Clearpool Security Analysis

by Pessimistic

This report is public

March 21, 2022

Abstract

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

The audit showed one critical issue — Optional debt refund; several issues of medium severity, including Overpowered role, ERC20 standard violation, a Bug, and several issues of low severity.

After the initial audit, the codebase was updated. In this update, the developers fixed some of the previously discovered issues. However, several new issues were found. Also, the code coverage has decreased.

After the recheck #1, another codebase update was performed. This update included fixes but also introduced a few new issues.

After the recheck #2, codebase update #3 was performed. This update included fixes for issues from previous recheck.

The documentation for the project is incomplete.

General recommendations

We recommend fixing the rest of the issues and improving the documentation for the project.

Project overview

Project description

For the audit, we were provided with Clearpool project on a private GitHub repository, commit 66f19392bbb54b7fbcd3fa9f62793a028b3ef056.

The documentation for the project is available by link: https://solar-kidney-3f8.notion.site/Architecture-Overview-fb59cb0f19634dd991f7c2f33f7f9be2. However, we consider this documentation incomplete.

The project has tests. All tests pass, the overall code coverage is 97.33%.

The total LOC of audited sources is 1588.

Codebase update #1

After the initial audit, we were provided with commit 0744159d11d59999cfa7710af519f7c780d90cbe.

In this update, the developers fixed a several issues and added new functionality. However, in the updated codebase, we discovered new issues, including two issues of medium severity.

Codebase update #2

After the recheck #1, another codebase update was performed. For the recheck #2, we were provided with commit 093d8a063b12dfc39a088d192d16b43e39f170da.

In this update, the developers also fixed a few issues and added new functionality. The recheck showed new issues in the code. Regarding the tests, the overall code coverage decreased to 95.37%.

Codebase update #3

For the recheck #3, we were provided with updated codebase on commit 45ff5bf444136709832da05fcc177ce6b8818ee3.

This update included only fixes for issues from previous reports.

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 tools: Slither and SmartCheck.
  - We manually verify (reject or confirm) all the issues found by the tools.
- 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.

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.

C01. Optional debt refund (fixed)

Owners of pools do not have to refund their debts completely since they can just close pools. When an owner of a pool calls `repay` function of **PoolMaster** contract with minimal `amount` value and `closeNow` parameter set to `true`, the pool will be closed due to a call to internal `_close` function at lines 264–266.

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

Medium severity issues

Medium issues can influence project operation in 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)

The owner of **PoolFactory** contract can:

- Create new pools.
- Pull funds that were transferred to a pool.
- Update addresses of pools, the treasury, the contract that calculates fees.
- Change insurance and reserve.
- Tweak criterion of pool default event.
- Rewards for token holders.

The owner of **MembershipStaking** contract can change the amount of tokens required to stake when a pool is created.

The owner of **Auction** contract can whitelist addresses so they could start auctions.

The owner of **FlashGovernor** contract can change voting parameters and proposers list.

In the current implementation, the system depends heavily on owners of these contracts. Thus, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if private keys of any of the owners become compromised.

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

> *Comment from developers: first, the plan is to use multisig (gnosis safe). Later, the plan is to switch to DAO.*

M02. ERC20 standard violation (fixed)

ERC-20 standard states:

```
Callers MUST handle false from returns (bool success). Callers MUST
NOT assume that false is never returned!
```

However, returned values of `transfer` and `transferFrom` calls are not checked in:

- **PoolFactory** contract at line 362.
- **MemebershipStaking** contract at lines 83, 136, and 237.

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

M03. Bug (fixed)

We assume that `baseDelta` represents assets that have not been borrowed or reserved as interest, insurance, or reserves yet. In this case, it is incorrectly computed at line 384 of **PoolBase** contract. It should be additionally reduced by the current borrows value.

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

M04. Formula without comments (fixed)

The formula at line 386 of **PoolBase** contract is confusing. It is not trivial to derive it from the code or high-level description of the project. Consider adding comments to `_entranceOfWarningUtilization` function to improve code readability and avoid possible mistakes.

The developers added comments with explanation of the formula to the code.

M05. Incorrect default occurance condition (fixed)

The `_entranceOfProvisionalDefault` function of **PoolBase** contract has a mistake in the expression for default condition calculation. Consider the following notation:

- `f` — reserve factor + insurance factor.
- `P` — principal.
- `C` — cash.
- `B` — borrows.
- `ΔB` — change of borrows that will lead to default.
- `k` — `provisionalDefaultUtilization` value.

The `_entranceOfProvisionalDefault` function calculates the `ΔB` with the following formula: `ΔB = (k × (P + C) - B) / (1 + f)`.

The balance is `P + C - ΔB × f`. The condition for default is `k × (P + C - ΔB × f) = B + ΔB`. From this equation, `k × (P + C) - B = ΔB × (1 + k × f)`. Therefore, `ΔB = (k × (P + C) - B) / (1 + k × f)`.

This means that the formula in the code is missing `k` coefficient for `f` value.

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 the future versions of the code. We recommend taking them into account.

L01. Code quality (fixed)

In **MembershipStaking** contract, the use of `uint32` type in `Checkpoint` struct at line 19 does not make any sense if the second element is of `uint256` type. In this case, the struct will occupy two slots of storage. Consider choosing field types for `Checkpoint` struct that can be packed into a single storage slot to optimize gas consumption, e.g. `uint32` and `uint224` types. Alternatively, use `uint256` type for all fields to simplify the code.

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

L02. Code quality

CPOOL contract relies on short `uint96` type to pack `Checkpoint` struct into a single slot. However, utilizing short types in other context is inconvenient and error-prone. We recommend using `uint256` all over the code and only safely casting values to smaller types when working with storage.

> *Comment from developers: CPOOL contract was not changed as contract was already deployed.*

L03. Code quality

The CEI (checks-effects-interactions) pattern is violated in several places within the project's codebase. We highly recommend following CEI pattern to increase the predictability of the code execution and protect from some types of re-entrancy attacks in:

- **PoolFactory** contract, lines 161, 347, 420–421, and 441–442.
- **Auction** contract, lines 107–108, 192.
- **MembershipStaking** contract, line 109.
- **PoolMaster** contract, lines 173, 234, 260, 338, 341, and 551.

> *Comment from developers: Applying CEI pattern from our perspective isn't required in the mentioned places as all interacted contracts are trusted and contracts are instantiated in-place.*

L04. Code quality (fixed)

Consider declaring functions as `external` instead of `public` when possible to improve code readability and optimize gas consumption.

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

L05. Code quality (fixed)

Consider emitting an event when the owner of the factory changes rewards in **PoolMaster** contract at line 353.

The developers added an event to ownership transfer.

L06. Code quality

In **CPOOL** contract, consider using [OpenZeppelin implementation]() of EIP-712 instead of re-implementing it at lines 205–218.

> *Comment from developers: CPOOL contract was not changed as contract was already deployed.*

L07. Code quality

Consider using [OpenZeppelin ECDSA]() library for replay protection at line 219 of **CPOOL** contract.

> *Comment from developers: CPOOL contract was not changed as contract was already deployed.*

L08. Code quality

Anyone can initialize **FlashGovernor** contract right after the deployment. Make sure that the contract is initialized correctly before its address is passed to any other contract.

L09. Code quality

`onlyActiveAccrual` modifier writes to storage at line 465 of **PoolBase** contract. In general, modifiers should not modify storage since it complicates code structure. Consider using modifiers only to perform checks, and utilize internal functions to reuse non-view logic.

L10. Code quality (fixed)

The same code is executed twice in a row at lines 145, 147 in `_distributeReward` function of **PoolRewards** contract.

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

L11. Incomplete documentation

The project has a documentation. However, it does not cover some important logic of the project, e.g., a motivation for participating an auction, borrow interests, upgradability and proxies, etc. The documentation is required to streamline both development and audit processes. A proper documentation should explicitly explain the purpose and behavior of the contracts, their interactions, and main design choices.

Notes

N01. Design drawback

The winner of the auction or the owner of the corresponding NFT token (if it was transferred) has no motivation to close the pool. Therefore, some of defaulted pools might remain open forever.

N02. Design drawback

Owners of a pools can act maliciously and increase their income. If they borrow maximum amount of tokens, this will trigger a pool default event. Then they win auctions and get the rewards. This way they benefit more than if they just refuse to repay the debt.

N03. Unsafe type casting (fixed)

Consider using SafeCast from OpenZeppelin to exclude overflow possibility at line 501 in `_proposePool` function of **PoolFactory** contract.

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

This analysis was performed by Pessimistic:

Daria Korepanova, Security Engineer
Evgeny Marchenko, Senior Security Engineer
Nikita Kirillov, Junior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

March 21, 2022