BLOCKSEC

Security Audit

Report for Penpie contracts

Date: October 2, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Magpiexyz
Target	Penpie contracts

Version History

Version	Date	Description
1.0	October 2, 2024	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2024-10-04 00:53:54+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

This audit [1] focuses on the Penpie contracts contract for Magpiexyz [2]. 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. Specifically, only the following contrats in the repository are included in the scope of this audit. Other files are not within the scope of this audit.

- contracts/rewards/MasterPenpie.sol
- contracts/VLPenpie.sol
- contracts/BuyBackBurnProvider.sol
- contracts/rewards/ARBRewarder.sol
- contracts/rewards/BaseRewardPoolV2.sol
- contracts/rewards/mPendleSVBaseRewarder.sol
- contracts/rewards/vlPenpieBaseRewarder.sol
- contracts/rewards/PenpieReceiptToken.sol
- contracts/pendle/PendleMarketDepositHelper.sol
- contracts/pendle/PendleStaking.sol
- contracts/pendle/PendleStakingBaseUpg.sol
- contracts/pendle/PendleStakingBaseUpgBNB.sol
- contracts/pendle/PendleStakingSideChain.sol
- contracts/pendle/PendleStakingSideChainBNB.sol
- contracts/pendle/SmartPendleConvert.sol
- contracts/pendle/mPendleConvertor.sol
- contracts/pendle/mPendleConvertorBaseUpg.sol
- contracts/pendle/mPendleConvertorSideChain.sol
- contracts/pendle/mPendleSV.sol
- contracts/pendle/zapInAndOutHelper.sol
- contracts/bribeMarket/PendleVoteManagerBaseUpg.sol
- contracts/bribeMarket/PendleVoteManagerMainChain.sol
- contracts/bribeMarket/PendleVoteManagerSideChain.sol
- contracts/bribeMarket/PenpieBribeManager.sol

[1] The audit process is not yet complete and remains ongoing. This report is preliminary, and additional findings may emerge in subsequent stages of the audit.

[2] https://github.com/magpiexyz/penpie-contracts

- contracts/bribeMarket/PenpieBribeRewardDistributor.sol
- contracts/rewards/ManualCompound.sol
- contracts/pendle/PendleRushV6.sol
- contracts/pendle/mPendleOFT.sol
- contracts/PenpieOFT.sol
- contracts/PenpieOFT.sol
- contracts/libraries/ERC20FactoryLib.sol
- contracts/libraries/UtilLib.sol
- libraries/WeekMath.sol
- pendle/BNBPadding.sol
- libraries/math/Math.sol
- libraries/layerZero/LayerZeroHelper.sol

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
Penpie contracts	`Version 1`	`f5a6682c301fad7358fe7ce02cfef3e710f66a6e`
	`Version 2`	`fc860a8f79123f24adb8e8565e743d3c9dde0a00`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist
* Economic impact
* Batch transfer

1.3.3 NFT Security

* Duplicated item
* Verification of the token receiver
* Off-chain metadata security

1.3.4 Additional Recommendation

* Gas optimization

* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [3] and Common Weakness Enumeration [4]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

[3] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[4] https://cwe.mitre.org/

Chapter 2 Findings

In total, we found **three** potential security issues. Besides, we have **three** recommendations and **ten** notes.

- Medium Risk: 1
- Low Risk: 2
- Recommendation: 3
- Note: 10

ID	Severity	Description	Category	Status
1	Medium	Lack of logic on handling specific reward tokens in function `compound`	Software Security	Fixed
2	Low	Potential DoS due to arbitrarily added markets in function `addPenpieBribePool`	Software Security	Fixed
3	Low	Lack of harvesting pool when the `allocPoint` is changed	DeFi Security	Fixed
4	-	Remove unused logic in `_deposit` and `_withdraw` in `MasterPenpie`	Recommendation	Fixed
5	-	Remove redundant checks in `ARBRewarder`	Recommendation	Fixed
6	-	Avoid multiplying after division	Recommendation	Confirmed
7	-	Centralization risks	Note	-
8	-	`MannualCompound` must not hold any token	Note	-
9	-	Token prices returned by `PenpieReader` can be inaccurate	Note	-
10	-	`PendleRushV6` must not hold mPendle	Note	-
11	-	Users can donate Pendle to `PendleStaking` via function `convertPendle`	Note	-
12	-	`PendleStaking`'s Pendle locked in `vePendle` can be locked permanently by anyone	Note	-
13	-	Precision loss in function `updatePool` is negligible	Note	-
14	-	`queuedRewards` will be distributed to the first depositor	Note	-
15	-	`penpieReward` should not be distributed to empty pools	Note	-
16	-	The protocol will avoid potential lock or draining of rewards for Pendle market	Note	-

The details are provided in the following sections.

2.1 Software Security

2.1.1 Lack of logic on handling specific reward tokens in function `compound`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `compound` in the contract `ManualCompound` collects rewards from `MasterPenpie` by invoking the function `multiclaimOnBehalf()`. However, the function only handles PENDLE or PENPIE tokens afterward. If other reward tokens are claimed from `MasterPenpie`, those rewards can be locked in contract `ManualCompound`.

```
207 function compound(
208     address[] memory _lps,
209     address[][] memory _rewards,
210     bytes[] memory _kyBarExectCallData,
211     address[] memory baseTokens,
212     uint256[] memory compoundingMode,
213     pendleDexApproxParams memory _pdexparams,
214     bool isClaimPNP
215  ) external {
216
217
218     if(_rewards.length != _lps.length) revert InputDataLengthMissMatch();
219     if(baseTokens.length != _kyBarExectCallData.length) revert InputDataLengthMissMatch();
220
221
222     uint256 userTotalPendleRewardToSendBack;
223     uint256 userTotalPendleRewardToConvertMpendle;
224     uint256[] memory userPendleRewardsForCurrentMarket = new uint256[](_lps.length);
225
226
227     for(uint256 k; k < _lps.length;k++)
228     {
229         (,,,userPendleRewardsForCurrentMarket[k]) = masterPenpie.pendingTokens(_lps[k], msg.
                sender, PENDLE);
230     }
231
232     if(compoundingMode.length != userPendleRewardsForCurrentMarket.length) revert
            InputDataLengthMissMatch();
233
234
235     masterPenpie.multiclaimOnBehalf(
236             _lps,
237             _rewards,
238             msg.sender,
239             isClaimPNP
240     );
241
242     for (uint256 i; i < _lps.length;i++) {
243
```

```
244             for (uint j; j < _rewards[i].length;j++) {
245
246                 address _rewardTokenAddress = _rewards[i][j];
247                 uint256 receivedBalance = IERC20(_rewardTokenAddress).balanceOf(
248                     address(this)
249                 );
250
251
252                 if(receivedBalance == 0) continue;
253
254
255                 if (!compoundableRewards[_rewardTokenAddress]) {
256                         IERC20(_rewardTokenAddress).safeTransfer(
257                             msg.sender,
258                             receivedBalance
259                         );
260                         continue;
261                 }
262
263
264                 if (_rewardTokenAddress == PENDLE) {
265                     if(compoundingMode[i] == LIQUIDATE_TO_PENDLE_FINANCE)
266                     {
267                         IERC20(PENDLE).safeApprove(address(pendleRouter),
                                userPendleRewardsForCurrentMarket[i]);
268                         _ZapInToPendleMarket(userPendleRewardsForCurrentMarket[i], _lps[i],
                                baseTokens[i], _kyBarExectCallData[i], _pdexparams );
269                     }
270                     else if( compoundingMode[i] == CONVERT_TO_MPENDLE )
271                     {
272                         userTotalPendleRewardToConvertMpendle += userPendleRewardsForCurrentMarket
                                [i];
273                     }
274                     else
275                     {
276                         userTotalPendleRewardToSendBack += userPendleRewardsForCurrentMarket[i];
277                     }
278                 }
279                 else if (_rewardTokenAddress == PENPIE) {
280                         _lockPenpie(receivedBalance);
281                 }
282             }
283     }
284
285     if(userTotalPendleRewardToConvertMpendle != 0) _convertToMPendle(
            userTotalPendleRewardToConvertMpendle);
286     if(userTotalPendleRewardToSendBack != 0 ) IERC20(PENDLE).safeTransfer( msg.sender,
            userTotalPendleRewardToSendBack );
287
288     emit Compounded(msg.sender, _lps.length, _rewards.length);
289 }
```

Listing 2.1: contracts/rewards/ManualCompound.sol

Impact Potential lock of rewards.

Suggestion Add the logic to handle other reward tokens.

2.1.2 Potential DoS due to arbitrarily added markets in function `addPenpieBribePool`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Currently, the function `addPenpieBribePool()` can be invoked by anyone to add any markets in `penpieBribeManager`. Thus, this would lead to two problems. First, an evil market can be added in `penpieBribeManager`, which is a potential risk. Second, a malicious user can add a large amount of markets that will cause denial of service due to exceeding gas limits in the loop.

```
81  function addPenpieBribePool(
82      address _market
83  ) external {
84      _newPool(_market);
85  }
```

Listing 2.2: contracts/pendle/PendleMarketRegisterHelper.sol

```
468  function newPool(address _market, uint16 _chainId) external _onlyPoolRegisterHelper {
469     if (_market == address(0)) revert ZeroAddress();
470
471
472     for (uint256 i = 0; i < pools.length; i++) {
473         if (pools[i]._market == _market) {
474             revert MarketExists();
475         }
476     }
477
478
479     Pool memory pool = Pool(_market, true, _chainId);
480     pools.push(pool);
481
482
483     marketToPid[_market] = pools.length - 1;
484
485
486     IPendleVoteManager(voteManager).addPool(_market, _chainId);
487
488
489     emit NewPool(_market, _chainId);
490 }
```

Listing 2.3: contracts/bribeMarket/PenpieBribeManager.sol

Impact First, an evil market can be added in `penpieBribeManager`, which is a potential risk. Second, a malicious user can add a large number of markets that will cause denial of service due to exceeding gas limits in the loop.

Suggestion Change the function `addPenpieBribePool()` to a privileged function.

2.2 DeFi Security

2.2.1 Lack of harvesting pool when the `allocPoint` is changed

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Currently, when the `allocPoint` of a specific pool is changed, the pool is not harvested. In this case, when the pool is harvested next time, the Penpie reward is calculated with the new `allocPoint`. However, the reward farmed before the change of `allocPoint` should be calculated with the original `allocPoint`.

```
1004    function set(
1005        address _stakingToken,
1006        uint256 _allocPoint,
1007        address _rewarder,
1008        bool _isActive
1009    ) external _onlyPoolManager {
1010        if (
1011            !Address.isContract(address(_rewarder)) &&
1012            address(_rewarder) != address(0)
1013        ) revert MustBeContractOrZero();
1014
1015        if (!tokenToPoolInfo[_stakingToken].isActive) revert OnlyActivePool();
1016
1017        // massUpdatePools();
1018
1019        totalAllocPoint =
1020            totalAllocPoint -
1021            tokenToPoolInfo[_stakingToken].allocPoint +
1022            _allocPoint;
1023
1024        tokenToPoolInfo[_stakingToken].allocPoint = _allocPoint;
1025        tokenToPoolInfo[_stakingToken].rewarder = _rewarder;
1026        tokenToPoolInfo[_stakingToken].isActive = _isActive;
1027
1028        emit Set(
1029            _stakingToken,
1030            _allocPoint,
1031            IBaseRewardPool(tokenToPoolInfo[_stakingToken].rewarder),
1032            _isActive
1033        );
1034    }
```

Listing 2.4: contracts/rewards/MasterPenpie.sol

Impact The harvested Penpie reward can be inaccurate.

Suggestion Update the pool when its `allocPoint` is changed.

2.3 Additional Recommendation

2.3.1 Remove unused logic in `_deposit` and `_withdraw` in `MasterPenpie`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `_deposit` and `_withdraw` function in `MasterPenpie` contract accepts a `_isLock` flag to determine whether there should be actual token transfers during processing.

However, this feature seems to be deprecated because all invocations to these two internal functions assign the flag to be true.

```
585   function _deposit(
586     address _stakingToken,
587     address _from,
588     address _for,
589     uint256 _amount,
590     bool _isLock
591   ) internal {
592     PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
593     UserInfo storage user = userInfo[_stakingToken][_for];
594
595
596     updatePool(_stakingToken);
597     _harvestRewards(_stakingToken, _for);
598
599
600     user.amount = user.amount + _amount;
601     if (!_isLock) {
602         user.available = user.available + _amount;
603         IERC20(pool.stakingToken).safeTransferFrom(
604             address(_from),
605             address(this),
606             _amount
607         );
608     }
609     user.rewardDebt = (user.amount * pool.accPenpiePerShare) / 1e12;
610
611
612     if (_amount > 0) {
613         pool.totalStaked += _amount;
614         if (!_isLock)
615             emit Deposit(_for, _stakingToken, pool.receiptToken, _amount);
616         else emit DepositNotAvailable(_for, _stakingToken, _amount);
617     }
618   }
619
620
```

```
621  /// @notice internal function to deal with withdraw staking token
622  function _withdraw(
623      address _stakingToken,
624      address _account,
625      uint256 _amount,
626      bool _isLock
627  ) internal {
628      PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
629      UserInfo storage user = userInfo[_stakingToken][_account];
630
631
632      if (!_isLock && user.available < _amount)
633          revert WithdrawAmountExceedsStaked();
634      else if (user.amount < _amount && _isLock)
635          revert UnlockAmountExceedsLocked();
636
637
638      updatePool(_stakingToken);
639      _harvestPenpie(_stakingToken, _account);
640      _harvestBaseRewarder(_stakingToken, _account);
641
642
643      user.amount = user.amount - _amount;
644      if (!_isLock) {
645          user.available = user.available - _amount;
646          IERC20(tokenToPoolInfo[_stakingToken].stakingToken).safeTransfer(
647              address(msg.sender),
648              _amount
649          );
650      }
651      user.rewardDebt = (user.amount * pool.accPenpiePerShare) / 1e12;
652
653
654      pool.totalStaked -= _amount;
655
656
657      emit Withdraw(_account, _stakingToken, pool.receiptToken, _amount);
658  }
```

Listing 2.5: contracts/rewards/MasterPenpie.sol

Impact N/A

Suggestion Remove the deprecated feature logic.

2.3.2 Remove redundant checks in `ARBRewarder`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The modifier `_onlyMasterChef` will check whether the `masterChef` is `address(0)`. However, this check is redundant since the functions `addPool()` and `setPool()` have already checked that the `masterChef` can not be `address(0)`.

```
77    modifier _onlyMasterChef(address _stakingToken) {
78        address masterChef = tokenToPoolInfo[_stakingToken].masterChef;
79        if (masterChef != msg.sender && masterChef != address(0)) {
80            revert onlymasterChef();
81        }
82        _;
83    }
```

Listing 2.6: contracts/rewards/ARBRewarder.sol

Impact N/A.

Suggestion Remove the redundant check of `masterChef != address(0)`.

2.3.3 Avoid multiplying after division

Status Confirmed

Introduced by `Version 1`

Description In the function `getClaimable` in the contract `PenpieVesting`, the vested is calculated using the expression on Line 78: `granted * ((block.timestamp - startVestingTime) / intervals) / (vestingPeriodCount);`. However, the division is performed first in the expression, resulting in a loss of precision.

```
65    function getClaimable(address account) public view returns (uint256) {
66        if (address(rPNP) == address(0)) revert RPNPNotSet();
67        if (block.timestamp < startVestingTime) {
68            return 0;
69        }
70
71
72        uint256 claimed = claimedAmount[account];
73        uint256 granted = rPNP.balanceOf(account);
74
75
76        if (claimed >= granted) {
77            return 0;
78        }
79
80
81        uint256 vested = granted * ((block.timestamp - startVestingTime) / intervals) / (
              vestingPeriodCount);
82        if (vested > granted) {
83            return granted - claimed;
84        }
85
86
87        return vested - claimed;
88    }
```

Listing 2.7: contracts/rewards/PenpieVesting.sol

Impact N/A.

Suggestion Adjust the order of the expression to avoid the loss of precision.

Feedback from the Project It is intended, as we want to give rewards on the completion of the interval, let's say we have interval of 1 week then first time user claim reward is when it has passed one week for the vesting.

2.4 Note

2.4.1 Centralization risks

Introduced by Version 1

Description There are several important functions in the protocol, which are only callable by the owner. If the owner's private key is lost or compromised, it could lead to losses for the protocol and users.

Feedback from the Project We're using multisig as owner to govern our contracts.

2.4.2 `MannualCompound` must not hold any token

Introduced by Version 1

Description The function `compound()` in contract `ManualCompound` can be called by anyone with any reward token parameters (i.e., `_rewards`). Since the reward token will be transferred to the `msg.sender`, malicious users can call function `compound()` to steal all the tokens if there are tokens in the contract.

```
207   function compound(
208     address[] memory _lps,
209     address[][] memory _rewards,
210     bytes[] memory _kyBarExectCallData,
211     address[] memory baseTokens,
212     uint256[] memory compoundingMode,
213     pendleDexApproxParams memory _pdexparams,
214     bool isClaimPNP
215   ) external {
216
217
218     if(_rewards.length != _lps.length) revert InputDataLengthMissMatch();
219     if(baseTokens.length != _kyBarExectCallData.length) revert InputDataLengthMissMatch();
220
221
222     uint256 userTotalPendleRewardToSendBack;
223     uint256 userTotalPendleRewardToConvertMpendle;
224     uint256[] memory userPendleRewardsForCurrentMarket = new uint256[](_lps.length);
225
226
227     for(uint256 k; k < _lps.length;k++)
228     {
229         (,,,userPendleRewardsForCurrentMarket[k]) = masterPenpie.pendingTokens(_lps[k], msg.
                sender, PENDLE);
```

```
230     }
231
232     if(compoundingMode.length != userPendleRewardsForCurrentMarket.length) revert
            InputDataLengthMissMatch();
233
234
235     masterPenpie.multiclaimOnBehalf(
236             _lps,
237             _rewards,
238             msg.sender,
239             isClaimPNP
240     );
241
242     for (uint256 i; i < _lps.length;i++) {
243
244             for (uint j; j < _rewards[i].length;j++) {
245
246                 address _rewardTokenAddress = _rewards[i][j];
247                 uint256 receivedBalance = IERC20(_rewardTokenAddress).balanceOf(
248                     address(this)
249                 );
250
251
252                 if(receivedBalance == 0) continue;
253
254
255                 if (!compoundableRewards[_rewardTokenAddress]) {
256                         IERC20(_rewardTokenAddress).safeTransfer(
257                             msg.sender,
258                             receivedBalance
259                         );
260                         continue;
261                 }
```

Listing 2.8: contracts/rewards/ManualCompound.sol

Feedback from the Project Noted, we'll keep that in mind not to have any token in `ManualCompound`.

2.4.3 Token prices returned by `PenpieReader` can be inaccurate

Introduced by `Version 1`

Description The function `getTokenPrice()` returns spot prices when `tokenRouter.routerType != ChainlinkType`. If this function is not used off-chain, it might introduce price manipulation risk.

Feedback from the Project Price returned by `PenpieReader` is only used by front-end to display data on UI and not by any contracts.

2.4.4 `PendleRushV6` must not hold mPendle

Introduced by `Version 1`

Description The `PendleRushV6` contract provides a `convert()` function that allows users to convert Pendle tokens to mPendle.

However, this function uses the mPendle balance after the conversion instead of the actual converted amount as the final amount sent back to the user. If the contract holds any mPendle token, a malicious user can invoke this function with the `_amount` to be zero to drain the mPendle balance of this contract.

```
217   function convert(
218       uint256 _amount,
219       pendleDexApproxParams memory _pdexparams,
220       uint256 _convertMode
221   ) external whenNotPaused nonReentrant {
222       if (!this.validConvertor(msg.sender)) revert InvalidConvertor();
223
224
225       if (mPendleMarket == address(0)) revert mPendleMarketNotSet();
226
227
228       (uint256 rewardToSend, uint256 bonusARBReward) = this.quoteConvert(_amount, msg.sender);
229
230
231       _convert(msg.sender, _amount);
232       uint256 treasuryFeeAmount = (IERC20(mPENDLE).balanceOf(address(this)) - _amount) *
              treasuryFee / DENOMINATOR;
233       uint256 mPendleToTransfer = _mPendleTransferAndLock(msg.sender, IERC20(mPENDLE).balanceOf(
              address(this)) - treasuryFeeAmount);
234
235
236       if (mPendleToTransfer > 0) {
237           if (_convertMode == CONVERT_TO_MPENDLE) {
238               IERC20(mPENDLE).safeTransfer(msg.sender, mPendleToTransfer);
239           } else if (_convertMode == LIQUIDATE_TO_PENDLE_FINANCE) {
240               _ZapInmPendleToMarket(mPendleToTransfer, _pdexparams);
241           } else {
242               revert InvalidConvertMode();
243           }
244       }
245
246
247       if (treasuryFeeAmount > 0){
248           IERC20(mPENDLE).safeTransfer(owner(), treasuryFeeAmount);
249       }
250
251
252       UserInfo storage userInfo = userInfos[msg.sender];
253       userInfo.converted += _amount;
254       userInfo.rewardClaimed += (rewardToSend - bonusARBReward);
255       userInfo.bonusRewardClaimed += bonusARBReward;
256       totalAccumulated += _amount;
257       userInfo.convertedTimes += 1;
258
259
```

```
260     ARB.safeTransfer(msg.sender, rewardToSend);
261
262
263     emit ARBRewarded(msg.sender, rewardToSend);
264 }
```

Listing 2.9: contracts/pendle/PendleRushV6.sol

Feedback from the Project `mPendle` to all the Pendle Rushes is minted when the conversion is done but we'll keep in mind not to have `mPendle` in any Pendle Rush.

2.4.5 Users can donate Pendle to `PendleStaking` via function `convertPendle`

Introduced by `Version 1`

Description The `PendleStaking` contract has a `convertPendle()` function, which can be invoked by anyone, for the operator of the `mPendleConverter` to lock the Pendle tokens in vePendle. Donating Pendle tokens to this contract and locking the tokens on behalf of this contract will not bring any financial benefits to the user.

```
78   function convertPendle(
79     uint256 _amount,
80     uint256[] calldata chainId
81 ) public payable override whenNotPaused returns (uint256) {
82     uint256 preVePendleAmount = accumulatedVePendle();
83     if (_amount == 0) revert ZeroNotAllowed();
84
85
86     IERC20(PENDLE).safeTransferFrom(msg.sender, address(this), _amount);
87     IERC20(PENDLE).safeApprove(address(vePendle), _amount);
88
89
90     uint128 unlockTime = _getIncreaseLockTime();
91     IPVotingEscrowMainchain(vePendle).increaseLockPositionAndBroadcast{value:msg.value}(uint128(
           _amount), unlockTime, chainId);
92
93
94     uint256 mintedVePendleAmount = accumulatedVePendle() -
95         preVePendleAmount;
96     emit PendleLocked(_amount, lockPeriod, mintedVePendleAmount);
97
98
99     return mintedVePendleAmount;
100 }
```

Listing 2.10: contracts/pendle/PendleStaking.sol

```
39   function lockAllPendle(
40     uint256[] calldata chainId
41 ) external payable onlyOperator {
42
43     uint256 allPendle = IERC20(pendle).balanceOf(address(this));
44
```

```
45
46      IERC20(pendle).safeApprove(pendleStaking, allPendle);
47
48
49      uint256 mintedVePendleAmount = IPendleStaking(pendleStaking)
50          .convertPendle{ value: msg.value }(allPendle, chainId);
51
52
53      emit PendleConverted(allPendle, mintedVePendleAmount);
54  }
```

Listing 2.11: contracts/pendle/mPendleConvertor.sol

Feedback from the Project Yes, the donator does not benefit from donating, we're aware of that.

2.4.6 `PendleStaking`'s Pendle locked in `vePendle` can be locked permanently by anyone

Introduced by `Version 1`

Description The `PendleStaking` contract locks Pendle tokens to the `vePendle` to get voting power. The lock time can be extended by calling `increaseLockPosition()` in the `vePendle`. The protocol specifies that the locked Pendle tokens will be locked eternally, so the contract provides an `increaseLockTime()` function to allow anyone to increase the lock time on behalf of this contract.

Feedback from the Project Yes, we're also aware of this, anyone can lock Pendle in our `PendleStaking`.

2.4.7 Precision loss in function `updatePool` is negligible

Introduced by `Version 1`

Description The `updatePool()` function in the `MasterPenpie` contract allows anyone to update the rewards of a specific pool. A malicious user can frequently invoke this function, resulting in the users receiving less or even no rewards. Specifically, the `penpieReward` calculation suffers precision losses if the pool is updated frequently enough. However, considering the current configuration of the protocol, the loss is too negligible that it can be ignored.

```
428   function updatePool(address _stakingToken) public whenNotPaused {
429     PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
430     if (
431         block.timestamp <= pool.lastRewardTimestamp || totalAllocPoint == 0
432     ) {
433         return;
434     }
435     uint256 lpSupply = pool.totalStaked;
436     if (lpSupply == 0) {
437         pool.lastRewardTimestamp = block.timestamp;
438         return;
439     }
```

```
440     uint256 multiplier = block.timestamp - pool.lastRewardTimestamp;
441     uint256 penpieReward = (multiplier * penpiePerSec * pool.allocPoint) /
442         totalAllocPoint;
443
444
445     pool.accPenpiePerShare =
446         pool.accPenpiePerShare +
447         ((penpieReward * 1e12) / lpSupply);
448     pool.lastRewardTimestamp = block.timestamp;
449
450
451     emit UpdatePool(
452         _stakingToken,
453         pool.lastRewardTimestamp,
454         lpSupply,
455         pool.accPenpiePerShare
456     );
457 }
```

Listing 2.12: contracts/rewards/MasterPenpie.sol

Feedback from the Project The current configuration can't make Penpie reward to be zero even when the update gap is 1 second.

2.4.8 `queuedRewards` will be distributed to the first depositor

Introduced by `Version 1`

Description In function `_provisionReward()`, the reward will be accumulated to `queuedRewards` if the supply of `receiptToken` is zero, and all the `queuedRewards` will be harvested to increase the `rewardPerTokenStored` once the supply of `receiptToken` becomes non-zero. As a result, the first staked user will get all the queued rewards.

```
251   function donateRewards(uint256 _amountReward, address _rewardToken) external {
252     if (!isRewardToken[_rewardToken])
253         revert MustBeRewardToken();
254
255
256     _provisionReward(_amountReward, _rewardToken);
257 }
```

Listing 2.13: contracts/rewards/BaseRewardPoolV2.sol

```
286   function _provisionReward(uint256 _amountReward, address _rewardToken) internal {
287     IERC20(_rewardToken).safeTransferFrom(
288         msg.sender,
289         address(this),
290         _amountReward
291     );
292     Reward storage rewardInfo = rewards[_rewardToken];
293
294
295     uint256 totalStake = totalStaked();
```

```
296     if (totalStake == 0) {
297         rewardInfo.queuedRewards += _amountReward;
298     } else {
299         if (rewardInfo.queuedRewards > 0) {
300             _amountReward += rewardInfo.queuedRewards;
301             rewardInfo.queuedRewards = 0;
302         }
303         rewardInfo.rewardPerTokenStored =
304             rewardInfo.rewardPerTokenStored +
305             (_amountReward * 10**receiptTokenDecimals) /
306             totalStake;
307     }
308     emit RewardAdded(_amountReward, _rewardToken);
309 }
```

Listing 2.14: contracts/rewards/BaseRewardPoolV2.sol

Feedback from the Project If the receipt token's total supply is zero, that means there's no TVL in the pool. Since we harvest rewards from Pendle Finance for our LP position, if the TVL in the pool is zero, we won't be receiving any rewards upon harvest, and thereby no rewards will be sent to the rewarder in case the receipt token's total supply is zero. In the case of the Penpie pools (vlPNP, mPendle, mPendleSV), they might receive rewards even if the TVL in them is 0, but it's intended behavior that if the pool's rewarder received rewards when the TVL was zero, then the first depositor gets the accumulated rewards.

2.4.9 `penpieReward` should not be distributed to empty pools

Introduced by `Version 1`

Description In the contract `MasterPenpie`, when the `pool.totalStaked == 0`, the `pool.lastRewardTimesta` will be updated to `block.timestamp` and return. As a result, this will cause part of `penpieRewards` to be unclaimed and locked in `MasterPenpie` when the `pool.allocPoint` is not zero. The `penpieReward` that is allocated to the pool will not be added to `pool.accPenpiePerShare`.

Feedback from the Project These rewards, given via MasterPempie, are the PNP tokens. PNP tokens are what Penpie distributes; all other rewards are harvested from Pendle Finance and then sent to the rewarders. If a pool has zero total staked, Penpie can avoid giving PNP tokens to that pool since there are no users who have staked in that pool. It is better not to give any PNP tokens to that pool!

2.4.10 The protocol will avoid potential lock or draining of rewards for Pendle market

Introduced by `Version 1`

Description In the `PendleStakingBaseUpg` contract, the rewards can be harvested by the `_harvestBatchMarketRewards()` function. The function accepts the markets to be harvested and gets the reward tokens of each market. By comparing the reward token balance changes before and after invoking the market's `redeemRewards()` function, the contract decides how many reward tokens are received and records the rewards to each pool.

```
718   function _harvestBatchMarketRewards(
719       address[] memory _markets,
720       address _caller,
721       uint256 _minEthToRecieve
722   ) internal {
723       uint256 harvestCallerTotalPendleReward;
724       uint256 pendleBefore = IERC20(PENDLE).balanceOf(address(this));
725
726
727       for (uint256 i = 0; i < _markets.length; i++) {
728           if (!pools[_markets[i]].isActive) revert OnlyActivePool();
729           Pool storage poolInfo = pools[_markets[i]];
730
731
732           poolInfo.lastHarvestTime = block.timestamp;
733
734
735           address[] memory bonusTokens = IPendleMarket(_markets[i]).getRewardTokens();
736           uint256[] memory amountsBefore = new uint256[](bonusTokens.length);
737
738
739           for (uint256 j; j < bonusTokens.length; j++) {
740               if (bonusTokens[j] == NATIVE) bonusTokens[j] = address(WETH);
741
742
743               amountsBefore[j] = IERC20(bonusTokens[j]).balanceOf(address(this));
744           }
745
746
747           IPendleMarket(_markets[i]).redeemRewards(address(this));
748
749
750           for (uint256 j; j < bonusTokens.length; j++) {
751               uint256 amountAfter = IERC20(bonusTokens[j]).balanceOf(address(this));
752
753
754               uint256 originalBonusBalance = amountAfter - amountsBefore[j];
755               uint256 leftBonusBalance = originalBonusBalance;
756               uint256 currentMarketHarvestPendleReward;
757
758
759               if (originalBonusBalance == 0) continue;
760
761
762               if (bonusTokens[j] == PENDLE) {
763                   currentMarketHarvestPendleReward =
764                       (originalBonusBalance * harvestCallerPendleFee) /
765                       DENOMINATOR;
766                   leftBonusBalance = originalBonusBalance - currentMarketHarvestPendleReward;
767               }
768               harvestCallerTotalPendleReward += currentMarketHarvestPendleReward;
769
```

```
770
771             _sendRewards(
772                 _markets[i],
773                 bonusTokens[j],
774                 poolInfo.rewarder,
775                 originalBonusBalance,
776                 leftBonusBalance
777             );
778         }
779     }
```

Listing 2.15: contracts/pendle/PendleStakingBaseUpg.sol

However, the `_markets[i]` is a `PendleMarket` contract that allows anyone to collect the rewards on behalf of another identity. Therefore, two potential paths exist to exploit this mechanism, causing different harms to this protocol.

- Lock of rewards. By first calling the `redeemRewards` of the corresponding market, the rewards of `PendleStakingBaseUpg` are cleared. As a result, in the following invocation to the `_harvestBatchMarketRewards()`, the reward token balance change will be negligible or even zero and the contract is unaware that the rewards are already distributed to itself. Thus, the rewards are locked in this contract rather than distributed to correct users.
- Draining of rewards. In the past versions, Penpie allowed a public Pendle market to be registered, all legal Pendle markets can be registered in this contract and the rewards can be harvested. In the audited version, this feature is temporarily restricted to `onlyOwner`. However, if the market can be publicly registered again, an attacker can drain the rewards of all registered markets. The attack steps are as follows:
  - Create a Pendle market with the underlying SY token to be controlled by the attacker. Register the market in the `PendleStakingBaseUpg` contract.
  - Invoke the `harvestMarketReward()` function to reach the `_harvestBatchMarketRewards()` logic.
  - In the `redeemRewards` function that will forward the execution flow to the malicious SY token, the attacker can redeem all rewards of the `PendleStakingBaseUpg` contract before returning to the `_harvestBatchMarketRewards()`.
  - All reward tokens of other markets are distributed to the contract, and the contract regards those tokens as the rewards of the malicious market. As a result, all rewards are sent to the malicious market rewarder(whose beneficiary will be only the attacker), leading to the reward being drained.

The protocol is aware of such risks and takes action to prevent them from happenning.

- Disable public pendle market register.
- Actively monitor the contract balance so that once the rewards are maliciously claimed the protocol will use a privileged function to manually distribute the rewards.

Feedback from the Project

- We will not allow to register pools by pendle market register helper, it will be done by the owner itself.
- If any reward is locked in the contract, there is an admin function: updateMarketRewards which to distribute stuck reward in pendleStaking. We can use off-chain monitor if there

is any PENDLE balance increased in Pendlestaking (meaning reward gets stuck).


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS