EQLAB

Warden Yield Security Analysis

by Pessimistic

This report is public

July 10, 2024

Abstract 2
Disclaimer 2
Summary 2
General recommendations 2
Project overview 3
Project description 3
Codebase update #1 3
Audit process 4
Manual analysis 5
Critical issues 5
Medium severity issues 6
M01. Overpowered role (addressed) 6
M02. USDT approve (fixed) 6
Low severity issues 7
L01. Checking AAVE results (commented) 7
L02. Public external (fixed) 7
L03. Redundant import (fixed) 7
L04. Returned value of approve not checked (fixed) 7
Notes 8
N01. Discrepancy with documentation (commented) 8
N02. Possible force undelegation (commented) 8

Abstract

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

Disclaimer

The audit does not give any warranties on the security of the code. A single 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, a security audit is not investment advice.

Summary

In this report, we considered the security of Warden Yield smart contracts. We described the audit process in the section below.

The audit showed two issues of medium severity: Overpowered role, USDT approve. Also, several low-severity issues were found.

After the initial audit, the developers updated the codebase. In this update, they fixed or commented on all the issues from the previous review. No new issues were found on the recheck.

The overall code quality of the project is good.

General recommendations

We do not have any recommendations for the project.

Project overview

Project description

For the audit, we were provided with Warden Yield project on a public GitHub repository, commit ddc3eda7ff437deaebabcaa4abcb6624ac40f559.

The scope of the audit included **contracts** folder, excluding **contracts/test** subfolder.

The documentation for the project included a private document with the project description.

All 32 tests pass successfully. The code coverage is 100%.

The total LOC of audited sources is 457.

Codebase update #1

After the initial audit, the developers provided us with a new version of the code, commit 7120c14dbe2646f52463d2f0c2439369b422aa91. In this update, the developers introduced fixes or provided additional comments for all the issues from the initial report.

34 tests out of 34 pass successfully. The code coverage is 100%.

Audit process

We started the audit on June 28, 2024 and finished on July 3, 2024.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project. After a discussion, we performed preliminary research and started the review.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among others, we verified the following properties of the contracts:

- Whether Lido integration performed correctly;
- Whether EigenLayer integration performed correctly;
- Whether AAVE integration performed correctly;
- Whether all the tokens from the documentation work correctly supplying to AAVE;
- Whether an arbitrary user could not withdraw funds if not allowed;
- Whether initialization of contracts performed correctly.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts. We also sent the results to the developers in the text file.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

After the initial audit, we discussed the results with the developers. On July 8, 2024, the developers provided us with an updated version of the code. In this update, they fixed some of the issues from our report and commented on the others.

We reviewed the updated codebase and scanned it with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules.

As a result of the manual review, no new issues were found.

Finally, we updated the report.

Manual analysis

The contracts were completely manually analyzed, and 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 project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The audit showed no critical issues.

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. Overpowered role (addressed)

In the current implementation, the system depends heavily on the owner's role. The owner is responisble for:

- **(removed)** Allowing and disallowing tokens supplying to AAVE;
- **(removed)** Enable and disable withdrawals from the **AaveYield** contract;
- Upgrading contracts implementations through proxies.

Thus, some scenarios can lead to undesirable consequences for the project and its users, e.g., if the owner's private keys become compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> *Comment from the developers: We have removed all methods that require admin rights, except for the rights to update the implementation.*

M02. USDT approve (fixed)

According to the documentation, several stablecoins will be used for supplying to the AAVE pool. However, approving **USDT** at line 74 of the **AaveInteractor** contract will fail as **USDT** implementation does not return `bool` status on `approve` calls. Consider using `forceApprove` function for token approvals.

The issue has been fixed and is not present in the latest version of the code.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Checking AAVE results (commented)

AAVE mints and redeems tokens on 1:1 basis. Consider removing additional checks for mint successes to reduce gas consumptions at lines 77–78 and 95–96 in the **AaveInteractor** contract.

> *Comment from the developers: We decided to keep this for additional verification of the call result.*

L02. Public external (fixed)

Some of the functions could be turned from public to external to reduce gas consumptions in the **YieldStorage** contract:

- `YieldStorage.totalShares`;
- `YieldStorage.totalStakedAmount`;
- `YieldStorage.userStakedAmount`;
- `YieldStorage.wardenAddress`.

The issues have been fixed and are not present in the latest version of the code.

L03. Redundant import (fixed)

Math.sol library is imported but never used at line 5 in the **LidoInteractor** contract. Consider removing it.

The issue has been fixed and is not present in the latest version of the code.

L04. Returned value of approve not checked (fixed)

There are places in the code where return value of `IERC20.approve` is not checked:

- In `AaveInteractor._aaveStake` function at line 74;
- In `EigenLayerInteractor._eigenLayerRestake` function at line 65.

It is safe to use `approve` like that with the tokens specified in the documentation, however future integrations could be affected by this issue.

The issues have been fixed and are not present in the latest version of the code.

Notes

N01. Discrepancy with documentation (commented)

There are many user flows in the documentation that are not yet implemented in the contracts. It is possible to add additional functionality later through a proxy pattern; however, consider checking that the first version of the project contains all the desired functions.

> *Comment from the developers:* *Acknowledged.*

N02. Possible force undelegation (commented)

In `EigenLayerInteractor.__EigenLayerInteractor_init` there is delegation of the tokens to `operator` via `IDelegationManager.delegateTo` call. But it is important to mention that `operator` and their `delegationApprover` can undelegate any staker. Currently, no functionality allows to redelegate undelegated tokens.

> *Comment from the developers:* *We decided not to add the possibility of redelegation in the first version of the contract due to the complexity of the process itself; we plan to implement this in future versions.*

This analysis was performed by Pessimistic:

Oleg Bobrov, Security Engineer
Evgeny Bokarev, Junior Security Engineer
Konstantin Zherebtsov, Business Development Lead
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

July 10, 2024