BLOCKSEC

Security Audit
Report for Penpie Contracts

Date: January 15, 2025 **Version:** 2.1

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**

1.1 About Target Contracts . . . 1
1.2 Disclaimer . . . 2
1.3 Procedure of Auditing . . . 3
1.3.1 Software Security . . . 3
1.3.2 DeFi Security . . . 3
1.3.3 NFT Security . . . 4
1.3.4 Additional Recommendation . . . 4
1.4 Security Model . . . 4

Chapter 2 Findings **6**

2.1 Software Security . . . 7
2.1.1 Potential DoS due to arbitrarily added markets in function `addPenpieBribePool` 7
2.1.2 The ascending order of the `_boostTokentier` can be broken by adding a single multiplier . . . 8
2.2 DeFi Security . . . 9
2.2.1 Lack of logic on handling specific reward tokens in function `compound` . . . 9
2.2.2 Lack of harvesting pool when the `allocPoint` is changed . . . 11
2.2.3 Potential incorrect state update due to insufficient check in `_addPool` function . . . 12
2.2.4 The `receiptToStakeToken` mapping of the original pool can be accidentally updated . . . 13
2.2.5 Potential inconsistent decimals in function `maxPRTByLeftMGP()` . . . 14
2.2.6 Lack of check on auction status in function `config()` . . . 15
2.2.7 Lack of checks when withdrawing bidTokens . . . 16
2.3 Additional Recommendation . . . 16
2.3.1 Remove unused logic in `_deposit` and `_withdraw` in `MasterPenpie` . . . 16
2.3.2 Remove redundant checks in `ARBRewarder` . . . 18
2.3.3 Refactor code to optimize gas consumption . . . 18
2.3.4 Fix typos . . . 18
2.3.5 Fix typos . . . 19
2.3.6 Avoid precision losses in function `getClaimable()` . . . 20
2.4 Note . . . 21
2.4.1 Potential centralization risk . . . 21
2.4.2 `MannualCompound` must not hold any token . . . 21
2.4.3 Token prices returned by `PenpieReader` can be inaccurate . . . 22
2.4.4 `PendleRushV6` must not hold mPendle . . . 22
2.4.5 Users can donate Pendle to `PendleStaking` via function `convertPendle` . . . 23
2.4.6 `PendleStaking`'s Pendle locked in `vePendle` can be locked permanently by anyone . . . 24

2.4.7 Precision loss in function `updatePool` is negligible 25
2.4.8 `queuedRewards` will be distributed to the first depositor 26
2.4.9 `penpieReward` should not be distributed to empty pools 27
2.4.10 The protocol will avoid potential lock or draining of rewards for Pendle market 27
2.4.11 Function `_convertPendleTomPendle` may lead to users' assets loss 29
2.4.12 `DutchAuction` owner fully controls the project tokens 29

Report Manifest

Item	Description
Client	Magpiexyz
Target	Penpie Contracts

Version History

Version	Date	Description
1.0	October 13, 2024	First release
1.1	November 6, 2024	Second release
2.0	December 18, 2024	Third release
2.1	January 15, 2025	Fourth release

Signature

Chapter 1 Introduction

1.1 About Target Contracts

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

This audit focuses on the Penpie Contracts contract for Magpiexyz [1]. 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, for the version 1, 2 and 3, only the following contracts 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
- contracts/bribeMarket/PenpieBribeRewardDistributor.sol
- contracts/rewards/ManualCompound.sol

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

- 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

Furthermore, for the version 4 and 5, only the following contracts in the repository are included in the scope of this audit. Other files are not within the scope of this audit.

- contracts/rewards/PRTAirdrop.sol
- contracts/DutchAuction.sol
- contracts/PRT.sol
- contracts/VLMGPExchange.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		Commit SHA
Penpie Contracts	Version 1	f5a6682c301fad7358fe7ce02cfef3e710f66a6e
	Version 2	c363aade34d0ef9e83a76a8bd83eb5f3cd71577e
	Version 3	ad901c9e92f15d69cb6d333d1f40347723224f31
	Version 4	6f26e064de4cbe072be368cd069ab036a603d35e
	Version 5	5dcfd27859dd513a2a73c4725b8fb51a011d8acc
	Version 6	24e4deeb10296c859f617fe70ec5fdd489674d0c

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

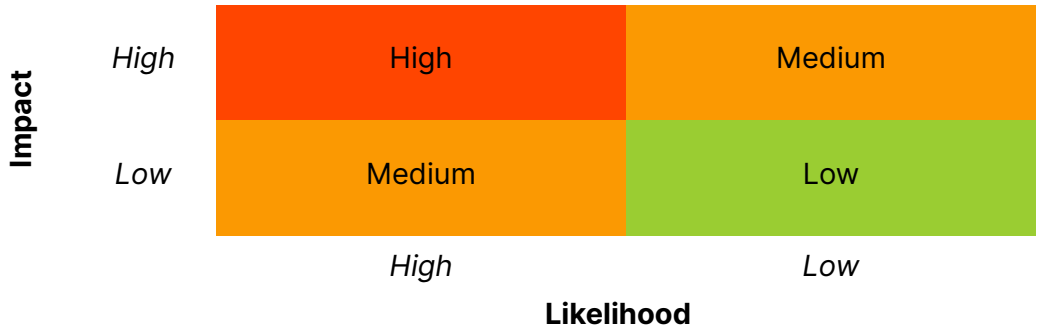
Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

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 [2] and Common Weakness Enumeration [3]. 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



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.

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

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

- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **nine** potential security issues. Besides, we have **six** recommendations and **twelve** notes.

- Medium Risk: 1
- Low Risk: 8
- Recommendation: 6
- Note: 12

ID	Severity	Description	Category	Status
1	Low	Potential DoS due to arbitrarily added markets in function `addPenpieBribePool`	Software Security	Fixed
2	Low	The ascending order of the `_boostTokentier` can be broken by adding a single multiplier	Software Security	Confirmed
3	Medium	Lack of logic on handling specific reward tokens in function `compound`	DeFi Security	Fixed
4	Low	Lack of harvesting pool when the `allocPoint` is changed	DeFi Security	Fixed
5	Low	Potential incorrect state update due to insufficient check in `_addPool` function	DeFi Security	Confirmed
6	Low	The `receiptToStakeToken` mapping of the original pool can be accidentally updated	DeFi Security	Fixed
7	Low	Potential inconsistent decimals in function `maxPRTByLeftMGP()`	DeFi Security	Confirmed
8	Low	Lack of check on auction status in function `config()`	DeFi Security	Fixed
9	Low	Lack of checks when withdrawing bidTokens	DeFi Security	Fixed
10	-	Remove unused logic in `_deposit` and `_withdraw` in `MasterPenpie`	Recommendation	Fixed
11	-	Remove redundant checks in `ARBRewarder`	Recommendation	Fixed
12	-	Refactor code to optimize gas consumption	Recommendation	Fixed
13	-	Fix typos	Recommendation	Fixed
14	-	Fix typos	Recommendation	Fixed
15	-	Avoid precision losses in function `getClaimable()`	Recommendation	Fixed
16	-	Potential centralization risk	Note	-
17	-	`MannualCompound` must not hold any token	Note	-

18	-	Token prices returned by `PenpieReader` can be inaccurate	Note	-
19	-	`PendleRushV6` must not hold mPendle	Note	-
20	-	Users can donate Pendle to `PendleStaking` via function `convertPendle`	Note	-
21	-	`PendleStaking`'s Pendle locked in `vePendle` can be locked permanently by anyone	Note	-
22	-	Precision loss in function `updatePool` is negligible	Note	-
23	-	`queuedRewards` will be distributed to the first depositor	Note	-
24	-	`penpieReward` should not be distributed to empty pools	Note	-
25	-	The protocol will avoid potential lock or draining of rewards for Pendle market	Note	-
26	-	Function `_convertPendleTomPendle` may lead to users' assets loss	Note	-
27	-	`DutchAuction` owner fully controls the project tokens	Note	-

The details are provided in the following sections.

2.1 Software Security

2.1.1 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 2.1: contracts/pendle/PendleMarketRegisterHelper.sol

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

Listing 2.2: 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.

2.1.2 The ascending order of the `_boostTokentier` can be broken by adding a single multiplier

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In `PendleRushV6`, the `boostTokenRewardMultiplier` should be in ascending order. However, when setting the multipliers, the `setBoostTokenMultiplier` function only checks the order of the current configured `_boostTokentier`. As a result, the assumption on the ascending order of the `boostTokenRewardMultiplier` may be broken by misconfiguration.

```
442   function setBoostTokenMultiplier(
443       uint256[] calldata _boostTokenmultiplier,
444       uint256[] calldata _boostTokentier
445   ) external onlyOwner {
446       if (_boostTokenmultiplier.length == 0 || _boostTokentier.length == 0 || (
              _boostTokenmultiplier.length != _boostTokentier.length))
447           revert BoostTokenLengthMismatch();
448
449       for (uint8 i; i < _boostTokenmultiplier.length; ++i) {
450           if (_boostTokenmultiplier[i] == 0) revert InvalidBoostTokenAmount();
451           if (i > 0) {
452               require(_boostTokentier[i] > _boostTokentier[i-1], "Boost Token reward tier values
                      must be in increasing order.");
```

```
453         }
454         boostTokenRewardMultiplier.push(_boostTokenmultiplier[i]);
455         boostTokenRewardTier.push(_boostTokentier[i]);
456         boostTokenTierLength += 1;
457     }
458 }
```

Listing 2.3: contracts/pendle/PendleRushV6.sol

Impact Misconfigurations might be applied to the protocol.

Suggestion Add sanity checks.

Feedback from the project We are aware of it, but to prevent extra loops we are using this and moreover we configure the pendle rush multiplier's in a single go, if we need to add different multipliers then we will reset the multipliers and then again add the multipliers.

2.2 DeFi Security

2.2.1 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     if(_rewards.length != _lps.length) revert InputDataLengthMissMatch();
218     if(baseTokens.length != _kyBarExectCallData.length) revert InputDataLengthMissMatch();
219
220     uint256 userTotalPendleRewardToSendBack;
221     uint256 userTotalPendleRewardToConvertMpendle;
222     uint256[] memory userPendleRewardsForCurrentMarket = new uint256[](_lps.length);
223
224     for(uint256 k; k < _lps.length;k++)
225     {
226         (,,,userPendleRewardsForCurrentMarket[k]) = masterPenpie.pendingTokens(_lps[k], msg.
                sender, PENDLE);
227     }
```

```
228
229     if(compoundingMode.length != userPendleRewardsForCurrentMarket.length) revert
            InputDataLengthMissMatch();
230
231     masterPenpie.multiclaimOnBehalf(
232             _lps,
233             _rewards,
234             msg.sender,
235             isClaimPNP
236     );
237
238     for (uint256 i; i < _lps.length;i++) {
239
240             for (uint j; j < _rewards[i].length;j++) {
241
242                 address _rewardTokenAddress = _rewards[i][j];
243                 uint256 receivedBalance = IERC20(_rewardTokenAddress).balanceOf(
244                     address(this)
245                 );
246
247                 if(receivedBalance == 0) continue;
248
249                 if (!compoundableRewards[_rewardTokenAddress]) {
250                         IERC20(_rewardTokenAddress).safeTransfer(
251                             msg.sender,
252                             receivedBalance
253                         );
254                         continue;
255                 }
256
257
258                 if (_rewardTokenAddress == PENDLE) {
259                     if(compoundingMode[i] == LIQUIDATE_TO_PENDLE_FINANCE)
260                     {
261                         IERC20(PENDLE).safeApprove(address(pendleRouter),
                                userPendleRewardsForCurrentMarket[i]);
262                         _ZapInToPendleMarket(userPendleRewardsForCurrentMarket[i], _lps[i],
                                baseTokens[i], _kyBarExectCallData[i], _pdexparams );
263                     }
264                     else if( compoundingMode[i] == CONVERT_TO_MPENDLE )
265                     {
266                         userTotalPendleRewardToConvertMpendle += userPendleRewardsForCurrentMarket
                                [i];
267                     }
268                     else
269                     {
270                         userTotalPendleRewardToSendBack += userPendleRewardsForCurrentMarket[i];
271                     }
272                 }
273                 else if (_rewardTokenAddress == PENPIE) {
274                         _lockPenpie(receivedBalance);
275                 }
276             }
```

```
277     }
278
279     if(userTotalPendleRewardToConvertMpendle != 0) _convertToMPendle(
            userTotalPendleRewardToConvertMpendle);
280     if(userTotalPendleRewardToSendBack != 0 ) IERC20(PENDLE).safeTransfer( msg.sender,
            userTotalPendleRewardToSendBack );
281
282     emit Compounded(msg.sender, _lps.length, _rewards.length);
283 }
```

Listing 2.4: contracts/rewards/ManualCompound.sol

Impact Potential lock of rewards.

Suggestion Add the logic to handle other reward tokens.

2.2.2 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 2.5: contracts/rewards/MasterPenpie.sol

Impact The harvested Penpie reward can be inaccurate.

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

2.2.3 Potential incorrect state update due to insufficient check in `_addPool` function

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the `MasterPenpie` contract, the validations in the `_addPool` function are insufficient. Specifically, the function checks `tokenToPoolInfo[_stakingToken].isActive` to verify that a pool does not exist. However, the `isActive` field can be modified via the `set` function. If an inactive pool is mistakenly added again as a new pool, the state of the origin pool can be overridden, resulting in unexpected results.

```
836    function _addPool(
837        uint256 _allocPoint,
838        address _stakingToken,
839        address _receiptToken,
840        address _rewarder
841    ) internal {
842        if (
843            !Address.isContract(address(_stakingToken)) ||
844            !Address.isContract(address(_receiptToken))
845        ) revert InvalidStakingToken();
846
847        if (
848            !Address.isContract(address(_rewarder)) &&
849            address(_rewarder) != address(0)
850        ) revert MustBeContractOrZero();
851
852        if (tokenToPoolInfo[_stakingToken].isActive) revert PoolExisted();
853
854        if (_allocPoint != 0){
855            massUpdatePools();
856        }
857
858        uint256 lastRewardTimestamp = block.timestamp > startTimestamp
859            ? block.timestamp
860            : startTimestamp;
861        totalAllocPoint = totalAllocPoint + _allocPoint;
862        registeredToken.push(_stakingToken);
```

```
863         // it's receipt token as the registered token
864         tokenToPoolInfo[_stakingToken] = PoolInfo({
865             receiptToken: _receiptToken,
866             stakingToken: _stakingToken,
867             allocPoint: _allocPoint,
868             lastRewardTimestamp: lastRewardTimestamp,
869             accPenpiePerShare: 0,
870             totalStaked: 0,
871             rewarder: _rewarder,
872             isActive: true
873         });
874 
875         receiptToStakeToken[_receiptToken] = _stakingToken;
876 
877         emit Add(
878             _allocPoint,
879             _stakingToken,
880             _receiptToken,
881             IBaseRewardPool(_rewarder)
882         );
883     }
```

Listing 2.6: contracts/rewards/MasterPenpie.sol

Impact Incorrect pool additions can lead to an incorrect contract state.

Suggestion Add a check to ensure that the `_stakingToken` is not added to the `registeredToken`. Meanwhile, ensure that the `receiptToStakeToken[_receiptToken]` does not exist before adding a new pool.

Feedback from the project The `set` function will be removed in the future.

2.2.4 The `receiptToStakeToken` mapping of the original pool can be accidentally updated

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description When a `receiptToken` of an existing pool is added again, the `receiptToStakeToken` mapping of the original pool can be accidentally updated, causing an incorrect `stakingToken` to be indexed.

```
836     function _addPool(
837         uint256 _allocPoint,
838         address _stakingToken,
839         address _receiptToken,
840         address _rewarder
841     ) internal {
842         if (
843             !Address.isContract(address(_stakingToken)) ||
844             !Address.isContract(address(_receiptToken))
845         ) revert InvalidStakingToken();
```

```
846
847         if (
848             !Address.isContract(address(_rewarder)) &&
849             address(_rewarder) != address(0)
850         ) revert MustBeContractOrZero();
851
852         if (tokenToPoolInfo[_stakingToken].isActive) revert PoolExisted();
853
854         if (_allocPoint != 0){
855             massUpdatePools();
856         }
857
858         uint256 lastRewardTimestamp = block.timestamp > startTimestamp
859             ? block.timestamp
860             : startTimestamp;
861         totalAllocPoint = totalAllocPoint + _allocPoint;
862         registeredToken.push(_stakingToken);
863         // it's receipt token as the registered token
864         tokenToPoolInfo[_stakingToken] = PoolInfo({
865             receiptToken: _receiptToken,
866             stakingToken: _stakingToken,
867             allocPoint: _allocPoint,
868             lastRewardTimestamp: lastRewardTimestamp,
869             accPenpiePerShare: 0,
870             totalStaked: 0,
871             rewarder: _rewarder,
872             isActive: true
873         });
874
875         receiptToStakeToken[_receiptToken] = _stakingToken;
876
877         emit Add(
878             _allocPoint,
879             _stakingToken,
880             _receiptToken,
881             IBaseRewardPool(_rewarder)
882         );
883     }
```

Listing 2.7: contracts/rewards/MasterPenpie.sol

Impact Incorrect pool additions can lead to an incorrect contract state.

Suggestion Ensure that the `receiptToStakeToken[_receiptToken]` does not exist before adding a new pool.

2.2.5 Potential inconsistent decimals in function `maxPRTByLeftMGP()`

Severity Low

Status Confirmed

Introduced by `Version 4`

Description In the contract `VLMGPExchange`, the function `maxPRTByLeftMGP()` uses the decimals of the token `vlMGP` to calculate the maximum amount of the token `PRT` that can be exchanged. This is incorrect since the calculation of the exchange is between the token `PRT` and the token `MGP`. Therefore, the calculation should use the decimals of the token `MGP` instead.

```
106   function maxPRTByLeftMGP() public view returns(uint256) {
107       uint256 maxPrt = ((IERC20(MGP).balanceOf(address(this)) * (10 ** IMintableERC20(PRT).
              decimals())) / (10 ** IMintableERC20(address(vlMGP)).decimals())) /
              currentExchangeRate;
108       return maxPrt;
109   }
```

Listing 2.8: contracts/VLMGPExchange.sol

Impact The calculation might be incorrect.

Suggestion Use the decimals of the token `MGP` instead.

Feedback from the project `Vlmgp` is a locked version of `MGP`. Decimals of `MGP` and `VLMGP` must always be the same.

2.2.6 Lack of check on auction status in function `config()`

Severity Low

Status Fixed in `Version 5`

Introduced by `Version 4`

Description In the contract `DutchAuction`, the function `config()` does not check whether the auction has started or not. However, if the function `config()` is invoked when the auction has started, the auction's `startingPrice`, `minPrice`, `priceInterval`, `priceDecrementPrcnt` and `auctionStartTime` will be changed, which will finally affect the process of the auction.

```
207   function config(
208       uint256 _startingPrice,
209       uint256 _minPrice,
210       uint256 _priceInterval,
211       uint256 _priceDecrementPrcnt,
212       uint256 _AuctionStartTime
213   ) external onlyOwner {
214       startingPrice = _startingPrice;
215       minPrice = _minPrice;
216       priceInterval = _priceInterval;
217       priceDecrementPrcnt = _priceDecrementPrcnt;
218       auctionStartTime = _AuctionStartTime;
219
220
221       emit ConfiguredNewData(startingPrice, minPrice, priceInterval, _priceDecrementPrcnt,
              auctionStartTime);
222   }
```

Listing 2.9: contracts/DutchAuction.sol

Impact The process of the auction will be affected.

Suggestion Add a check on `block.timestamp` to make sure the auction has not started yet.

2.2.7 Lack of checks when withdrawing bidTokens

Severity Low

Status Fixed in `Version 5`

Introduced by `Version 4`

Description In the contract `DutchAuction`, the function `withdrawBidTokens()` allows the owner to withdraw the `bidTokens` without checking whether the auction has ended or not. It would be more appropriate to restrict this withdrawal to after the auction's end.

```
239   function withdrawBidTokens() external onlyOwner nonReentrant {
240       uint256 balancebidToken = IERC20(bidToken).balanceOf(address(this));
241       IERC20(bidToken).transfer(msg.sender, balancebidToken);
242   }
```

Listing 2.10: contracts/DutchAuction.sol

Impact This could lead to failures of user claims.

Suggestion Revise the logic accordingly.

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       updatePool(_stakingToken);
596       _harvestRewards(_stakingToken, _for);
597
598       user.amount = user.amount + _amount;
599       if (!_isLock) {
600           user.available = user.available + _amount;
601           IERC20(pool.stakingToken).safeTransferFrom(
602               address(_from),
603               address(this),
```

```
604             _amount
605         );
606     }
607     user.rewardDebt = (user.amount * pool.accPenpiePerShare) / 1e12;
608
609     if (_amount > 0) {
610         pool.totalStaked += _amount;
611         if (!_isLock)
612             emit Deposit(_for, _stakingToken, pool.receiptToken, _amount);
613         else emit DepositNotAvailable(_for, _stakingToken, _amount);
614     }
615 }
616
617 /// @notice internal function to deal with withdraw staking token
618 function _withdraw(
619     address _stakingToken,
620     address _account,
621     uint256 _amount,
622     bool _isLock
623 ) internal {
624     PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
625     UserInfo storage user = userInfo[_stakingToken][_account];
626
627     if (!_isLock && user.available < _amount)
628         revert WithdrawAmountExceedsStaked();
629     else if (user.amount < _amount && _isLock)
630         revert UnlockAmountExceedsLocked();
631
632     updatePool(_stakingToken);
633     _harvestPenpie(_stakingToken, _account);
634     _harvestBaseRewarder(_stakingToken, _account);
635
636     user.amount = user.amount - _amount;
637     if (!_isLock) {
638         user.available = user.available - _amount;
639         IERC20(tokenToPoolInfo[_stakingToken].stakingToken).safeTransfer(
640             address(msg.sender),
641             _amount
642         );
643     }
644     user.rewardDebt = (user.amount * pool.accPenpiePerShare) / 1e12;
645
646     pool.totalStaked -= _amount;
647
648     emit Withdraw(_account, _stakingToken, pool.receiptToken, _amount);
649 }
```

Listing 2.11: contracts/rewards/MasterPenpie.sol

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 2.12: contracts/rewards/ARBRewarder.sol

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

2.3.3 Refactor code to optimize gas consumption

Status Fixed in `Version 2`

Introduced by `Version 1`

Description There are two invocations of the function `getUserTotalLocked()` in the function `startUnlock()` and can be optimized to only call it once.

```
317    function startUnlock(
318        uint256 _amountToCoolDown
319    ) external override whenNotPaused nonReentrant {
320        if (_amountToCoolDown > getUserTotalLocked(msg.sender))
321            revert NotEnoughLockedPenpie();
322
323        uint256 totalLockAfterStartUnlock = getUserTotalLocked(msg.sender) -
324            _amountToCoolDown;
```

Listing 2.13: contracts/VLPenpie.sol

Suggestion Optimize the code to reduce gas consumption.

2.3.4 Fix typos

Status Fixed in `Version 2`

Introduced by `Version 1`

Description There are some typos in the project. For instance, the comment “stacking” in the `mPendleSV` contract should be corrected to “staking”, and “toal” in `VLPenpie` should be changed to “total”.

```
20  /// @notice mPendle is designed for Locking mPendle tokens and earn higher rewards than
        regular mPendle stacking
```

Listing 2.14: contracts/mPendleSV.sol

```
512 function _lock(
513     address spender,
514     address _for,
515     uint256 _amount
516 ) internal {
517     penpie.safeTransferFrom(spender, address(this), _amount);
518     IMasterPenpie(masterPenpie).depositVlPenpieFor(_amount, _for);
519     totalAmount += _amount; // trigers update pool share, so happens after toal amount increase
520 }
```

Listing 2.15: contracts/VLPenpie.sol

Suggestion Fix these typos ensure the code more clean.

2.3.5 Fix typos

Status Fixed in `Version 5`

Introduced by `Version 4`

Description In the contract `DutchAction`, the function `getClaimable()` has a typo, which should be `userAllocated` rather than `userAlloacted`. Additionally, the function `ClaimProjecToken()` has a typo in the name, which should be `ClaimProjectToken()` instead.

```
143 function getClaimable(address account) public view returns (uint256 claimableAmount) {
144     UserInfo storage userInfo = userInfos[account];
145     uint256 userAlloacted = (userInfo.userBidAmount * (10 ** projectTokenDecimals)) /
            clearingPrice();
146
147
148     if (userAlloacted > 0 && cliffEndTime != 0) {
149         uint256 nonVestedAmount = userAlloacted * (DENOMINATOR - vestingPercentage) /
                DENOMINATOR;
150         uint256 vestedAmount = 0;
151         if (block.timestamp >= cliffEndTime) {
152             uint256 totalVestingAmount = (userAlloacted * vestingPercentage) / DENOMINATOR;
153             vestedAmount = (block.timestamp - cliffEndTime) * totalVestingAmount /
                    vestingPeriodDuration;
154             if (vestedAmount >= totalVestingAmount) {
155                 vestedAmount = totalVestingAmount;
156             }
157         }
158
159
160         claimableAmount = nonVestedAmount + vestedAmount - userInfo.userClaimedProjectToken;
161     }
162 }
```

Listing 2.16: contracts/DutchAction.sol

```
194   function ClaimProjecToken() external whenNotPaused nonReentrant {
195       if (!claimPhaseStart) revert ClaimPhaseNotStart();
196       uint256 claimableAmount = getClaimable(msg.sender);
197       if (claimableAmount == 0) revert NoMoreClaimbleProjectTokens();
198
199
200       userInfos[msg.sender].userClaimedProjectToken += claimableAmount;
201       IERC20(projectToken).transfer(msg.sender, claimableAmount);
202
203
204       emit ClaimedProjectToken(msg.sender, claimableAmount);
205   }
```

Listing 2.17: contracts/DutchAction.sol

Suggestion Revise the typo.

2.3.6 Avoid precision losses in function `getClaimable()`

Status Fixed in `Version 5`

Introduced by `Version 4`

Description In the contract `DutchAction`, the function `getClaimable()` calculates the `totalVestingAmount` with the formula `(userAlloacted * vestingPercentage) / DENOMINATOR`. However, the calculation may suffer from precision losses, making `nonVestedAmount + totalVestingAmount` not equal to `userAllocated`. Calculating the `totalVestingAmount` by `userAlloacted - nonVestedAmount` is recommended.

```
143   function getClaimable(address account) public view returns (uint256 claimableAmount) {
144       UserInfo storage userInfo = userInfos[account];
145       uint256 userAlloacted = (userInfo.userBidAmount * (10 ** projectTokenDecimals)) /
              clearingPrice();
146
147
148       if (userAlloacted > 0 && cliffEndTime != 0) {
149           uint256 nonVestedAmount = userAlloacted * (DENOMINATOR - vestingPercentage) /
                  DENOMINATOR;
150           uint256 vestedAmount = 0;
151           if (block.timestamp >= cliffEndTime) {
152               uint256 totalVestingAmount = (userAlloacted * vestingPercentage) / DENOMINATOR;
153               vestedAmount = (block.timestamp - cliffEndTime) * totalVestingAmount /
                      vestingPeriodDuration;
154               if (vestedAmount >= totalVestingAmount) {
155                   vestedAmount = totalVestingAmount;
156               }
157           }
158
159
160           claimableAmount = nonVestedAmount + vestedAmount - userInfo.userClaimedProjectToken;
161       }
162   }
```

Listing 2.18: contracts/DutchAction.sol

Suggestion Change the calculation to `userAlloacted - nonVestedAmount`.

2.4 Note

2.4.1 Potential centralization risk

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     if(_rewards.length != _lps.length) revert InputDataLengthMissMatch();
218     if(baseTokens.length != _kyBarExectCallData.length) revert InputDataLengthMissMatch();
219
220     uint256 userTotalPendleRewardToSendBack;
221     uint256 userTotalPendleRewardToConvertMpendle;
222     uint256[] memory userPendleRewardsForCurrentMarket = new uint256[](_lps.length);
223
224     for(uint256 k; k < _lps.length;k++)
225     {
226         (,,,userPendleRewardsForCurrentMarket[k]) = masterPenpie.pendingTokens(_lps[k], msg.
                sender, PENDLE);
227     }
228
229     if(compoundingMode.length != userPendleRewardsForCurrentMarket.length) revert
            InputDataLengthMissMatch();
230
```

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

Listing 2.19: 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       if (mPendleMarket == address(0)) revert mPendleMarketNotSet();
225
226       (uint256 rewardToSend, uint256 bonusARBReward) = this.quoteConvert(_amount, msg.sender);
227
228       _convert(msg.sender, _amount);
229       uint256 treasuryFeeAmount = (IERC20(mPENDLE).balanceOf(address(this)) - _amount) *
              treasuryFee / DENOMINATOR;
230       uint256 mPendleToTransfer = _mPendleTransferAndLock(msg.sender, IERC20(mPENDLE).balanceOf(
              address(this)) - treasuryFeeAmount);
231
232       if (mPendleToTransfer > 0) {
233           if (_convertMode == CONVERT_TO_MPENDLE) {
234               IERC20(mPENDLE).safeTransfer(msg.sender, mPendleToTransfer);
235           } else if (_convertMode == LIQUIDATE_TO_PENDLE_FINANCE) {
236               _ZapInmPendleToMarket(mPendleToTransfer, _pdexparams);
237           } else {
238               revert InvalidConvertMode();
239           }
240       }
241
242       if (treasuryFeeAmount > 0){
243           IERC20(mPENDLE).safeTransfer(owner(), treasuryFeeAmount);
244       }
245
246       UserInfo storage userInfo = userInfos[msg.sender];
247       userInfo.converted += _amount;
248       userInfo.rewardClaimed += (rewardToSend - bonusARBReward);
249       userInfo.bonusRewardClaimed += bonusARBReward;
250       totalAccumulated += _amount;
251       userInfo.convertedTimes += 1;
252
253       ARB.safeTransfer(msg.sender, rewardToSend);
254
255       emit ARBRewarded(msg.sender, rewardToSend);
256   }
```

Listing 2.20: 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       IERC20(PENDLE).safeTransferFrom(msg.sender, address(this), _amount);
86       IERC20(PENDLE).safeApprove(address(vePendle), _amount);
87
88       uint128 unlockTime = _getIncreaseLockTime();
89       IPVotingEscrowMainchain(vePendle).increaseLockPositionAndBroadcast{value:msg.value}(uint128(
             _amount), unlockTime, chainId);
90
91       uint256 mintedVePendleAmount = accumulatedVePendle() -
92           preVePendleAmount;
93       emit PendleLocked(_amount, lockPeriod, mintedVePendleAmount);
94
95       return mintedVePendleAmount;
96   }
```

Listing 2.21: 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       IERC20(pendle).safeApprove(pendleStaking, allPendle);
46
47       uint256 mintedVePendleAmount = IPendleStaking(pendleStaking)
48           .convertPendle{ value: msg.value }(allPendle, chainId);
49
50       emit PendleConverted(allPendle, mintedVePendleAmount);
51   }
```

Listing 2.22: 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     pool.accPenpiePerShare =
445         pool.accPenpiePerShare +
446         ((penpieReward * 1e12) / lpSupply);
447     pool.lastRewardTimestamp = block.timestamp;
448
449     emit UpdatePool(
450         _stakingToken,
451         pool.lastRewardTimestamp,
452         lpSupply,
453         pool.accPenpiePerShare
454     );
455   }
```

Listing 2.23: 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       _provisionReward(_amountReward, _rewardToken);
256   }
```

Listing 2.24: 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       uint256 totalStake = totalStaked();
295       if (totalStake == 0) {
296           rewardInfo.queuedRewards += _amountReward;
297       } else {
298           if (rewardInfo.queuedRewards > 0) {
299               _amountReward += rewardInfo.queuedRewards;
300               rewardInfo.queuedRewards = 0;
301           }
302           rewardInfo.rewardPerTokenStored =
303               rewardInfo.rewardPerTokenStored +
304               (_amountReward * 10**receiptTokenDecimals) /
305               totalStake;
306       }
307       emit RewardAdded(_amountReward, _rewardToken);
308   }
```

Listing 2.25: 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.lastRewardTimestamp` 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     for (uint256 i = 0; i < _markets.length; i++) {
727         if (!pools[_markets[i]].isActive) revert OnlyActivePool();
728         Pool storage poolInfo = pools[_markets[i]];
729
730         poolInfo.lastHarvestTime = block.timestamp;
731
732         address[] memory bonusTokens = IPendleMarket(_markets[i]).getRewardTokens();
733         uint256[] memory amountsBefore = new uint256[](bonusTokens.length);
734
735         for (uint256 j; j < bonusTokens.length; j++) {
736             if (bonusTokens[j] == NATIVE) bonusTokens[j] = address(WETH);
737
738             amountsBefore[j] = IERC20(bonusTokens[j]).balanceOf(address(this));
739         }
740
741         IPendleMarket(_markets[i]).redeemRewards(address(this));
742
743         for (uint256 j; j < bonusTokens.length; j++) {
```

```
744             uint256 amountAfter = IERC20(bonusTokens[j]).balanceOf(address(this));
745
746             uint256 originalBonusBalance = amountAfter - amountsBefore[j];
747             uint256 leftBonusBalance = originalBonusBalance;
748             uint256 currentMarketHarvestPendleReward;
749
750             if (originalBonusBalance == 0) continue;
751
752             if (bonusTokens[j] == PENDLE) {
753                 currentMarketHarvestPendleReward =
754                     (originalBonusBalance * harvestCallerPendleFee) /
755                     DENOMINATOR;
756                 leftBonusBalance = originalBonusBalance - currentMarketHarvestPendleReward;
757             }
758             harvestCallerTotalPendleReward += currentMarketHarvestPendleReward;
759
760             _sendRewards(
761                 _markets[i],
762                 bonusTokens[j],
763                 poolInfo.rewarder,
764                 originalBonusBalance,
765                 leftBonusBalance
766             );
767         }
768     }
```

Listing 2.26: 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).

2.4.11 Function `_convertPendleTomPendle` may lead to users' assets loss

Introduced by `Version 1`

Description The function `_convertPendleTomPendle()` will directly convert `PENDLE` to `mPENDLE` with a ratio of 1:1 since `smartPendleConvert` is currently set as `address(0)`. However the current ratio of `PENDLE` and `mPENDLE` in `pancake` is 1:3.24553 at UTC 2024-10-15 05:34:27. Thus this might lead to users' assets loss.

```
825   function _convertPendleTomPendle(uint256 _pendleAmount) internal returns(uint256 mPendleToSend
          ) {
826       uint256 mPendleBefore = IERC20(mPendleOFT).balanceOf(address(this));
827
828       if (smartPendleConvert != address(0)) {
829           IERC20(PENDLE).safeApprove(smartPendleConvert, _pendleAmount);
830           ISmartPendleConvert(smartPendleConvert).smartConvert(_pendleAmount, 0);
831           mPendleToSend = IERC20(mPendleOFT).balanceOf(address(this)) - mPendleBefore;
832       } else {
833           IERC20(PENDLE).safeApprove(mPendleConvertor, _pendleAmount);
834           IConvertor(mPendleConvertor).convert(address(this), _pendleAmount, 0);
835           mPendleToSend = IERC20(mPendleOFT).balanceOf(address(this)) - mPendleBefore;
836       }
837   }
```

Listing 2.27: contracts/pendle/PendleStakingBaseUpg.sol

Feedback from the Project N/A.

2.4.12 `DutchAuction` owner fully controls the project tokens

Introduced by `Version 4`

Description In the contract `DutchAuction`, `projectToken` is not transferred in the function `_DutchAuction_init()`. This omission can result in failures when attempting to claim the `projectToken`, due to an insufficient balance of the `projectToken`.

Furthermore, the function `withdrawUnsoldProjectToken()` allows the owner to withdraw all the `projectTokens` in the contract. It doesn't reserve the amount of the tokens have been sold and not been claimed yet, which may also lead to failures of user claims.

```
74  function _DutchAuction_init(
75      address _projectToken,
76      address _bidToken,
77      uint256 _totalProjectToBid,
78      uint256 _startingPrice,
79      uint256 _minPrice,
80      uint256 _priceInterval,
81      uint256 _priceDecrementPrcnt,
82      uint256 _AuctionStartTime
83  ) public initializer {
84      __Ownable_init();
85      __ReentrancyGuard_init();
86      __Pausable_init();
87      projectToken = _projectToken;
88      bidToken = _bidToken;
89      totalProjectToBid = _totalProjectToBid;
90      startingPrice = _startingPrice;
91      minPrice = _minPrice;
92      priceInterval = _priceInterval;
93      priceDecrementPrcnt = _priceDecrementPrcnt;
94      auctionStartTime = _AuctionStartTime;
95      projectTokenDecimals = ERC20(projectToken).decimals();
96      bidTokenDecimals = ERC20(bidToken).decimals();
97  }
```

Listing 2.28: contracts/DutchAuction.sol

```
244  function withdrawUnsoldProjectToken() external onlyOwner nonReentrant {
245      uint256 balanceOfProjectToken = IERC20(projectToken).balanceOf(address(this));
246      IERC20(projectToken).transfer(msg.sender, balanceOfProjectToken);
247  }
```

Listing 2.29: contracts/DutchAuction.sol

Feedback from the Project Admin will manually transfer the total project for sale to the auction contract after its deployment. And the admin will make sure there are enough `projectTokens` for user claims.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS