designed to invoke the removePoolInfo functionality.

LOW | ACKNOWLEDGED

Unlimited emission rate can be set for penPiePerSec

In Contract MasterPenPie, the method updateEmissionRate(...) can be used by the owner to set any emission rate which if set very high can be manipulated by the owner to obtain a high reward share as penPiePerSec is used in reward calculation methods.

Recommendation:
Timelock-based ownership contract can be used to update the emission rate which will give time to other protocol users in such incidents.

LOW | RESOLVED

Unutilized bonus reward tokens functionality

The addBonusRewardForAsset function in PendleStaking contract allows the owner to add bonus reward tokens associated with a specific market address. However, the bonus reward tokens stored in the assetToBonusRewards mapping are not utilized or accessed in any functionality within the Pendle staking contract, including the claimed functionality.
This means that although the addBonusRewardForAsset function allows for the addition of bonus reward tokens, there is no corresponding code that utilizes these tokens during the claim process or any other relevant functionality. As it does not directly affect the core functionality or security of the Pendle staking contract, it signifies a discrepancy between the existence of the bonus reward tokens in the assetToBonusRewards mapping and the lack of utilization within the contract.

Recommendation:
Review the intended functionality and purpose of the bonus reward tokens and the addBonusRewardForAsset function. If the bonus reward tokens are intended to serve a specific purpose, such as being distributed to stakers during the claim process, ensure that the relevant functionality is implemented within the contract. Otherwise, consider removing the function and the corresponding storage of bonus reward tokens from the contract.

LOW | RESOLVED

Upgradeable contracts missing method to disable initializers for implementation contracts

Contract PendleStaking.sol, Contract mPendleConverter.sol, Contract PendleMarketDepositHelper.sol and Contract MasterPenPie have imported the `Initializable` contract and using `initializer` modifier. It is recommended in OpenZeppelin's documentation to not leave the implementation contract uninitialised as attacker can take advantage of the same and that may affect the proxy contract.

Recommendation:
Use this suggested method in OpenZeppelin's documentation to mitigate this issue.

Fix: The issue was fixed partially in commit 4a8a2. The fix is missing in PendleMarketDepositHelper and MasterPenpie contract.

Fix: Issue fixed in commit `0097c`

LOW | ACKNOWLEDGED

Method convert(...) does not validate parameter `_mode`

In contract mPendleConverter.sol, the method `convert(uint _amount, uint _mode)` has a comment mentioning _mode can be `0` or 1. It should be validated in the method logic as well.

Recommendation:
Add the following require statement.
`require(_mode >=0 && _mode <=1, "Invalid mode")`In the future, if this _mode accepts other values, the `require` statement can be updated accordingly.

LOW | RESOLVED

Zero address validation missing for setPendleStaking

In mPendleConverted.sol contract, the method `setPendleStaking(address _pendleStaking)` does not check if address _pendleStaking is zero-address or not. Setting a zero-address accidentally can cause DoS for method lockAllPendle().

Recommendation:
Add zero-address validation for the _pendleStaking parameter.
Fix: As of commit `0097c`, issue is fixed.

LOW | RESOLVED

Lack of Input Validation in addBonusRewardForAsset function

Description:
The `addBonusRewardForAsset` function in the pendleStaking Contract is susceptible to lack of input validation.
The function does not include checks to validate that the input addresses `_market` and `_bonusToken` are non-zero. This could potentially lead to misuse of the function, allowing the addition of invalid or zero addresses.
For input validation, the function does not prevent calls such as `addBonusRewardForAsset(0x0, 0x0)`, which should typically be disallowed.

Recommendation:
Add checks to ensure that the `_market` and `_bonusToken` addresses are valid (non-zero) before proceeding with the function execution.

Fixed: Issue fixed in commit f2e27ad

Client Comment : No need for addBonusRewardForAsset function anymore since reward tokens are now directly from Pendle Market Token

LOW | RESOLVED

Missing Reentrancy checks in internal methods

In contract PendleMarketDepositHelper.sol, the internal methods _depositMarket(...) and _withdrawMarket(...) missing non-reentrant modifier although it is imported, and these methods are used in public-facing methods such as depositMarket(...), depositMarketFor(...) and withdrawMarket(...).

Recommendation:
It is recommended to add `nonReentrant` modifier in the _depositMarket(...) and _withdrawMarket(...) internal methods.
Fix: Issue fixed in commit 4a8a2

LOW | ACKNOWLEDGED

Missing check for the `newMinter` parameter if it's an EOA

In contract mPendleOFT.sol, the method `setMinter(address _newMinter)` sets a new minter to mint the mPendle token which should always be minted by mPendleConverter method only after the same amount of Pendle has been deposited to convert.
Setting an EOA as a minter risk minting mPendle tokens without depositing Pendle tokens.

Recommendation:
It is recommended to add a check to ensure _newMinter is not an EOA.

INFORMATIONAL | RESOLVED

Unused imports, events, custom errors and modifiers

The code base contains unused imports, events, custom errors and modifiers that should be removed to improve code cleanliness and eliminate unnecessary dependencies.

- MasterPenPie.sol#L18

```
import "../interfaces/ILocker.sol";
```

- MasterPenPie.sol#L93

```
    event EmergencyWithdraw( address indexed _user, address indexed
_stakingToken, uint256 _amount );
```

- MasterPenPie.sol#L100

```
    event LockFreePoolUpdated(address _stakingToken, bool _isRewardPenPie);
```

- PendleMarketDepositHelper.sol#L15

```
import { IPendleMarket } from "../interfaces/pendle/IPendleMarket.sol";
```

- mPendleConvertor.sol#L38

```
    event HelperSet(address indexed _helper);
```

- mPendleConvertor.sol#L45

```
    error PendleStakingNotSet();
```

- mPendleConvertor.sol#L46

```
    error MustBeContract();
```

- mPendleConvertor.sol#L47

```
    error NoIncentive();
```

- mPendleConvertor.sol#L11

```
import { Address } from "@openzeppelin/contracts/utils/Address.sol";
```

- PendleStaking.sol#L128

```
    error OnlyActiveFee();
```

- PendleStaking.sol#L104

```
    event PoolRemoved(uint256 _pid, address _lpToken);
```

- PendleStaking.sol#L105

```
    event PoolHelperUpdated(address _lpToken);
```

- PendleStaking.sol#L171

```
    modifier _onlyActivePoolHelper(address _market) {
        Pool storage poolInfo = pools[_market];

        if (msg.sender != poolInfo.helper) revert OnlyPoolHelper();
        if (!poolInfo.isActive) revert OnlyActivePool();
        _;
    }
```

Recommendation:
Consider removing listed dependencies from the code.

INFORMATIONAL | ACKNOWLEDGED

Unbounded iteration in `massUpdatePools` function (MasterPenPie.sol)

Description:
The `massUpdatePools` function iterates over all pool elements and updates them by calling `updatePool`. This could potentially cause the function to fail if the number of pools becomes too large, causing the transaction to consume more gas than the Ethereum block gas limit.

Recommendation:
Consider implementing a paginated approach in the `massUpdatePools` function. This can be accomplished by introducing two parameters, `startIndex` and `endIndex` in the function. This approach would allow the function to process a specified range of pools in each call, thereby limiting the number of loops in one transaction.

Client Comment : only for Pendle Finance integration, which pools number shouldn't go extreme

INFORMATIONAL | RESOLVED

Missing initialization of Reentrancy Guard Upgradeable Contract

Contract PendleStaking.sol, Contract mPendleConverter.sol, Contract PendleMarketDepositHelper.sol and Contract MasterPenPie, these contracts inherits `ReentrancyGuardUpgradeable` which has an init method `` `__ReentrancyGuard_init` `` that needs to be called in the `initialize()` to set the _status to NOT_ENTERED state.

Recommendation:
Call the `` `__ReentrancyGuard_init` `` method in the `initialize()` method.
Fix: Issue is partially fixed as `__ReentrancyGuard_init` still missing in MasterPenpie_init
Fix: Issue fixed in commit `0097c`

INFORMATIONAL | ACKNOWLEDGED

Missing initialization of PausableUpgradeable Contract

Contract PendleStaking.sol, Contract mPendleConverter.sol, Contract PendleMarketDepositHelper.sol and Contract MasterPenPie, these contracts inherits `PausableUpgradeable`
which has an init method __Pausable_init that needs to be called in the `initialize()` to set the _paused to `false` state.

Recommendation:
Call the __Pausable_init method in the `initialize()` method.

INFORMATIONAL | RESOLVED

Reuse code by adding a modifier

In contract PendleMarketDepositHelper.sol, the internal methods _depositMarket(...) and _withdrawMarket(...) are performing the same following check

```
if (!poolInfo[_market].isActive) revert DeactivatePool();
```

Recommendation:
It is advised to add the following modifier and use it in the internal methods _depositMarket(...) and _withdrawMarket(...).

`modifier onlyActivePool(address market) {

```
if (!poolInfo[market].isActive) revert DeactivatePool();`
        _;
}`
```

Fix: Issue fixed in commit 4a8a2

INFORMATIONAL | ACKNOWLEDGED

Missing events in methods called only by Operator and Admin

In the contract PendleMarketDepositHelper.sol, the methods setPoolInfo(...), removePoolInfo(...), and setOperator(...) make updated but do not emit logs.

Recommendation:
It is advised to emit logs from the methods setPoolInfo(...), removePoolInfo(...) and setOperator(...).

INFORMATIONAL | RESOLVED

Wrong NatSpec comments

In contract MasterPenPie.sol, the method stakingInfo(...) missing return parameter Natspec comments.

In Contract PendleStaking.sol, the method setFee(...) NatSpec comment is missing parameters comments.

Recommendation:
Update the Natspec as per the methods.

Fix: Issue fixed in commit 4a8a2

INFORMATIONAL | RESOLVED

Use specific Solidity compiler version

Audited contracts use the following floating pragma:

```
pragma solidity ^0.8.0; pragma solidity ^0.8.19;
```

It allows to compile contracts with various versions of compiler and introduces the risk of using a different version when deploying than during testing.

Recommendation:
Use a specific version of the Solidity compiler.

Fix: Issue fixed in commit `0097c`

INFORMATIONAL | ACKNOWLEDGED

Upgradeable contracts not using upgradeable smart contract libraries

Contract PendleStaking.sol, Contract mPendleConverter.sol, Contract PendleMarketDepositHelper.sol, and Contract MasterPenPie uses OpenZeppelin contracts SafeERC20 and ERC20 which are not imported from the OpenZeppelin Upgradeable library as mentioned here in the OpenZeppelin documentation.

Recommendation:
Use this suggested method in OpenZeppelin's documentation to mitigate this issue.

Redundant use of `this` keyword in the function invocations

The convertPendle and convertAllPendles from PendleStaking contract use the `this` keyword keyword unnecessarily when invoking public functions. This results in the execution of a CALL opcode, which is gas expensive. When calling public functions within the same contract, there is no need to use this as the function calls can be made directly without the additional gas cost incurred by the CALL opcode.

Recommendation:
Do not use `this` keyword when invoking public functions within the same contract.

	MasterPenpie.sol PendleStaking.sol mPendleConvertor.sol mPendleOFT.sol PendleMarketDepositHelper.sol PenpieReceiptToken.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the Penpie team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Penpie team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo