Penpie

Security Audit Report

May 21, 2024

Contents

1 | Introduction

1.1 About Penpie

`Penpie` is a next-generation DeFi platform designed to provide `Pendle Finance` users with yield and veTokenomics boosting services. Integrated with `Pendle Finance`, `Penpie` focuses on locking `PENDLE` tokens to obtain governance rights and enhanced yield benefits within `Pendle Finance`. `Penpie` revolutionizes the way users can maximize returns on their investments and monetize their governance power.

`Penpie` offers users the opportunity to deposit their assets to earn maximized APR while it allows `Pendle Finance` voters to cost-effectively acquire voting power and earn passive income at the same time through the `PNP` token.

1.2 Audit Scope

First Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/penpie-contracts/pull/119
- Commit ID: 7bd7b16

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

- Commit ID: 4a43404

Second Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/penpie-contracts/pull/119
- Commit ID: f00da12

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

- Commit ID: e98a2ad

Third Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/penpie-contracts/pull/124
- Commit ID: b115bd3

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

- Commit ID: f2bd5d0

Forth Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/penpie-contracts/pull/121
- Commit ID: 172a668

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

- https://github.com/magpiexyz/penpie-contracts/pull/139
- Commit ID: bd52148

Fifth Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/penpie-contracts/pull/147
- Commit ID: 24d3226

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

- Commit ID: 24d3226

Sixth Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/penpie-contracts/pull/153
- Commit ID: b233679

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

- Commit ID: b726c05

Seventh Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/penpie-contracts/pull/155
- Commit ID: 2ff9b44

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

- Commit ID: dc3f65a

1.3 Changelog

Version	Date
First Audit	December 28, 2023
Second Audit	January 28, 2024
Third Audit	February 7, 2024
Forth Audit	April 3, 2024
Fifth Audit	May 2, 2024
Sixth Audit	May 8, 2024
Seventh Audit	May 21, 2024

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `Penpie` project. Throughout this audit, we identified several 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	1	-	-	1
Medium	4	1	-	3
Low	1	-	-	1
Informational	3	-	-	3
Undetermined	-	-	-	-

3 Vulnerability Summary

3.1 Overview

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

- ~~H-1~~ Inadequate Validation of User Input in registerPenpiePool()/addPenpieBribePool()
- ~~M-1~~ Revised ARB Harvest for vlPenpie&mPendleSV in MasterPenpie
- M-2 Potential Risks Associated with Centralization
- ~~M-3~~ Improper Reward Update Logic in ARBRewarder::setPool()
- ~~M-4~~ Revisited Logic of MasterPenpie::_harvestBaseRewarder()
- ~~L-1~~ Revisited Update of Active Pool in ARBRewarder
- ~~I-1~~ Meaningful Events for Key Operations
- ~~I-2~~ Improved Logic in ARBRewarder::massUpdatePools()
- ~~I-3~~ Improved Logic of quoteConvert()

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

[H-1] Inadequate Validation of User Input in registerPenpiePool()/addPenpieBribePool()

Target	Category	IMPACT	LIKELIHOOD	STATUS
PendleMarketRegisterHelper.sol	Business Logic	High	Medium	Addressed

The `registerPenpiePool()` function facilitates the permissionless registration of `Pendle Market LP` token on `Penpie`. That is to say, anyone can use this function to register a valid `Pendle Market LP` token on `Penpie`. Upon thorough examination of the current implementation, we observe that it only verifies the input `_market` parameter but fails to validate other parameters, particularly `_allocPoints` . This oversight is critical as it directly affects reward distribution. Without proper validation, a malicious actor could manipulate `_allocPoints` to unfairly increase their reward allocation.

PendleMarketRegisterHelper::registerPenpiePool()

```
function registerPenpiePool(
    address _market,
    uint256 _allocPoints,
    string memory name,
    string memory symbol
) external {
    _registerMarket(_market, _allocPoints, name, symbol);
}

function _registerMarket(
    address _market,
    uint256 _allocPoints,
    string memory name,
    string memory symbol
) internal onlyVerifiedMarket(_market) nonReentrant {
    IPendleStaking(pendleStaking).registerPool(
        _market,
        _allocPoints,
        name,
        symbol
    );

    emit NewMarketAdded(_market, _allocPoints, name, symbol);
}
```

Remediation Thoroughly validate all user input parameters in the `registerPenpiePool()/addPenpieBribePool()` functions.

[M-1] Revised ARB Harvest for vlPenpie&mPendleSV in MasterPenpie

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

The `MasterPenpie` contract accepts the deposit of the supported assets and rewards the depositor with reward tokens like `Penpie`, `ARB` and so on. While examining the deposit of `vlPenpie` and `mPendleSV`, we notice it harvests `ARB` for the depositor but doesn't update the `rewardDebt` accordingly in `ARBRewarder` .

MasterPenpie::_deposit()

```
function _deposit(...
) internal {
    PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
    UserInfo storage user = userInfo[_stakingToken][_for];

    updatePool(_stakingToken);
    _harvestRewards(_stakingToken, _for);

    user.amount = user.amount + _amount;
    if (!_isLock) {...}
    user.rewardDebt = (user.amount * pool.accPenpiePerShare) / 1e12;
    ...
}
```

Moreover, while reviewing the withdrawal of `vlPenpie` and `mPendleSV` in `MasterPenpie`, we notice it doesn't harvest `ARB` from `ARBRewarder` for the depositor. As a result, the depositors of `vlPenpie` and `mPendleSV` in `MasterPenpie` will not receive any ARB reward.

ARBRewarder::_withdraw()

```
function _withdraw(...
  ) internal {
      PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
      UserInfo storage user = userInfo[_stakingToken][_account];

      if (!_isLock && user.available < _amount)
          revert WithdrawAmountExceedsStaked();
      else if (user.amount < _amount && _isLock)
          revert UnlockAmountExceedsLocked();

      updatePool(_stakingToken);
      _harvestPenpie(_stakingToken, _account);
      _harvestBaseRewarder(_stakingToken, _account);
```

```
        user.amount = user.amount - _amount;
        if (!_isLock) {...}
        user.rewardDebt = (user.amount * pool.accPenpiePerShare) / 1e12;

        pool.totalStaked -= _amount;

        emit Withdraw(_account, _stakingToken, pool.receiptToken, _amount);
    }
```

Remediation Properly update the `rewardDebt` of `ARBRewarder` in `MasterPenpie::_deposit()` and harvest `ARB` in `MasterPenpie::_withdraw()`.

[M-2] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
ARBRewarder.sol	Security	Medium	Medium	Acknowledged

In the `ARBRewarder` protocol, the existence of a privileged account introduces centralization risks, as it holds significant control and authority over critical operations governing the protocol. In the following, we show the representative functions potentially affected by the privileges associated with the privileged account.

ARBRewarder

```
function updateEmissionRate(uint256 _ARBPerSec) public onlyOwner {
    massUpdatePools();
    uint256 oldEmissionRate = ARBPerSec;
    ARBPerSec = _ARBPerSec;

    emit UpdateEmissionRate(msg.sender, oldEmissionRate, ARBPerSec);
}
function addPools(
    address[] calldata _stakingToken,
    uint256[] calldata _allocPoint,
    address[] calldata _masterChefs
) external onlyOwner {

    massUpdatePools();
    if(_stakingToken.length != _allocPoint.length   _stakingToken.length !=
        _masterChefs.length)
        revert LengthMismatch();

    for(uint256 index = 0; index < _stakingToken.length; index++){
        _addPool(_stakingToken[index], _allocPoint[index], _masterChefs[index]);
    }
```

```
}
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged account. 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.

[M-3] Improper Reward Update Logic in ARBRewarder::setPool()

Target	Category	IMPACT	LIKELIHOOD	STATUS
ARBRewarder.sol	Business Logic	Medium	Low	Addressed

In the `ARBRewarder` contract, the `setPool()` function allows the privileged account, i.e., `owner`, to change the status and modify the `masterChef` address for an existing pool. Our analysis shows its current reward update logic is incorrect.

To elaborate, we show below its code snippet. It comes to our attention that when the status of an existing pool transitions from inactive to active, there is a lack of update for the pool's `lastRewardTimestamp`. As a result, the pool will receive more `ARB` rewards than expected.

ARBRewarder::setPool()

```
    function setPool(address _stakingToken, address _masterChef, bool _isActive) external onlyOwner {
        PoolInfo storage pool = tokenToPoolInfo[_stakingToken];

        if(!_isActive){
            address rewarder = IMasterPenpie(pool.masterChef).getRewarder(_stakingToken);
            _calculateAndSendARB(_stakingToken, rewarder);
        }
        pool.masterChef = _masterChef;
        pool.isActive = _isActive;

        emit SetPool(_stakingToken, _masterChef, _isActive);
    }
```

Remediation Timely update the `lastRewardTimestamp` for an existing pool when the pool transitions from inactive to active. An example revision is shown as follows:

ARBRewarder::setPool()

```
    function setPool(address _stakingToken, address _masterChef, bool _isActive)
         external onlyOwner {
        PoolInfo storage pool = tokenToPoolInfo[_stakingToken];

        if (pool.isActive) {
            address rewarder = IMasterPenpie(pool.masterChef).getRewarder(
                _stakingToken);
            _calculateAndSendARB(_stakingToken, rewarder);
        } else if (_isActive) {
            pool.lastRewardTimestamp = block.timestamp;
        }
        pool.masterChef = _masterChef;
        pool.isActive = _isActive;

        emit SetPool(_stakingToken, _masterChef, _isActive);
    }
```

[M-4] Revisited Logic of MasterPenpie::_harvestBaseRewarder()

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Business Logic	Medium	Medium	Addressed

The MasterPenpie contract provides an internal _harvestBaseRewarder function to update the reward counting in the base rewarder. While reviewing its logic, we notice this function needs to be revisited.

In the following, we show the related code snippet. Specifically, the rewarder address for a staking pool can be configured as address(0). In this case, the execution of the current function will revert (lines 777-782).

MasterPenpie::_harvestBaseRewarder(address, address)

```
772     /// only update the reward counting on in base rewarder but not sending them
             to user
773     function _harvestBaseRewarder(
774         address _stakingToken,
775         address _account
776     ) internal {
777         IBaseRewardPool rewarder = IBaseRewardPool(
778             tokenToPoolInfo[_stakingToken].rewarder
779         );

781         if(address(ARBRewarder) != address(0))
782             IARBRewarder(ARBRewarder).harvestARB(_stakingToken, address(rewarder
                ));
```

```
        if (address(rewarder) != address(0)) rewarder.updateFor(_account);

    }
```

Note a number of functions share the similar issue, including `MasterPenpie::setARBRewarderAsQueuer()`, `MasterRadpie::_harvestRewards()`, and `RadiantStaking::updateRewardQueuers()`.

Remediation Perform necessary validity checks on the obtained `rewarder` address.

[L-1] Revisited Update of Active Pool in ARBRewarder

Target	Category	IMPACT	LIKELIHOOD	STATUS
ARBRewarder.sol	Business Logic	Low	Low	Addressed

The `ARBRewarder` contract provides an external `updatePoolsAlloc()` function for the privileged `owner` account to update the allocation points for the staking pools specified by the `owner`. Our analysis shows the logic can be improved by applying a more rigorous sanity check. Specifically, the updates should be only applied to staking pools that are in active state. Updating an inactive pool may increase the `totalAllocPoint`, thereby affecting the reward distribution to active pools.

ARBRewarder::updatePoolsAlloc()

```
    function updatePoolsAlloc(
        address[] calldata _stakingTokens,
        uint256[] calldata _allocPoints
    ) external onlyOwner {

        if (_stakingTokens.length != _allocPoints.length)
            revert LengthMismatch();
        massUpdatePools();
        for (uint256 i = 0; i < _stakingTokens.length; i++) {

            uint256 oldAllocPoint = tokenToPoolInfo[_stakingTokens[i]].allocPoint
                ;
            totalAllocPoint = totalAllocPoint - oldAllocPoint + _allocPoints[i];
            tokenToPoolInfo[_stakingTokens[i]].allocPoint = _allocPoints[i];

            emit UpdatePoolAlloc(
                _stakingTokens[i],
                oldAllocPoint,
                _allocPoints[i]
            );
        }
    }
```

Remediation Allow the update of allocation points for active staking pools only.

[I-1] Meaningful Events for Key Operations

Target	Category	IMPACT	LIKELIHOOD	STATUS
MasterPenpie.sol	Coding Practices	N/A	N/A	Addressed

The `event` feature is vital for capturing runtime dynamics in a contract. Upon emission, `events` store transaction arguments in logs, supplying external analytics and reporting tools with crucial information. They play a pivotal role in scenarios like modifying system-wide parameters or handling token operations.

However, in our examination of protocol dynamics, we observed that certain privileged routines lack meaningful events to document their changes. We highlight the representative routines below.

MasterPenpie

```
function setARBRewarder(address _ARBRewarder) external onlyOwner{
    ARBRewarder = _ARBRewarder;
}

function addPoolsForARBIncentive(address stakingToken) external {
    if(msg.sender != ARBRewarder)
        revert onlyARBRewarder();

    isARBIncentivePool[stakingToken] = true;
}
```

Note this issue is also applicable to other contracts, including `MasterRadpie` and `MasterMagpie`.

Remediation Ensure the proper emission of meaningful events containing accurate information to promptly reflect state changes.

[I-2] Improved Logic in ARBRewarder::massUpdatePools()

Target	Category	IMPACT	LIKELIHOOD	STATUS
ARBRewarder.sol	Coding Practices	N/A	N/A	Addressed

In the `ARBRewarder` contract, the public `massUpdatePools()` function allows to calculate and queue `ARB` rewards for multiple pools simultaneously. While reviewing its implementation, we notice that it can benefit from additional validity checks.

To elaborate, we show below the full implementation of the `massUpdatePools()` function. Specifically, if `pool.isActive == false` or `block.timestamp == pool.lastRewardTimestamp`, the execution of `_calculateAndSendARB()` for this pool is not necessary and just a waste of gas.

ARBRewarder::massUpdatePools()

```
    function massUpdatePools() public whenNotPaused {

        for (uint256 pid = 0; pid < registeredPools.length; ++pid) {
            address stakingToken = registeredPools[pid];
            PoolInfo memory pool = tokenToPoolInfo[stakingToken];
            if(pool.ARBPerSec == 0)
                continue;
            address rewarder = IMasterPenpie(pool.masterChef).getRewarder(
                stakingToken);
            _calculateAndSendARB(stakingToken, rewarder);
        }
    }
```

Remediation Add necessary validity checks of an existing pool for the above mentioned function.

[I-3] Improved Logic of quoteConvert()

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Coding Practices	N/A	N/A	Addressed

In the `WomUp5` contract, the `quoteConvert()` function serves as a query mechanism for users seeking to determine the amount of `ARB` token that can be acquired through the conversion of a specified quantity of `WOM` token. When scrutinizing its logic, we notice that when the length of the `rewardTier` array is zero, the function depends solely on the implicit array bound-checks generated by the compiler, lacking of explicit sanity check.

To fortify robustness and security, it is advisable to implement explicit sanity check at the beginning of the function to ensure that the length of the `rewardTier` array is not zero.

WomUp5::quoteConvert()

```
function quoteConvert(
    uint256 _amountToConvert,
    address _account
) external view returns (uint256) {
    UserInfo memory userInfo = userInfos[_account];
    uint256 arbReward = 0;

    uint256 accumulatedRewards = _amountToConvert + userInfo.converted;
    uint256 i = 1;
```

```
    while (i < rewardTier.length && accumulatedRewards > rewardTier[i]) {
        arbReward += (rewardTier[i] - rewardTier[i - 1]) * rewardMultiplier[i -
            1];
        i++;
    }

    arbReward += (accumulatedRewards - rewardTier[i - 1]) * rewardMultiplier[i -
         1];

    arbReward = (arbReward / DENOMINATOR) - userInfo.rewardClaimed;
    uint256 arbleft = ARB.balanceOf(address(this));

    uint256 finalReward = arbReward > arbleft ? arbleft : arbReward;
    return finalReward;
}
```

Remediation Apply explicit sanity checks on the length of the `rewardTier` array in the `PendleRush6::quoteConvert()`, `WomUp5::quoteConvert()`, and `DlpRush2::quoteConvert()` functions.

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	+86 176 2267 4194
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI