Magpie

SMART CONTRACTS REVIEW

zokyo

September 24th 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING MAGPIE

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 1 Medium issue: 1 acknowledged = - 3 points deducted
- 1 Low issue: 1 acknowledged = - 1 points deducted
- 2 Informational issues: 2 resolved = 0 points deducted

Thus, 100 - 3 - 1 = 96

TECHNICAL SUMMARY

This document outlines the overall security of the Magpie smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Magpie smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Magpie team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Magpie repository:
Repo: https://github.com/magpiexyz/penpie-contracts/pull/181

Last commit - fa3a5359a0e7feb07de74422c2f8ddb1e634b5f6

Scope for Hack security review:

- **PendleStaking (ETH, ARB, BNB, OPT):**
  - Functions to review:
  - updateAllowedPauser()
  - batchRemovePools() (ETH & ARB)
  - updatePoolHelper() (ETH & ARB)
  - setAffectedMarketWithdrawRatio() (ETH & ARB)
  - emergencyWithdraw() (ETH & ARB)
- **MasterPenpie (ETH, ARB, BNB, OPT):**
  - Functions to review:
  - updateAllowedPauser()
  - removeEvilPools() (ETH & ARB)
  - makePoolInactive() (ETH & ARB)
- **PendleMarketDepositHelper (ETH & ARB):**
  - Functions to review:
  - emergencyWithdraw()
  - removePoolLogic()
- **Pauser Management:**
  - Contracts:
  - PendleStaking (all chains)
  - MasterPenpie (all chains)
  - Function to review:
  - updateAllowedPauser()

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Magpie smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contract/s by industry leaders.
03	Thorough manual review of the codebase line by line.

Executive Summary

The Penpie developers have fixed the bug by adding a reentrancy check in the malicious methods `harvestMarketRewards()` and `batchHarvestMarketRewards()` in the `PendleStaking.sol` contract. Additionally, a new method, `batchRemovePools`, has been added to remove the malicious pools introduced by the attacker. The method `updatePoolHelper` has also been modified to mark the affected pools as inactive and set their allocation points to zero.

Users will be allowed to perform emergency withdrawals from the affected pools based on the `affectedMarketWithdrawRatio` set by the protocol.

Finally, for security reasons, the `pause()` method can now be called by multiple authorized addresses, allowing deposits, withdrawals, and reward harvesting to be paused immediately in the event of such attacks.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Magpie team and the Magpie team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Gas Excessive Loop in removePool Function	Medium	Acknowledged
2	Method updatePoolHelper is not removing pool info from the last helper contract	Low	Acknowledged
3	Unused modifier	Informational	Resolved
4	Suggestion to add the reentrant modifier in the internal method _harvestBatchMarketRewatds	Informational	Resolved

Gas Excessive Loop in `removePool` Function

In the `removePool` function of the MasterPenpie contract, the function iterates through the `registeredToken` array to locate and remove the `_stakingToken`. This loop has a time complexity of O(n), where n is the number of tokens in the `registeredToken` array. If the array grows large, the gas cost associated with this loop can increase significantly, making the transaction potentially too expensive or even failing due to out-of-gas errors.

The specific section of concern is:

```
uint256 length = registeredToken.length;
for (uint i = length; i > 0; i--) {
    if (registeredToken[i-1] == _stakingToken) {
        if ((i - 1) != (length - 1)) {
            registeredToken[i - 1] = registeredToken[length - 1];
        }
        registeredToken.pop();
        break;
    }
}
```

This loop performs a linear search to find the `_stakingToken`, which becomes inefficient if the array has many entries. As the number of pools grows, this will lead to higher gas costs.

Recommendation:

To mitigate the gas costs, consider replacing the array with a more gas-efficient data structure, such as a **mapping of registered tokens to their index** in the array. This would allow you to achieve constant time complexity (O(1)) when removing a token from the array.

LOW-1 | ACKNOWLEDGED

Method updatePoolHelper is not removing pool info from the last helper contract

In Contract PendleStakingBaseUpg.sol, updatePoolHelper(...) sets a new helper contract for a market.
In the logic, pool info is being set in the new helper contract, pool info is not removed from the previous helper contract.
Although this doesn't seem to cause any serious issues, it is advised to do so if possible.

Recommendation:
Remove the pool info from the previous helper contract when setting a new one for the market.

INFORMATIONAL-1 | RESOLVED

Unused modifier

In Contract PendleStakingBaseUpg.sol, the following modifier is unused after the fix updates.

```
modifier _onlyPoolHelper(address _market) {
        Pool storage poolInfo = pools[_market];

        if (msg.sender != poolInfo.helper) revert OnlyPoolHelper();
        _;
    }
```

Recommendation:
Remove the unused modifier.

Suggestion to add the reentrant modifier in the internal method _harvestBatchMarketRewatds

In Contract PendleStakingBaseUpg.sol, methods harvestMarketRewards() and batchHarvestMarketRewards() are using the internal method _harvestbatchMarketRewards().
Instead of adding a `nonReentrant` modifier to all the external methods using the internal method, adding the same in the internal method is advised.

Recommendation:
Add the nonReentrant modifier in the internal method to avoid duplication of the same modifier in all the external methods.

Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the Magpie team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Magpie team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo