SMART CONTRACTS REVIEW

zokyo

December 8th 2023 | 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 issue: 0 points deducted
- 1 High issue: 1 resolved and = 0 points deducted
- 2 Medium issues: 2 resolved = 0 points deducted
- 2 Low issues: = 2 resolved = 0 points deducted
- 6 Informational issues: 6 resolved = 0 points deducted

Thus, score is 100

TECHNICAL SUMMARY

This document outlines the overall security of the Repl smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Repl smart contracts 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 contracts 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 contracts.

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

Last commit - aee2b6fa869a1ed256772954ccab054154bf58ec

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

- ./wPFIL.sol
- ./AgentProxy.sol
- ./ReplAuction.sol
- ./PFIL.sol
- ./AgentImplementation.sol
- ./Repl.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 resent 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 contracts. 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 contracts by industry leaders.

03 Thorough manual review of the codebase line by line.

Executive Summary

The Zokyo team detected vulnerabilities of varying severity levels, including high, medium, and low, as well as a few informational issues. It's important to note that the Repl team promptly reacted to all identified issues. For a more comprehensive breakdown of these findings, please refer to 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	An attacker can cause legitimate transactions to revert and purchase auctioned tokens at the lowest possible price	High	Resolved
2	The pFILAmount Should Be Used Instead Of _amount	Medium	Resolved
3	The subtraction in the receiveWithdraw function may cause excessive reverts due to arithmetic underflow	Medium	Resolved
4	Lack of protection against initialization of implementation contracts	Low	Resolved
5	Unnecessary usage of library	Low	Resolved
6	Use of floating pragma	Informational	Resolved
7	Misleading mapping name	Informational	Resolved
8	Unneeded costly Safe Math computation	Informational	Unresolved
9	Misleading documentation comments	Informational	Resolved
10	CEI Violation	Informational	Resolved
11	No Need To Check amount < 0	Informational	Resolved
12	Length Of The Array Should Be Cached Outside The For Loop	Informational	Resolved

HIGH-1 | RESOLVED

An attacker can cause legitimate transactions to revert and purchase auctioned tokens at the lowest possible price

The buy function in ReplAuction contract allows users to purchase FIL with pFIL tokens. The function calculates the FIL token price dynamically, with the price decreasing over time to benefit auction participants. An attacker can take advantage of this by monitoring the mempool for pending transactions that attempt to purchase a large number of FIL tokens. Before the legitimate transaction is confirmed, the attacker can execute a front-running attack causing this transaction to revert.

Let's assume the following scenario:
When a legitimate user attempts to purchase the maximum amount of FIL available in an auction, an attacker can observe this transaction and front-run it by submitting their own transaction to buy a trivial amount of FIL (as little as 1 wei).
Once the attacker's transaction is executed, the maxAmount of FIL available for auction decreases. As a result, when the legitimate user's transaction is processed, it fails the check:

```
        require(_amount <= maxAmount, "amount > maxAmount");
```

because the maxAmount has been reduced by the attacker's purchase. This leads to the legitimate transaction being reverted, causing the user to lose out on the opportunity to buy FIL at the intended price and wasting their transaction fees. By doing so, the attacker can continuously front-run legitimate transactions and purchase FIL tokens at the lowest possible price.

Recommendation:
Implement a minimum transaction threshold to prevent trivial purchase from impacting the greater orders.

The pFILAmount Should Be Used Instead Of _amount

Users can buy FIL with pFIL via the buy() function in the ReplAuction.sol contract. At L159 it makes a check *require(pFILToken.balanceOf(msg.sender) >= _amount, "pFIL balance < price");* . This check is trying to make sure that the user has enough pFIL balance (making sure it is more than or equal to the amount being sent) , but it compares pFIL balance of the user with FIL amount instead of pFIL amount , the correct check would be *require(pFILToken.balanceOf(msg.sender) >= pfilAmount, "pFIL balance < price");* since we want to make sure the user has enough pFIL not FIL.
This might make valid transactions revert .

Recommendation:
Change the statement as stated.

LOW-1 | RESOLVED

Lack of protection against initialization of implementation contracts

The audited contracts use the Initializable module. To prevent leaving an implementation contract uninitialized, OpenZeppelin's documentation recommends adding the _disableInitializers function in the constructor, which automatically locks the contracts when they are deployed.

Recommendation:
Consider adding an empty constructor that calls _disableInitializers().

INFORMATIONAL-1 | RESOLVED

Use of floating pragma

The audited contracts use the following floating pragma:

```
agma solidity ^0.8.17;
```

It allows for the compilation of contracts with various compiler versions and introduces the risk of deploying with a different version than the one used during testing.

Recommendation:
Use a specific version of the Solidity compiler.

INFORMATIONAL-2 | RESOLVED

Misleading mapping name

Repl.sol - The mapping `ownerToAgentMap` is referring to owners given an agent address. The mapping name can cause confusion as it implies that it refers to agent address given address of owner.

```
ownerToAgentMap[address(agent)] = msg.sender;
```

Recommendation:
Rename mapping and refactor the code accordingly.

Unneeded costly Safe Math computation

PFIL.sol - In function `decreaseAllowance()` the operation `currentAllowance.sub(_subtractedValue)` can be executed without the overhead of SafeMath. This is because it is already required and asserted that `currentAllowance` is greater in value than or equal to `_subtractedValue`.

```
    function decreaseAllowance(
        address _spender,
        uint256 _subtractedValue
    ) public override returns (bool) {
        uint256 currentAllowance = allowances[msg.sender][_spender];
        require(currentAllowance >= _subtractedValue,
"ALLOWANCE_BELOW_ZERO");
        _approve(msg.sender, _spender,
currentAllowance.sub(_subtractedValue));
        return true;
    }
```

Similarly in `_burnShares` for operation `_getTotalShares().sub(_sharesAmount)`. Since `_sharesAmount <= accountShares` and `accountShares` is less than or at most equal to the total shares of the token.

```
  function _burnShares(address _account, uint256 _sharesAmount) internal
returns (uint256) {
        require(_account != address(0), "BURN_FROM_ZERO_ADDR");

        uint256 accountShares = shares[_account];
        require(_sharesAmount <= accountShares, "BALANCE_EXCEEDED");

        uint256 preRebaseTokenAmount = getFILByShares(_sharesAmount);

        totalShares = _getTotalShares().sub(_sharesAmount);

        shares[_account] = accountShares.sub(_sharesAmount);

        uint256 postRebaseTokenAmount = getFILByShares(_sharesAmount);
```

```
        emit SharesBurnt(_account, preRebaseTokenAmount,
postRebaseTokenAmount, _sharesAmount);

        return totalShares;

    }
```

Recommendation:
Operations can be marked unchecked to save the overhead of library SafeMath and the built-in solidity safe maths.

Fix: Client no longer uses the SafeMath Library. Still no response regarding this specific issue because the unnecessary safe math computation by default is being carried out in some cases where it is not needed.

INFORMATIONAL-4 | RESOLVED

Misleading documentation comments

AgentImplementation.sol - Comment for function `_getOwnerReturn` refers mistakenly to beneficiary address.

```
   /**
     * @dev get beneficiary address of current miner
     */
    function _getOwnerReturn()
        internal
        view
        virtual
        returns (MinerTypes.GetOwnerReturn memory ownerReturn)
    {
        CommonTypes.FilActorId _actorId =
CommonTypes.FilActorId.wrap(actorID);
        ownerReturn = MinerAPI.getOwner(_actorId);
    }
```

Recommendation:
A minor correction to the comment required.

INFORMATIONAL-5 | RESOLVED

CEI Violation

Checks Effects Interaction pattern is being violated in the auctionBidded() function in Repl.sol , there is an external call to the user at L320 which is done before the state changes at L326,327,328.

Recommendation:
The external call should be in the end after all the state changes.

INFORMATIONAL-6 | RESOLVED

No Need To Check amount < 0

The check at L151 in AgentImplementation.sol can be reduced to **if (amount >= ownerActor.balance) revert InvalidSafePledge();** since amount is of type uint256 which can not be negative.

Recommendation:
Remove the extra check

INFORMATIONAL-7 | RESOLVED

Length Of The Array Should Be Cached Outside The For Loop

The for loop at L167 in AgentImplementation.sol computes the length of the controlAddresses array with each iteration , this costs 100 extra gas each time. Instead the length of the array can be cached outside the for loop to save on gas.

Recommendation:
Cache the length of the controlAddresses array outside the for loop.

	./wPFIL.sol ./AgentProxy.sol ./ReplAuction.sol ./PFIL.sol ./AgentImplementation.sol ./Repl.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 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