Opium Pools Security Analysis

by Pessimistic

This report is public

January 25, 2023

Abstract

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

Disclaimer

Summary

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

The audit showed three issues of medium severity: Missing fees in calculations, Overpowered owner, and Tests issues. Also, two low-severity issues were found.
The overall code quality is good.

After the initial audit, the codebase was updated.
In this update, the developers fixed or commented all of the previously discovered issues.
Also, all tests passed and the code coverage increased.

After the recheck #1, the developers found the vulnerable scenario. The point was that the `rageQuit` function could be used to bypass the payment of fees to the protocol. This issue was fixed by adding the fee to the `rageQuit` function at the commit a63c7139529ff86198676dfbbb9f6239fe2b1634.

General recommendations

We have no further recommendations.

Project overview

Project description

For the audit, we were provided with the Opium Pools project on a private GitHub repository, commit 8a4e22985db2d06fd6a7b46e56828b1e5cf87940.

The scope of the audit includes:

- base/RegistryManager.sol;
- modules/strategies/OptionsSellingStrategyModule.sol;
- modules/AccountingModule.sol;
- modules/LifecycleModule.sol;
- modules/RegistryModule.sol;
- modules/StakingModule.sol;

The documentation for the project includes a Google Docs file.

32 of 33 tests pass, the code coverage for the audited scope is 79,94%.

The total LOC of audited sources is 1091.

Codebase update #1

After the initial audit, we were provided with commit 9ccbeaf864e6bf2b9ba87acfd4d41bae95beb29c.

In this update, the developers fixed all issues. All 33 tests passed and the code coverage increased to 90.21%.

Procedure

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

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan the project's codebase with the automated tool Slither.
  - We manually verify (reject or confirm) all the issues found by the tool.

- Manual audit
  - We manually analyze the codebase for security vulnerabilities.
  - We assess the overall project structure and quality.

- Report
  - We reflect all the gathered information in the report.

Inter alia, we verify that:

- No standard Solidity issues are present in the codebase.
- The liquidity is correctly updated for deposits, withdrawals, and at the end of the strategy.
- The fees are accounted for properly.
- The user can correctly exit the project by taking the share of unused underlying tokens and positions.
- The process of transition between phases and epochs is correct.
- Scheduled and regular withdrawals and deposits are processed correctly.
- The price-fixing in epochs works correctly.
- The integration with Zodiac is proper.
- The implementation of the EIP4626 is proper.

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 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 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. Missing fees in calculations (fixed)

When users call the `rageQuit` function of the **StakingModule** contract, they can withdraw some of the tokens that belong to the project. The calculation of needed assets does not consider the strategy fees that may remain in the vault balance after the end of the trading phase. As a result, the user will withdraw part of the fees.

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

M02. Overpowered owner

In the **RegistryManager** contract, owner can change the address of the **RegistryModule**.

In the **RegistryModule** contract, owner can change the address of the **RegistryManager**.

In the **OptionsSellingStrategyModule** contract, `ADVISOR_ROLE` can change strategy parameters.

In the **AccountingModule** contract owner can:

1. Change the maintenance fee;
2. Change the immediate profit fee;
3. Change the fee collector address;

This may lead to an exploit when the owner withdraws profit and liquidity from the Vault.

The exploit example is described below:

The owner frontruns the `rebalance` call in the strategy contract, sets high fees, and sets its address for the fee withdrawal.

We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> *Comment from the developers:*
>
> *Regarding Ownable contracts: Modules owners are not going to be neither private keys nor multisigs. All "Owners" in the smart contracts will be set to the Vault (Gnosis Safe) address. The Vault itself will be controlled by DAO.*
>
> *Regarding Advisory role: The provided Strategy for Options Trading is just a reference and will not be used in production. Production strategies will be implemented separately and Advisors there will have more restrictions in their actions.*

M03. Tests issues (fixed)

Below are several issues regarding the project tests:

- One of the tests does not pass.
- **OptionsSellingStrategyModule** contract is almost not covered with tests.
- The percentage of passing tests may change from time to time.

After interaction with the developers, all the tests have passed and their number has not changed.

> *Comment from the developers: Regarding test coverage for the strategy module: See the reason of the strategy contract in M02. It will not be used in production as is.*

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. Code quality - CEI (fixed)

The CEI pattern is violated in several places of the project:

1. `safeTransferFrom` is called before updating storage variables in the `scheduleDeposit` function of the **StakingModule** contract.
2. External calls occur before updating the storage variables `totalScheduledDeposits` and `totalScheduledWithdrawals` in the `postRebalancing` function of the **StakingModule** contract.

We highly recommend following the CEI pattern to increase the predictability of the code execution and protect from some types of re-entrancy attacks.

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

L02. Code quality - external (fixed)

Consider declaring function `mintRef` of the **StakingModule** contract as `external` instead of `public` to improve code readability.

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

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Daria Korepanova, Security Engineer
Nikita Kirillov, Security Researcher
Irina Vikhareva, Project Manager
Alexander Seleznev, Founder

January 25, 2023