HOLYHELD

Holyheld Security Analysis

by Pessimistic

This report is public.

Published: November 20, 2020

Abstract

In this report, we consider the security of the smart contracts of Holyheld project. Our task is to find and describe security issues in smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, security audit is not an investment advice.

Summary

In this report, we considered the security of the smart contracts of Holyheld project. We performed our audit according to the procedure described below.

The initial analysis showed a discrepancy with the documentation, and several issues of low severity. The owner of **HolyPaladin** contract can change or stop withdrawals. However, the owner accesses the contract via multisig which decreases chances of compromise.

The contracts have already been deployed to addresses mentioned in README.md file in the repository, the deployed code matches the code from repository, commit c6970bd522f49019e5af9a87e78c935b8d62b34f.

The audit did not reveal any possibilities of unauthorized withdrawals of assets. The code processes yCRV correctly.

General recommendations

We highly recommend eliminating discrepancies with the documentation, implementing tests, and fixing possible underflow. We also recommend following CEI-pattern in future versions of the code.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether code logic corresponds to the specification.
2. Whether the code is secure.
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan project's code base with automated tools: MythX and SmartCheck.
  - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
  - We inspect the specification and check whether the logic of smart contracts is consistent with it.
  - We manually analyze code base for security vulnerabilities.
  - We assess overall project structure and quality.
- Report
  - We reflect all the gathered information in the report.

Project overview

Project description

For the analysis, we were provided with code base on GitHub repository of Holyheld project, commit c6970bd522f49019e5af9a87e78c935b8d62b34f.

The contracts have already been deployed to addresses mentioned in README.md.

The documentation for the project was provided as a README.md file on the same repository.

The project has no tests.

The total LOC of audited sources is 478.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger smart contracts security. We highly recommend fixing them.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence project operation in current implementation. We highly recommend addressing them.

Discrepancy with documentation

The documentation states:

```
10% of the total supply or 10,000,000 HOLY will be reserved for operational
and marketing expenses. This reserve will also be vested at a 2% per week
unlock rate. Supporting the transparency spirit of DeFi, we will announce
all major operational costs in advance.
```

However, 1M tokens are reserved without vesting.

No tests

The project has no tests. Testing is crucial for code security and audit does not replace tests in any way.

We highly recommend both covering the code with tests and making sure that the test coverage is sufficient.

Low severity issues

Low severity issues can influence project operation in future versions of code. We recommend taking them into account.

Code quality

- In **HolyKnight** contract, `putToTreasuryAmount()` function is called within `putToTreasury()`. However, both functions perform the same checks. Thus, when `putToTreasury()` function is called, these checks are performed twice.

  We recommend completely separating these functions to improve code readability and reduce gas consumption.

- `revoke()` function of **HolderVesting** contract is unused. Also, the whole contract is used only with `_revocable` variable set to `false`. Thus, all the functionality connected to `_revocable` is redundant.
- There is a possibility of an underflow at line 358 of **HolyKnight** contract. In the current implementation, this underflow is unreachable if used with the tokens mentioned in the documentation. However, we recommend using safeMath here to avoid any issues in future versions of code or if the contract is used with other tokens.
- Consider using `external` visibility level instead of `public` where possible.

Code logic

In **HolyKnight** contract, approving infinity (`0xfff...fff`) once might not be enough.

We recommend checking approval and reapprove infinity if necessary.

Misleading comment

In **HolyKnight** contract, line 128 states:

```
initially, 10% of tokens are reserved for future pools to be added
```

However, 20% of tokens are reserved initially.

Missing input validation

- In `putToTreasuryAmount()` function of **HolyKnight** contract, `_amount` parameter is not checked to be non-zero.
- There is a comment for `add()` function of **HolyKnight** contract at line 145 stating

  ```
  DO NOT add the same LP token more than once. Rewards will be messed
  up if you do.
  ```

  However, no checks added to exclude this scenario.

Violation of checks-effects-interactions pattern

In **HolyKnight** contract, functions `deposit()`, `withdraw()`, and `emergencyWithdraw()` violate checks-effects-interactions pattern and thus are susceptible to reentrancy attack.

We recommend using reentrancyGuard in these functions.

The contracts have already been deployed. Therefore, for these contracts, it does not threaten security. However, we recommend following CEI-pattern to avoid this issue in future versions of the code.

Notes

Overpowered owner

The owner of **HolyPaladin** contract can modify reward rate or completely stop rewards for any pool.

However, the owner accesses the contract via multisig, which significantly decreases chances of compromise.

This analysis was performed by Pessimistic:

Igor Sobolev, Security Engineer
Evgeny Marchenko, Senior Security Engineer
Boris Nikashin, Analyst
Alexander Seleznev, Founder

November 20, 2020