Penpie

Security Audit Report

July 3, 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

Round	LINK	Base Commit	Final Commit
1	https://github.com/magpiexyz/penpie-contracts/pull/119	7bd7b16	4a43404
2	https://github.com/magpiexyz/penpie-contracts/pull/119	f00da12	e98a2ad
3	https://github.com/magpiexyz/penpie-contracts/pull/124	b115bd3	f2bd5d0
4	https://github.com/magpiexyz/penpie-contracts/pull/121	172a668	bd52148
5	https://github.com/magpiexyz/penpie-contracts/pull/147	24d3226	24d3226
6	https://github.com/magpiexyz/penpie-contracts/pull/153	b233679	b726c05
7	https://github.com/magpiexyz/penpie-contracts/pull/155	2ff9b44	dc3f65a
8	https://github.com/magpiexyz/penpie-contracts/pull/117	37bbd07	2ac44f2
9	https://github.com/magpiexyz/penpie-contracts/pull/167	6ea774f	702d547
10	https://github.com/magpiexyz/penpie-contracts/pull/158	14657ea	b0d51d6

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
Eighth Audit	June 8, 2024
Ninth Audit	June 29, 2024
Tenth Audit	June 30, 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	5	1	-	4
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.

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()

```
73  function registerPenpiePool(
74      address _market,
75      uint256 _allocPoints,
76      string memory name,
77      string memory symbol
78  ) external {
79      _registerMarket(_market, _allocPoints, name, symbol);
80  }

82  function _registerMarket(
83      address _market,
84      uint256 _allocPoints,
85      string memory name,
86      string memory symbol
87  ) internal onlyVerifiedMarket(_market) nonReentrant {
88      IPendleStaking(pendleStaking).registerPool(
89          _market,
90          _allocPoints,
91          name,
92          symbol
93      );

95      emit NewMarketAdded(_market, _allocPoints, name, symbol);
96  }
```

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()

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

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

618        user.amount = user.amount + _amount;
619        if (!_isLock) {...}
620        user.rewardDebt = (user.amount * pool.accPenpiePerShare) / 1e12;
621        ...
622    }
```

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()

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

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

653          updatePool(_stakingToken);
654          _harvestPenpie(_stakingToken, _account);
655          _harvestBaseRewarder(_stakingToken, _account);
```

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

661         pool.totalStaked -= _amount;

663         emit Withdraw(_account, _stakingToken, pool.receiptToken, _amount);
664     }
```

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

```
381 function updateEmissionRate(uint256 _ARBPerSec) public onlyOwner {
382     massUpdatePools();
383     uint256 oldEmissionRate = ARBPerSec;
384     ARBPerSec = _ARBPerSec;

386     emit UpdateEmissionRate(msg.sender, oldEmissionRate, ARBPerSec);
387 }
388 function addPools(
389     address[] calldata _stakingToken,
390     uint256[] calldata _allocPoint,
391     address[] calldata _masterChefs
392 ) external onlyOwner {

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

398     for(uint256 index = 0; index < _stakingToken.length; index++){
399         _addPool(_stakingToken[index], _allocPoint[index], _masterChefs[index]);
400     }
```

```
401 }
```

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()

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

199         if(!_isActive){
200             address rewarder = IMasterPenpie(pool.masterChef).getRewarder(
                    _stakingToken);
201             _calculateAndSendARB(_stakingToken, rewarder);
202         }
203         pool.masterChef = _masterChef;
204         pool.isActive = _isActive;

206         emit SetPool(_stakingToken, _masterChef, _isActive);
207     }
```

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()

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

199         if (pool.isActive) {
200             address rewarder = IMasterPenpie(pool.masterChef).getRewarder(
                    _stakingToken);
201             _calculateAndSendARB(_stakingToken, rewarder);
202         } else if (_isActive) {
203             pool.lastRewardTimestamp = block.timestamp;
204         }
205         pool.masterChef = _masterChef;
206         pool.isActive = _isActive;

208         emit SetPool(_stakingToken, _masterChef, _isActive);
209     }
```

[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
                    ));
```

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

786         }
```

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.

[M-5] Revisited Logic of mPendleSV::relock()

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

The `relock()` function is designed to relock `mPendle` tokens if they are not withdrawn within a specified period. The logic to enforce this is implemented by comparing `slot.endTime + reLockTimeLimitInSecs` and `block.timestamp`. Specifically, relocking is allowed only if `block.timestamp` is greater than `slot.endTime + reLockTimeLimitInSecs`. However, the current implementation prevents relocking if the condition (`(slot.endTime + reLockTimeLimitInSecs)< block.timestamp`) is met (line 342), which clearly violates the original design.

Additionally, considering that the user's `mPendle` tokens in the `masterPenpie` contract do not change as a result of the relock operation, there is no need to execute `withdrawMPendleSVFor()` (line 344) followed by `depositMPendleSVFor()` (line 348).

mPendleSV::relock()

```
327 function relock( address[] memory _users, uint256[] memory _slotIndex) external
        whenNotPaused {
328     uint256 usersLength = _users.length;

330     if(usersLength != _slotIndex.length)
331         revert InvalideArrayLength();

333     for(uint256 i = 0; i < usersLength; i++)
334     {
335         _checkIdexInBoundary(_users[i], _slotIndex[i]);

337         UserUnlocking storage slot = userUnlockings[_users[i]][_slotIndex[i]];
338         if (slot.amountInCoolDown == 0) revert UnlockedAlready();

340         uint256 amountForReLock = slot.amountInCoolDown;
```

```
342             if ((slot.endTime + reLockTimeLimitInSecs) < block.timestamp) revert
                    CanNotRelockBeforeRelockTimeDuration();

344             IMasterPenpie(masterPenpie).withdrawMPendleSVFor(amountForReLock,
                    _users[i]); // trigers update pool share, so happens before total
                    amount reducing
345             totalAmountInCoolDown -= amountForReLock;
346             slot.amountInCoolDown = 0; // not in cool down anymore that mean it's on
                     lock state

348             IMasterPenpie(masterPenpie).depositMPendleSVFor(amountForReLock,  _users
                    [i]);
349             emit ReLock(_users[i], _slotIndex[i], amountForReLock);
350         }
351     }
```

Remediation Correct the implementation of the `relock()` function as above-mentioned.

[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()

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

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

420             uint256 oldAllocPoint = tokenToPoolInfo[_stakingTokens[i]].allocPoint
                    ;
421             totalAllocPoint = totalAllocPoint - oldAllocPoint + _allocPoints[i];
422             tokenToPoolInfo[_stakingTokens[i]].allocPoint = _allocPoints[i];
```

```
424                 emit UpdatePoolAlloc(
425                     _stakingTokens[i],
426                     oldAllocPoint,
427                     _allocPoints[i]
428                 );
429             }
430         }
```

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

```
1103 function setARBRewarder(address _ARBRewarder) external onlyOwner{
1104     ARBRewarder = _ARBRewarder;
1105 }

1107 function addPoolsForARBIncentive(address stakingToken) external {
1108     if(msg.sender != ARBRewarder)
1109         revert onlyARBRewarder();

1111     isARBIncentivePool[stakingToken] = true;
1112 }
```

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()

```
92      function massUpdatePools() public whenNotPaused {
94          for (uint256 pid = 0; pid < registeredPools.length; ++pid) {
95              address stakingToken = registeredPools[pid];
96              PoolInfo memory pool = tokenToPoolInfo[stakingToken];
97              if(pool.ARBPerSec == 0)
98                  continue;
99              address rewarder = IMasterPenpie(pool.masterChef).getRewarder(
                    stakingToken);
100             _calculateAndSendARB(stakingToken, rewarder);
101         }
102     }
```

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()

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

102     uint256 accumulatedRewards = _amountToConvert + userInfo.converted;
103     uint256 i = 1;

105     while (i < rewardTier.length && accumulatedRewards > rewardTier[i]) {
106         arbReward += (rewardTier[i] - rewardTier[i - 1]) * rewardMultiplier[i -
                1];
107         i++;
108     }

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

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

115     uint256 finalReward = arbReward > arbleft ? arbleft : arbReward;
116     return finalReward;
117 }
```

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