AstraSec

Penpie

Security Audit Report

October 3, 2024

Contents

1 Introduction

1.1 About Penpie

Penpie is a DeFi platform integrated with Pendle Finance, offering users boosted rewards and governance participation without the need to lock their own PENDLE token. Penpie enhances yield across Pendle Finance's liquidity pools by locking PENDLE to gain vePendle. Additionally, users can lock PNP token on Penpie to gain vlPNP, which allows them to participate in Pendle Finance's governance and earn passive income through revenue sharing.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/penpie-contracts.git
- CommitID: f5a6682

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/magpiexyz/penpie-contracts.git
- CommitID: fc860a8

Note that this audit will cover the following contracts:

- `MasterPenpie.sol`, `VLPenpie.sol`, `BuyBackBurnProvider.sol`
- `ARBRewarder.sol`, `BaseRewardPoolV2.sol`, `mPendleSVBaseRewarder.sol`
- `vlPenpieBaseRewarder.sol`, `PenpieReceiptToken.sol`, `PendleMarketDepositHelper.sol`
- `PendleStaking.sol`, `PendleStakingBaseUpg.sol`, `PendleStakingBaseUpgBNB.sol`
- `PendleStakingSideChain.sol`, `PendleStakingSideChainBNB.sol`, `SmartPendleConvert.sol`

- `mPendleConvertor.sol`, `mPendleConvertorBaseUpg.sol`, `mPendleConvertorSideChain.sol`
- `mPendleSV.sol`, `zapInAndOutHelper.sol`, `PendleVoteManagerBaseUpg.sol`
- `PendleVoteManagerMainChain.sol`, `PendleVoteManagerSideChain.sol`, `PenpieBribeManager.sol`
- `PenpieBribeRewardDistributor.sol`, `ManualCompound.sol`, `PendleRushV6.sol`
- `mPendleOFT.sol`, and `PenpieOFT.sol`.

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `Penpie` protocol. Throughout this audit, we identified a total of 6 issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	-	-	-	-
Medium	4	1	-	3
Low	2	1	-	1
Informational	-	-	-	-
Undetermined	-	-	-	-

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~M-1~~ Revisited Logic of ETHZapper::swapExactTokensToETH()

~~M-2~~ Possible Reward Token Loss in ManualCompound::compound()

~~M-3~~ Improper initializer() Use in BNBPadding::_BNBPadding_init()

M-4 Potential Risks Associated with Centralization

~~L-1~~ Array Out-of-Bounds in vlPenpieBaseRewarder::rewardTokenInfosWithBribe()

L-2 Revisited Slippage Control in PendleRush6::_ZapInmPendleToMarket()

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Description
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

3.3.1 [M-1] Revisited Logic of swapExactTokensToETH()

Target	Category	IMPACT	LIKELIHOOD	STATUS
ETHZapper.sol	Business Logic	Medium	Medium	Addressed

The `ETHZapper::swapExactTokensToETH()` function is designed to swap a specified amount of `tokenIn` for native token (e.g., ETH). While examining the current implementation, we identify a vulnerability in the internal `_swapForEthUsingPancakeV2()` function called inside the `ETHZapper::swapExactTokens-ToETH()` function (line 72). It uses `getAmountsOut()` (line 86) to calculate the amount of swapped-out `ETH` based on the current state of the liquidity pool, making it vulnerable to front-running attacks. As a result, users may receive significantly less `ETH` than expected when `swapTokensForExactETH()` is executed (line 92). This flaw exposes users to potential asset loss due to insufficient slippage protection and the risk of price manipulation.

ETHZapper::swapExactTokensToETH()

```
55 function swapExactTokensToETH(
56     address tokenIn,
57     uint tokenAmountIn,
58     uint256 _amountOutMin,
59     address amountReciever
60 ) external {
61     if(fromTokenToDex[tokenIn] == 0) revert TokenNotSupported();
62     if(tokenAmountIn == 0) revert IsTokenAmountZero();
63     if(amountReciever == ADDRESS_ZERO) revert IsZeroAddressReciever();

65     IERC20(tokenIn).safeTransferFrom(msg.sender, address(this), tokenAmountIn);

67     if (fromTokenToDex[tokenIn] == BALANCERV2) {
68         _swapUsingBalancerV2(tokenIn, NATIVE, tokenAmountIn, _amountOutMin,
               amountReciever);
69     } else if (fromTokenToDex[tokenIn] == CAMELOTV2) {
70         _swapUsingCamelotV2(tokenIn, WETH, tokenAmountIn, _amountOutMin,
               amountReciever);
71     } else if (fromTokenToDex[tokenIn] == PANCAKEV3) {
72         _swapForEthUsingPancakeV2(tokenIn, WETH, tokenAmountIn, amountReciever);
73     }
74 }

76 function _swapForEthUsingPancakeV2(
77     address tokenIn,
78     address tokenOut,
79     uint tokenAmountIn,
80     address amountReciever
```

```
) internal {
    address[] memory path = new address[](2);
    path[0] = tokenIn;
    path[1] = tokenOut;

    uint[] memory amounts = pancakeRouter.getAmountsOut(
        tokenAmountIn, path
    );

    IERC20(tokenIn).safeApprove(address(pancakeRouter), tokenAmountIn);

    pancakeRouter.swapTokensForExactETH(
        amounts[1],
        type(uint256).max,
        path,
        amountReciever,
        block.timestamp + 1000
    );
}
```

Remediation Apply effective slippage control inside the `_swapForEthUsingPancakeV2()` function.

3.3.2 [M-2] Possible Reward Token Loss in compound()

Target	Category	IMPACT	LIKELIHOOD	STATUS
ManualCompound.sol	Business Logic	Medium	Medium	Addressed

The `compound()` function is designed to claim user rewards from `MasterPenpie` and re-compound them accordingly. While examining its implementation, we identify two potential scenarios that may result in reward tokens being left in the `ManualCompound` contract.

- If `isClaimPNP` is set to true but the `_rewards` array does not include the `PNP` token, the claimed `PNP` token will remain in the contract.
- If one of the reward tokens is configured to be compoundable, but it is neither `PENDLE` nor `PNP`, the reward token will also be left in the contract.

These issues can lead to rewards being left in the contract, preventing users from receiving their full expected rewards.

ManualCompound::compound()

```
function compound(
    address[] memory _lps,
    address[][] memory _rewards,
```

```
    bytes[] memory _kyBarExectCallData,
    address[] memory baseTokens,
    uint256[] memory compoundingMode,
    pendleDexApproxParams memory _pdexparams,
    bool isClaimPNP
) external {
    ...
    masterPenpie.multiclaimOnBehalf(
            _lps,
            _rewards,
            msg.sender,
            isClaimPNP
    );

    for (uint256 i; i < _lps.length;i++) {

        for (uint j; j < _rewards[i].length;j++) {

            address _rewardTokenAddress  = _rewards[i][j];
            uint256 receivedBalance = IERC20(_rewardTokenAddress).balanceOf(
                address(this)
            );

            if(receivedBalance == 0) continue;

            if (!compoundableRewards[_rewardTokenAddress]) {
                IERC20(_rewardTokenAddress).safeTransfer(
                    msg.sender,
                    receivedBalance
                );
                continue;
            }

            if (_rewardTokenAddress == PENDLE) {
                ...
            }
            else if (_rewardTokenAddress == PENPIE) {
                ...
            }
        }
    }

    if(userTotalPendleRewardToConvertMpendle != 0) _convertToMPendle(userTotalPendleRewardToConvertMpendle);
    if(userTotalPendleRewardToSendBack != 0 ) IERC20(PENDLE).safeTransfer( msg.sender, userTotalPendleRewardToSendBack );

    emit Compounded(msg.sender, _lps.length, _rewards.length);
}
```

Remediation Improve the implementation of the `compound()` function to ensure that all reward tokens are either compounded or transferred appropriately.

3.3.3 [M-3] Improper initializer() Use in _BNBPadding_init()

Target	Category	IMPACT	LIKELIHOOD	STATUS
BNBPadding.sol	Business Logic	High	Low	Addressed

While reviewing the implementation of the `BNBPadding` contract, we identify a vulnerability caused by the incorrect use of the `initializer()` modifier in the `_BNBPadding_init()` function. In earlier versions of OpenZeppelin, the `initializer()` modifier was functional in sub-level initialization functions. However, starting from OpenZeppelin version 4.4, this usage leads to a revert, as `initializer()` is intended exclusively for top-level initialization. The correct modifier for sub-level initialization functions like `_BNBPadding_init()` is `onlyInitializing()`. This mistake can cause failures during contract deployment or upgrades. Additionally, it is critical to maintain consistent use of the same OpenZeppelin version throughout the protocol to prevent potential storage conflicts and initialization issues.

PendleStakingSideChainBNB::__PendleStakingSideChain_init()

```
function __PendleStakingSideChain_init(
    address _pendle,
    address _WETH,
    address _vePendle,
    address _distributorETH,
    address _pendleRouter,
    address _masterPenpie
) public initializer {
    _BNBPadding_init();
    ...
}
```

BNBPadding::_BNBPadding_init()

```
function _BNBPadding_init() public initializer {
    __Ownable_init();
    __ReentrancyGuard_init();
    __Pausable_init();
}
```

Remediation Replace `initializer()` with `onlyInitializing()` in sub-initialization functions.

3.3.4 [M-4] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	High	Low	Acknowledged

In the `Penpie` protocol, the existence of a series of privileged accounts introduces centralization risks, as they hold significant control and authority over critical operations governing the protocol. In the following, we show the representative function potentially affected by the privileges associated with the privileged accounts.

Examples of Privileged Operations

```
function setPoolManagerStatus(
    address _account,
    bool _allowedManager
) external onlyOwner {
    PoolManagers[_account] = _allowedManager;

    emit PoolManagerStatus(_account, PoolManagers[_account]);
}

function setPenpie(address _penpieOFT) external onlyOwner {
    if (address(penpieOFT) != address(0)) revert PenpieOFTSetAlready();

    if (!Address.isContract(_penpieOFT)) revert MustBeContract();

    penpieOFT = IERC20(_penpieOFT);
    emit PenpieOFTSet(_penpieOFT);
}

function updateAllowedPauser(address _pauser, bool _allowed) external onlyOwner {
    allowedPauser[_pauser] = _allowed;

    emit UpdatePauserStatus(_pauser, _allowed);
}

function setCompounder(address _compounder)
    external
    onlyOwner
{
    address oldCompounder = compounder;
    compounder = _compounder;
    emit CompounderUpdated(compounder, oldCompounder);
}

function setVlPenpie(address _vlPenpie) external onlyOwner {
    address oldvlPenpie = address(vlPenpie);
    vlPenpie = IVLPenpie(_vlPenpie);
```

```
    emit VlPenpieUpdated(address(vlPenpie), oldvlPenpie);
}

function setMPendleSV(address _mPendleSV)
    external
    onlyOwner
{
    address oldMPendleSV = mPendleSV;
    mPendleSV = _mPendleSV;
    emit mPendleSVUpdated(_mPendleSV, oldMPendleSV);
}
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged accounts. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

Response By Team This issue has been confirmed by the team. The multi-sig mechanism will be used to mitigate this issue.

3.3.5 [L-1] Array Out-of-Bounds in rewardTokenInfosWithBribe()

Target	Category	IMPACT	LIKELIHOOD	STATUS
vlPenpieBaseRewarder.sol	Business Logic	Medium	Medium	Addressed

In the `vlPenpieBaseRewarder::rewardTokenInfosWithBribe()` function, the second `for` loop incorrectly accesses the `claimable` array using the index `i` (lines 239/241), which ranges from `rewardTokens.length` to `rewardTokensLength`. This leads to an out-of-bounds access when `i` exceeds the bounds of the `claimable` array, potentially causing the function to revert or return invalid data. The correct index for accessing the `claimable` array should be `i - rewardTokens.length`, ensuring proper alignment and preventing overflow issues.

vlPenpieBaseRewarder::rewardTokenInfosWithBribe()

```
217 function rewardTokenInfosWithBribe(
218     IBribeRewardDistributor.Claim[] calldata _proof
219 )
220     external
221     view
222     returns (
223         address[] memory bonusTokenAddresses,
224         string[] memory bonusTokenSymbols
225     )
226 {
```

```
    IBribeRewardDistributor.Claimable[] memory claimable =
        bribeRewardDistributor.getClaimable(_proof);
    uint256 rewardTokensLength = rewardTokens.length + claimable.length;
    bonusTokenAddresses = new address[](rewardTokensLength);
    bonusTokenSymbols = new string[](rewardTokensLength);
    for (uint256 i; i < rewardTokens.length; i++) {
        bonusTokenAddresses[i] = rewardTokens[i];
        bonusTokenSymbols[i] = IERC20Metadata(
            address(bonusTokenAddresses[i])
        ).symbol();
    }

    for (uint256 i = rewardTokens.length; i < rewardTokensLength; i++) {
        bonusTokenAddresses[i] = claimable[i].token;
        bonusTokenSymbols[i] = IERC20Metadata(
            address(claimable[i].token)
        ).symbol();
    }
}
```

Remediation Improve the implementation of the `rewardTokenInfosWithBribe()` function as above-mentioned.

3.3.6 [L-2] Revisited Slippage Control in _ZapInmPendleToMarket()

Target	Category	IMPACT	LIKELIHOOD	STATUS
PendleRush6.sol ManualCompound.sol	Coding Practices	Low	Low	Acknowledged

The `PendleRush6::_ZapInmPendleToMarket()` function facilitates providing liquidity to `Pendle Finance` in return for LP token, which is subsequently deposited into `Penpie` for yield generation. Upon reviewing the current implementation, we identify a vulnerability stemming from the absence of slippage control during the call to the `addLiquiditySingleToken()` (line 293) function. This lack of slippage control exposes the transaction to potential front-running attacks, potentially leading to unfavorable execution or reduced returns for users.

PendleRush6::_ZapInmPendleToMarket()

```
286 function _ZapInmPendleToMarket(
287     uint256 mPendleAmount,
288     pendleDexApproxParams memory _pdexparams
289 ) internal {
290     IERC20(mPENDLE).safeApprove(address(pendleRouter), 0);
291     IERC20(mPENDLE).safeApprove(address(pendleRouter), mPendleAmount);
```

```
    (uint256 netLpOut, ) = pendleRouter.addLiquiditySingleToken(
        address(this),
        mPendleMarket,
        type(uint256).min,
        IPendleRouter.ApproxParams(
            _pdexparams.guessMin,
            _pdexparams.guessMax,
            _pdexparams.guessOffChain,
            _pdexparams.maxIteration,
            _pdexparams.eps
        ),
        IPendleRouter.TokenInput(
            mPENDLE,
            mPendleAmount,
            mPENDLE,
            address(0),
            address(0),
            IPendleRouter.SwapData(IPendleRouter.SwapType.NONE, address(0), "0x",
                 false)
        )
    );
    IERC20(mPendleMarket).safeApprove(pendleStaking, netLpOut);
    IPendleMarketDepositHelper(pendleMarketDepositHelper).depositMarketFor(
        mPendleMarket,
        msg.sender,
        netLpOut
    );

    emit mPendleLiquidateToMarket(msg.sender, mPendleAmount);
}
```

Remediation Apply necessary slippage control in the `PendleRush6::_ZapInmPendleToMarket()` and `ManualCompound::_ZapInToPendleMarket()` functions.

Response By Team This issue has been confirmed by the team.

4 Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI