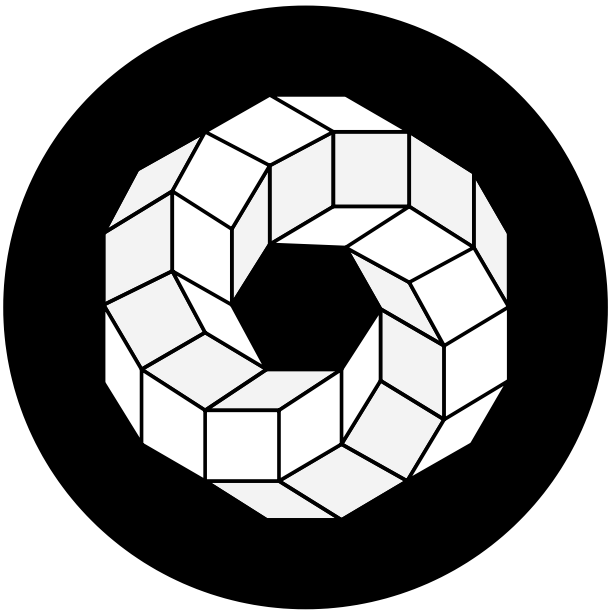

PowerPool Agent Security Analysis

by Pessimistic

This report is private

February 22, 2023

Abstract

In this report, we consider the security of smart contracts of PowerPool Agent 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 PowerPool Agent smart contracts. We described the audit process in the section below.

The audit showed multiple critical issues: Repeated slashings might fail, Job owners can force slashing, Malicious job can drain all stakes #1 and #2, and Weak randomness. The audit also revealed three issues of medium severity: Missed Edge case, Insufficient minimal job reward, and Front-running attack. Moreover, several low-severity issues were found.

The project should not be deployed before these issues are fixed.

General recommendations

We recommend fixing the mentioned issues and simplifying codebase by removing inheritance.

Project overview

Project description

For the audit, we were provided with PowerPool Agent project on a private GitHub repository, commit f4d444417a7166aa775e79b92965adfce160a639. During the audit, the codebase was updated and we moved to commit 6ba92b5048209918901846581 1ee2e0d8ea13f30.

The scope of the audit included:

1. **PPAgentV2Randao.sol** file,
2. Changes in the **PPAgentV2.sol** file from the previous audit.

The documentation for the project included the following files:

1. **Agent V2 Randao Specification (1).V1**, sha1sum is `a5eab5b517acc21032475157153f4a2f1037b2b5`,
2. **PP-AGENT-LITE-SPEC.txt**, sha1sum is `d360ed672d1c120a12c8b85309508c93d7d048c8`.

All 199 tests pass successfully. However, the code coverage could not be calculated due to the Foundry issue.

Audit process

We audited this earlier version of this project. The current stage of the audit started on February 7 and finished on February 22, 2023.

We inspected the materials provided for the audit. Then, we contacted the developers to learn more about introduced modifications.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

We manually analyzed all the contracts within the audit scope and checked their logic. Among other, we checked the contracts against the following attacks:

- Possibility to steal tokens from the contract;
- Possible malicious behavior of keepers and job owners;
- If the randomness can be manipulated;
- If the slashing and job assignment logic works as intended.

We scanned the project with the static analyzer Slither and our private plugin with an extended set of rules and then manually verified all the occurrences found by the tool.

We ran the project tests.

On February 13 the developers updated the codebase. We reviewed the changes and considered them in the audit.

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

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. Repeated slashings might fail

In the **PPAgentV2Randao** contract, a keeper can have multiple jobs. If they go offline, they will be slashed multiple times, which might completely drain their stake. In this case, consequent slashings will become impossible due to the check at line 435. As a result, other jobs assigned to this keeper will be stuck, as slashing is required to assign a new keeper.

C02. Job owners can force slashing

A malicious user can create a job that checks `tx.origin` and reverts for some addresses. The assigned keeper cannot release the job or otherwise protect themself.

If only the attacker's worker can execute this job, they can wait until `getCurrentSlasherId()` returns the user's address and slash a keeper that was assigned to this job.

C03. Malicious job can drain all stakes #1

It is possible to drain all `CVP` token stakes with the following attack:

1. An attacker creates a job where the job address is the Agent contract itself and a selector is the `stake` function. In this case, the `keeperId` argument represents the id of the keeper created by the attacker, and the `amount` parameter is how much the attacker wants to drain.

2. The assigned keeper executes the attack, and the Agent contract starts the staking operation. In the process, it transfers CVP tokens to itself, which counts as providing a stake.

3. The attacker is the keeper admin and thus can withdraw the stake.

Consider prohibiting specifying Agent address as `jobAddress`.

C04. Malicious job can drain all stakes #2

Similarly to C03, one can create three jobs that call `registerAsKeeper`, `initiateRedeem`, and `finalizeRedeem`. When executed, the Agent contract becomes the keeper's owner and calls itself to execute these operations. The recommendation from C03 works for this issue as well.

C05. Weak randomness

In the **PPAgentV2Randao** contract, users can affect the output of the `getSlasherIdByBlock` function by activating/deactivating a bunch of keepers. E.g., a malicious user can try to win a slashing contest for profit. The cost of such an attack grows linearly with the number of available slashers, while the success probability grows quadratically. Besides, an attacker can grieve the contract by front-running `initiateSlashing` calls with cheap `setKeeperActiveStatus` transactions. Among others, this attack allows to delay slashing, so it might never happen.

Note that `_assignNextKeeper` utilizes a similar random function, and attackers can manipulate it by activating/deactivating keepers. Multiple external functions rely on it, and attackers can always perform this attack, e.g., using MEV bundles.

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. Missed Edge case

In the **PPAgentV2Randao** contract, the `releaseJob` function allows the keeper to release an interval job without waiting for the required period for new jobs that were not executed. At line 193, `period2EndsAt` is assigned to `lastExecutionAt + rdConfig.period1 + rdConfig.period2`. However, for new jobs, the `lastExecutionAt` will be `0`. Therefore, it is possible to bypass the check at line 194 and release the job.

M02. Insufficient minimal job reward

Any user can create a job with `1e15 wei` payment, and an assigned keeper will have to execute this job to avoid slashing. Moreover, users can set a stake requirement high enough to assign this job to keepers with larger stakes. This will force keepers to execute such jobs due to the risks of significant losses.

M03. Front-running attack

A malicious user can front-run `setWorkerAddress` and `registerAsKeeper` transactions, setting the `worker` address of the victim keeper to the user's keeper. As a result, the victim's transaction will fail due to the requirement that the `worker` must not be assigned.

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. Redundant code

In the **PPAgentV2Randao** contract, the `_assertOnlyKeeperWorker` function is never used.

L02. Gas consumption

In the **PPAgentV2Randao** contract, the `_beforeExecute` function reads the `jobNextKeeperId` variable directly from storage at lines 373, 378, and 397. Consider using the `nextKeeper` local variable declared at line 373 instead, since reading from storage is expensive.

L03. Gas consumption

In the **PPAgentV2Randao** contract, the `releaseJob` function accepts `keeperId_` as the first argument. However, this value can be obtained from the `jobNextKeeperId[jobKey_]` value.

L04. Memory-safe assembly

All assembly blocks in the code are memory-safe. However, many of them do not include the corresponding annotation. Consider adding `memory-safe` indicator to assist the compiler in **PPAgentV2** and **PPAgentV2Randao** contracts.

L05. Missing check

The `stake` function of the **PPAgentV2** contract does not check if the specified keeper exists. Users can lose their assets if staking for an incorrect `keeperId`.

L06. Deprecated property

In the **PPAgentV2Randao** contract, the `_getPseudoRandom` function uses the `block.difficulty` property which is deprecated. Consider using the `block.prevrandao` instead.

L07. Keeper assignment issues

In a case when a user creates a task with a high stake requirement, and there is only one suitable keeper, it may cause problems if that keeper gets slashed (for another job) and the keeper's stake amount drops below the required stake for this task. This job will get stuck because nobody (including the current keeper) can execute this job since no keeper has the required stake amount, and the contract fails to assign the next keeper.

Also note that in a case when there will be lots of keepers and a small number of suitable keepers with the required stake, `_assignNextKeeper` might eat a lot of gas (and maybe cause a DoS because of the block gas limit) in `_assignNextKeeper` function of **PPAgentV2Randao** contract.

L08. Complicated code structure

PPAgentV2Randao contract inherits from **PPAgentV2**. It extends base functionality via hooks and `super` calls, which complicates code structure and does not improve modularity. Importantly, this code is not supposed to be extensible. We recommend combining these contracts into one and simplifying the execution flow.

Notes

N01. Exploitable randomness

The **PPAgentV2Randao** contract allows exploiting the usage of the `_getPseudoRandom` function. E.g., when a job owner is a contract, and it calls the `assignKeeper` function, the **PPAgentV2Randao** contract selects a random keeper for this job. However, the job owner contract can revert the transaction if the selected keeper does not satisfy a desirable requirement. As a result, with some extra effort, the job owner can select the keeper.

This mechanism is not a thread by itself but can be used in other attack vectors.

This analysis was performed by Pessimistic:

Yhtyyar Sahatov, Junior Security Engineer
Evgeny Marchenko, Senior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

February 22, 2023