PowerPool Agent Security Analysis

by Pessimistic

This report is public

May 24, 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

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.

After the initial audit, the codebase was updated. Most of the issues have been fixed or commented on. However, two new critical issues (Force slashing through reentrancy, Possibility not to pay for the jobs) and four new issues with a medium severity (Missing calldata validation, Resolver calldata is not validated, Problems with randomness result distribution on edge cases, Slashing logic problems) were found.

After the first recheck, the codebase was updated. Most of the critical issues were fixed. One new medium severity (Problems with the resolver job calldata) issue was discovered.

One issue of medium severity (Resolver calldata is not validated, M05) was removed from the report as it was identified as a false positive.

General recommendations

We recommend fixing the remaining issues and simplifying the 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.

Codebase update #1

After the audit, we were provided with commit 7bab843f80cb9690d12396589cbe074a457956a1. In the update, the developers fixed or commented on most of the issues and implemented new tests. However, we discovered two new critical issues (C06, C07) and four new issues with a medium severity (M04, M05, M06, M07).

All 213 tests pass successfully.

Codebase update #2

After the audit, we were provided with commit 09c57bac05bf907a0228f8f7fb15892eee471131. In the update, the developers fixed most of the critical issues and implemented new tests. However, we discovered one new medium severity issue (M08) and one low severity issue (L09).

All 220 tests pass successfully.

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 plugin Slitherin with an extended set of rules and manually verified the output.

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.

On March 31, the developers provided a new version of the code. We reviewed the fixes for the issues and inserted comments from the developers.

Finally, we updated the report.

On May 17, the developers provided a new version of the code. We reviewed the fixes for the issues and updated 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. Repeated slashings might fail (fixed)

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.

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

C02. Job owners can force slashing (fixed)

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.

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

C03. Malicious job can drain all stakes #1 (fixed)

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`.

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

C04. Malicious job can drain all stakes #2 (fixed)

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.

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

C05. Weak randomness (addressed)

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.

> *The developers added the activation delay to make the keepers active. However, one can still try to manipulate the randomness result by preparing lots of active keepers and deactivating them to adjust the randomness result.*

C06. **Force slashing through reentrancy** (fixed)

An attacker can create a malicious job and force slash a victim keeper by reentering the contract.

The job can call `withdrawJobOwnerCredits` or `releaseJob` functions and reliably revert the whole `execute` transaction. This method also effectively prevents `execute` transaction from reaching `_afterExecutionReverted` logic, so the keeper cannot reject this job. Besides, the attacker can utilize a `tx.origin` check to ensure their keeper can slash.

We recommend adding reentrancy locks to prevent jobs from accessing administrative functionality.

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

C07. **Possibility not to pay for the jobs** (fixed)

A malicious user can create a resolver-type job that never pays for execution.

1. This job should revert all direct execution attempts. It can revert if `tx.origin` is the assigned keeper to achieve this behavior.
2. Some address might initiate slashing to utilize this slashing opportunity. However, `initiateSlashing` function calls the resolver internally, which can execute the actual task. The resolver can check if `msg.sender` is the Agent contract to ensure it is currently within `initiateSlashing` transaction
3. By this step, the attacker has their task executed, and the job contract may revert all slashing attempts to avoid slashing fees.

Consider simulating the resolver call via `checkCouldBeExecuted` when it is interacted via `initiateSlashing`, so it cannot have any side effects.

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

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 (fixed)

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.

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

M02. Insufficient minimal job reward (fixed)

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.

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

M03. Front-running attack (commented)

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.

> *Comment from the developers: We will instruct keepers to deposit ETH on a worker's address only after it has been created.*

M04. Missing calldata validation (addressed)

In the `initiateSlashing` of **PPAgentV2Randao** contract, the `jobCalldata_` is not validated, and the keeper can provide any calldata.

> *The developers added the check of the selector. However, for the jobs with the predefined calldata, it is required to check the whole provided calldata (or take the calldata from the storage). Otherwise, the malicious keeper can pass different calldata and manipulate the result of the call.*

M06. Problems with randomness result distribution on edge cases

In the `_assignNextKeeper` function of the **PPAgentV2Randao** contract, the keeper ordering might have a great effect on randomness result distribution. For instance, if there are only two keepers with stake `stake>=x`. And the two keepers are on the positions `y` and `y+1`. The job with the stake requirement `x` will be assigned to the keeper on the position `y` with the probability `n-1/n`, where `n` is the number of active keepers.

M07. Lazy keeper attack

A malicious keeper can listen to `initiateSlashing` events instead of monitoring assigned resolver-type jobs and execute tasks after other keepers initiate slashing.

In this scenario, the lazy keeper gets the rewards, and the slashers just pay for gas. This can discourage the keepers from initiating slashing. Besides, job owners are affected, as this attack creates additional delays before the job execution. Moreover, some tasks may not be executed at all.

M08. Problems with the resolver job calldata (new)

It is assumed that the keeper will pass calldata to the job from the result of the job resolver call. However, this assumption can lead to the following problems:

1. It is required to validate calldata in the job contract since, in `execute_44g58pv`, only the selector is checked. Also, note that the keeper might be able to submit a subset of the calldata (if the function is expecting an array) and pass the validation. As a result, the user's job might be executed as not desired.

2. It is also required to validate that the keeper is not passing additional data in the calldata. Note that Solidity does not check if there is any data except the expected one. The keeper can pass additional random data along with valid calldata, and force the user to pay for an enormous amount of gas.

3. A malicious keeper can pass invalid data after the slashing is initiated and move the task to the inactive state because the `_afterExecutionReverted` function will be called.

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 (fixed)

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

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

L02. Gas consumption (fixed)

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.

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

L03. Gas consumption (fixed)

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

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

L04. Memory-safe assembly (fixed)

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.

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

L05. Missing check (fixed)

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`.

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

L06. Deprecated property (commented)

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

> *Comment from the developers: We would like to stick to Solidity v0.8.17 for now, which supports only the **block.difficulty**. We will update this line if we decide to upgrade to the newer compiler.*

L07. Keeper assignment issues (commented)

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.

> *Comment from the developers: Now we allow a job owner to release a keeper without any restriction by calling `releaseJob(jobKey)`.*

L08. Complicated code structure (commented)

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.

> *Comment from the developers: We would like to keep **PPAgentV2** as a base contract to inherit from. We would add other contracts like **PPAgentV2Randao** in the future. In this codebase **PPAgentV2** is not intended to be deployed.*

L09. Code logic (new)

The expression `totalSlashAmount = eKeeper.cvpStake;` at line 514, in **PPAgentV2Randao** contract, should be moved to the line after 517 to correctly calculate the `slashAmountMissing` variable. Otherwise, this variable will always be equal to zero.

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 threat by itself but can be used in other attack vectors.

N02. Exploitable randomness #2

Since the index of the keeper to whom the job will be assigned is calculated as `index = ((pseudoRandom + uint256(jobKey_)) % totalActiveKeepers)`, one can assign the job to a desired keeper by creating multiple inactive jobs and activate the job only if the assigned keeper is the wanted one.

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

N03. Job owner compensation

We think that the job owner also needs to get some compensation from the slashing since they are also a victim in this situation (their task are executed with delay, etc.).

N04. Slashing rewards might encourage the keepers for the attacks

High slashing reward (which depends on the stake of the assigned keeper) can encourage keepers to attack for this reward.

This analysis was performed by Pessimistic:

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

May 24, 2023