SMART CONTRACT AUDIT

zokyo

June 5th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

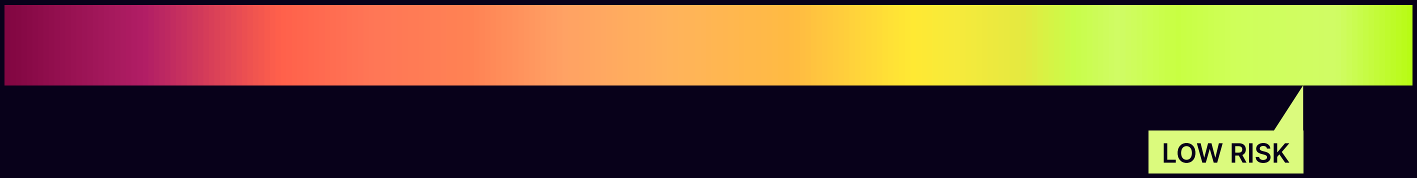
SCORE
95

TECHNICAL SUMMARY

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

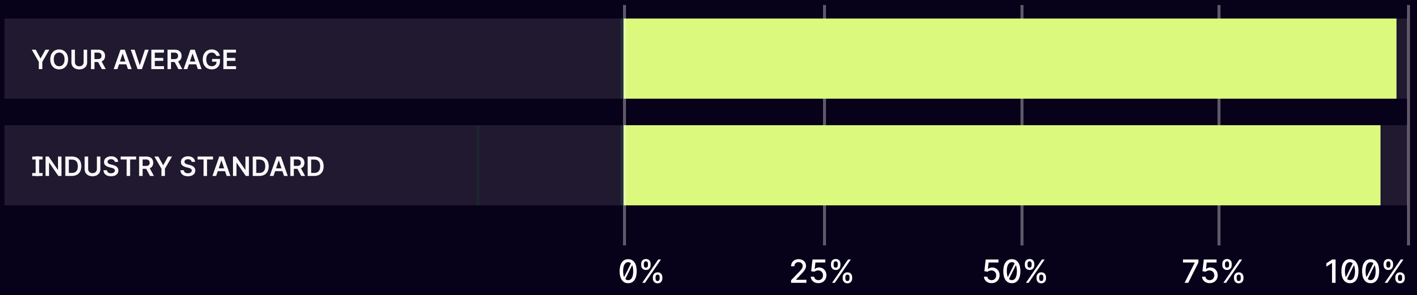
The scope of this audit was to analyze and document the Fluxfire smart contracts codebase for quality, security, and correctness.

Contract Status



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

Testable Code



100% of the code is testable, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract but rather limited to an assessment of the logic and implementation. In order to ensure a secure contracts that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Fluxfire 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 3
Executive Summary 4
Structure and Organization of the Document 5
Complete Analysis 6
Code Coverage and Test Results for all files written by Zokyo Security 12

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Fluxfire repository:
https://github.com/fluxfirexyz/IDOFCFS

ea8cc2b8a758f78c7d0652dab25f14cfa3a963e5

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

- ito.sol
- IQLF.sol
- qualification.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 Fluxfire 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. Part of this work includes writing a test suite using the Hardhat testing framework. 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 Testing contracts logic against common and uncommon attack vectors.

04 Thorough manual review of the codebase line by line.

Executive Summary

The audit revealed no critical findings. Our team diligently investigated and found one medium issue, two low issues, and two informational findings, all of which are extensively described 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 Fluxfire team and the Fluxfire team is aware of it, but they have chosen to not solved 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	Function destruct in HappyTokenPool contract does not reset pool's Packed3 struct fields and also does not set pool's claimed variable to true.	Medium	Acknowledged
2	Function claim() in HappyTokenPool, does not update pool's pool.swapped_map[msg.sender].	Low	Acknowledged
3	Function initialize() in HappyTokenPool should be restricted to contract owner or creator.	Low	Acknowledged
4	Function swap() sends ether, it is recommended that you add openzeppelin's reentrancy guard to prevent any possible reentrancy attacks.	Informational	Acknowledged
5	The code line `msg.sender == pool.creator` repeated in different functions in HappyTokenPool contract should be abstracted into a modifier to save gas and also make it easily reusable.	Informational	Unresolved

MEDIUM | ACKNOWLEDGED

Function destruct in HappyTokenPool contract does not reset pool's Packed3 struct fields and also does not set pool's claimed variable to true.

Description: This allows a user to be able to call the claim function even when the pool is destructed. The claim function validates fields of Packed3 struct, failure to update the fields of the struct together with the pool's SwapStatus struct's claimed field upon function destruct call leaves users to still be able to call claimed function.

Recommendation:

To fix this problem, the pool's packed3 fields should be updated accordingly as part of this function's call. issue 1: Since

Comment: Since client says it's by design which I also thought as much but needed to mention just in case it's not intentional.

LOW | ACKNOWLEDGED

Function claim() in HappyTokenPool, does not update pool's pool.swapped_map[msg.sender].

Description: The function does not update the pool.swapped_map[msg.sender] value after getting it value at "`uint256 claimed_amount = pool.swapped_map[msg.sender]`" Even though the claimed status is updated thus "`pool.swap_status[msg.sender].claimed = true;`" which prevents further claims but it is also advisable to reset the swapped_map value after claim.

Recommendation:

Update the variable after the claim is successful.

```
Pool.swapped_map[msg.sender] = 0;
```

Comment: Yes, resetting only swap_status maps would save gas instead of resetting both maps but it was worth mentioning because the amount was already sent to the caller so resetting it to zero would be a good practice not necessarily a security issue.

LOW | ACKNOWLEDGED

Function initialize() in HappyTokenPool should be restricted to contract owner or creator.

Description: The function has no restriction as to who can call it. Because this function sets up some key information, this can lead to unexpected values set if it is not restricted to the owner or creator.

Recommendation:
Since this function initializes some important contract information, it is recommended that it is restricted to only the contract creator.

Comment: Since the contract would be deployed using the proxy pattern this issue is not an issue.

INFORMATIONAL | ACKNOWLEDGED

Function swap() sends ether, it is recommended that you add openzeppelin's reentrancy guard to prevent any possible reentrancy attacks.

Recommendation:
Add openzeppelin reentrancy guard library to this function.

INFORMATIONAL | UNRESOLVED

The code line `msg.sender == pool.creator` repeated in different functions in HappyTokenPool contract should be abstracted into a modifier to save gas and also make it easily reusable.

Description: The line of code stated above could be moved into a custom modifier, to make it reusable and then also save gas instead of implementing it in the body of the function.

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

	qualification.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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting Fluxfire in verifying the correctness of their contract/s code, our team was responsible for writing integration tests using the Hardhat testing framework.

The tests were based on the functionality of the code, as well as a review of the Fluxfire contract/s requirements for details about issuance amounts and how the system handles these.

```
HappyTokenPool
    Initialization
    ✓ fill_pool
    ✓ Should fill pool (118ms)
    ✓ Should revert with Start time should be earlier than end time. (70ms)
    ✓ Should revert with End time should be earlier than unlock time (42ms)
    ✓ Should revert Limit needs to be less than or equal to the total supply (39ms)
    ✓ Should revert with Exchange token addresses need to be set (38ms)
    ✓ Should revert with Size of ratios = 2 * size of exchange_addrs (56ms)
    swap()
    ✓ Should swap swap (108ms)
    ✓ Should revert with Already swapped (112ms)
    ✓ Should revert with already swapped (88ms)
    ✓ Should swap successfully swap (109ms)
    ✓ Should revert with expired (83ms)
    ✓ Should swap with packed3.start_time + base_time < block.timestamp (111ms)
    ✓ Should revert with no enough ether (85ms)
    ✓ Should swap if swapped_tokens > packed2.total_tokens (94ms)
    ✓ Should revert with Wrong Password (72ms)
    ✓ Should revert with Out of Stock (171ms)
    ✓ Should  revert with Not started. (68ms)
    ✓ Should revert with Expired. (74ms)
    setUnlockTime
    ✓ setUnlockTime (39ms)
    ✓ Cannot set to 0 (51ms)
    ✓ Pool Creator Only (50ms)
    ✓ Too Late (39ms)
    ✓ Not eligible when unlock_time is 0 (40ms)
```

```
        destruct
        ✓ Should destruct (99ms)
        ✓ Only the pool creator can destruct. (61ms)
        ✓ SHould destruct (99ms)
        ✓ Should destrcuted (91ms)
        claim
        ✓ Should claim (74ms)
        ✓ (39ms)
        ✓ (53ms)
        ✓ (189ms)
        ✓ (63ms)
        check_availability
        ✓ Should check pool's check_availability (68ms)
    QLF
        Initialization
        ✓ set_start_time (38ms)
        ✓ set_start_time (39ms)
        ✓ ifQualified
        ✓ logQualified (65ms)
        ✓ logQualified (80ms)
        ✓ logQualified (63ms)
        ✓ supportsInterface (65ms)

        41 passing (15s)
```

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% FUNCS	UNCOVERED LINES
	100	60	100	100	
Ito.sol	94.95	79.07	100	93.28	
All Files	**100**	**100**	**100**	**100**	

We are grateful for the opportunity to work with the Fluxfire 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 Fluxfire team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.