# SMART CONTRACTS REVIEW

zokyo

March 5th 2024 | v. 1.0

# Security Audit Score

## PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

# # ZOKYO AUDIT SCORING REPL

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

# HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 0 Medium issues: 0 points deducted
- 2 Low issues: 2 acknowledged issues = - 3 points deducted
- 2 Informational issues: 2 resolved issues = 0 points deducted

Thus, 100 - 3 = 97

# TECHNICAL SUMMARY

This document outlines the overall security of the Repl smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Repl smart contract/s codebase for quality, security, and correctness.

## Contract Status


There were 0 critical issues found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Repl team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

# Table of Contents

# AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Repl repository:
Repo: https://github.com/Project-pFIL/pFIL-contracts/
The last fix - PR: https://github.com/Project-pFIL/pFIL-contracts/pull/92
Last commit: 8f0b5ecb58315132df11d62422c45c8ab05b88a8

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- contracts/AgentImplementation.sol
- contracts/Repl.sol
- contracts/ReplAuction.sol
- contracts/interfaces/Interface.sol

**During the audit, Zokyo Security ensured that the contract:**

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Repl smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01 Due diligence in assessing the overall code quality of the codebase.

02 Cross-comparison with other, similar smart contract/s by industry leaders.

03 Thorough manual review of the codebase line by line.

# Executive Summary

The Zokyo team has performed a security audit of the provided codebase. The contracts submitted for auditing are well-crafted and organized. Detailed findings from the audit process are outlined in the "Complete Analysis" section.

# STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Repl team and the Repl team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

**Critical**

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

**High**

The issue affects the ability of the contract to compile or operate in a significant way.

**Medium**

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

**Low**

The issue has minimal impact on the contract’s ability to operate.

**Informational**

The issue has no impact on the contract’s ability to operate.

# COMPLETE ANALYSIS

## FINDINGS SUMMARY

| # | Title | Risk | Status |
|---|---|---|---|
| 1 | Use of single step ownership transfer | Low | Acknowledged |
| 2 | Lack of return validation in ERC20 transfer | Low | Acknowledged |
| 3 | Redundant condition in withdrawal validation logic | Informational | Resolved |
| 4 | Potential gas inefficiency due to unneeded public accessibility | Informational | Resolved |

**Use of single step ownership transfer**

The multiple contracts in the protocol use the OwnableUpgradeable contract which allows changing the owner address. However, this contract does not implement a 2-step-process for transferring ownership. If the admin's address is set incorrectly, this could potentially result in critical functionalities becoming locked.

**Recommendation:**
Consider implementing a two-step pattern. Utilize OpenZeppelin's Ownable2StepUpgradeable contract.

**Lack of Return Value Validation in ERC20 Transfer**

Location: AgentImplementation.sol
The smart contract initiates ERC20 token transfers without validating the return value of the transfer operation. Failing to check the return value may result in unhandled failures, exposing the contract to potential vulnerabilities and financial risks. The absence of return value validation may lead to unhandled transfer failures, causing unintended consequences in contract state and potentially resulting in financial losses.

This is taking place in 2 occurrences:

In function `reclaimOwnerAddress` -

```
if (_pFILBalance > 0) {
    IPFIL(pFIL).transfer(owner, _pFILBalance);
}
```

Also in function `paybackPFIL` -

```
IPFIL(pFIL).transferFrom(msg.sender, address(this), _receive);
```

**Recommendation:**
Validate Return Values: Implement validation checks on the return value of ERC20 token transfers to handle success or failure appropriately.

**Redundant condition in withdrawal validation logic**

The condition within the if statement in the agentWithdrawFromMiner function contains redundancy. Specifically, the two conditions are attempting to ensure that after the withdrawal, there is still enough balance left to meet the reserved balance requirement.

```
if (_total < getReservedBalance() || _total < getReservedBalance() + amount)
```

However, the second condition implicitly covers the first. If _total is less than getReservedBalance() + amount, it is inherently less than or equal to getReservedBalance() alone since amount is assumed to be a positive number. Therefore, the first condition (_total < getReservedBalance()) is unnecessary because if the second condition is true, the first is automatically true as well.

**Recommendation:**
Simplify conditional checks. Revise the conditional statement to eliminate redundancy, ensuring clarity and efficiency in logic. The recommended condition would be:

```
if (_total < getReservedBalance() + amount)
```

**Potential Gas Inefficiency due to unneeded public accessibility**

The smart contracts declare certain functions as public, even though they are not invoked from within the contract (i.e. internally). This could lead to potential gas inefficiency, as external function calls may offer more gas-efficient execution compared to public functions.

```
// ReplAuction.sol
function getRemainingFILForAuction() public view returns (uint)
function auctionIsExpired(address _agent) public view returns (bool)

// AgentImplementation.sol
function calculateSafePledge() public onlyOwner
```

**Recommendation:**
To optimize gas usage and promote efficiency, it is recommended to adjust visibility levels by changing the visibility of functions that are not utilized internally to external, considering potential gas savings.

| | contracts/AgentImplementation.sol<br>contracts/Repl.sol<br>contracts/ReplAuction.sol<br>contracts/interfaces/Interface.sol |
|---|---|
| Re-entrancy | Pass |
| Access Management Hierarchy | Pass |
| Arithmetic Over/Under Flows | Pass |
| Unexpected Ether | Pass |
| Delegatecall | Pass |
| Default Public Visibility | Pass |
| Hidden Malicious Code | Pass |
| Entropy Illusion (Lack of Randomness) | Pass |
| External Contract Referencing | Pass |
| Short Address/ Parameter Attack | Pass |
| Unchecked CALL<br>Return Values | Pass |
| Race Conditions / Front Running | Pass |
| General Denial Of Service (DOS) | Pass |
| Uninitialized Storage Pointers | Pass |
| Floating Points and Precision | Pass |
| Tx.Origin Authentication | Pass |
| Signatures Replay | Pass |
| Pool Asset Security (backdoors in the underlying ERC-20) | Pass |

We are grateful for the opportunity to work with the Repl team.

**The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.**

Zokyo Security recommends the Repl team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo