SMART CONTRACT AUDIT

ZOKYO.

September 7th, 2021 | v. 2.0

PASS

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

SCORE

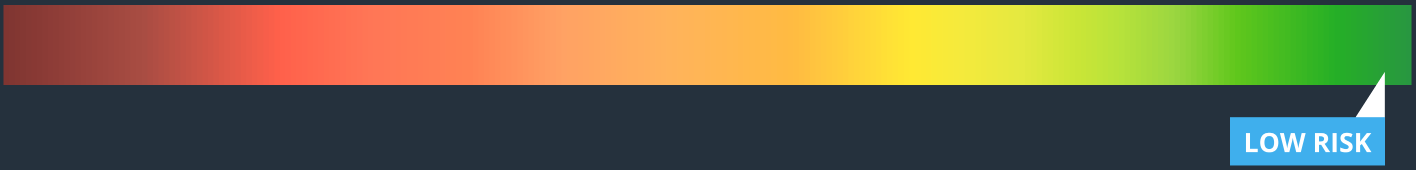
99

TECHNICAL SUMMARY

This document outlines the overall security of the Saffron smart contracts, evaluated by Zokyo's Blockchain Security team.

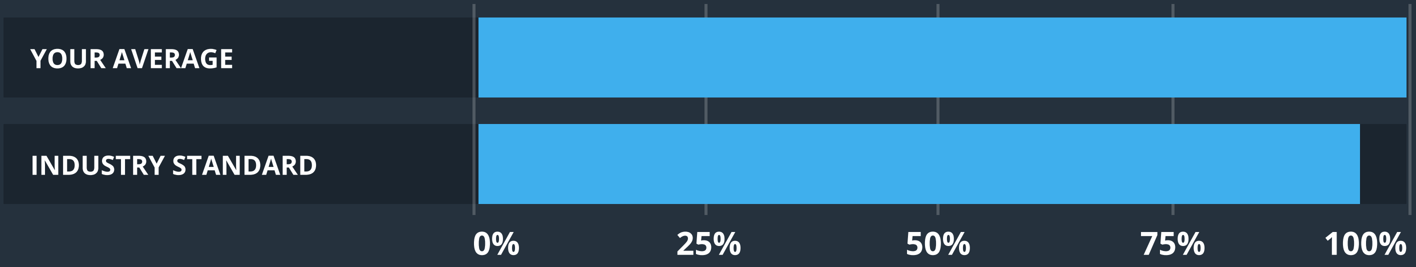
The scope of this audit was to analyze and document the Saffron smart contract codebase for quality, security, and correctness.

Contract Status



There were no critical issues found during the audit.

Testable Code



The testable code is 100%, 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, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Saffron team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Saffron repository.

Repository:
https://github.com/saffron-finance/saffron-staking-v2/commit/808bccd29c175b036dc141148bfe5eb4f3bfad28

Last commit:
808bccd29c175b036dc141148bfe5eb4f3bfad28

Contracts:

- MockERC20.sol
- SFIRewarder.sol
- SaffronStakingV2.sol

Throughout the review process, care was taken to ensure that the token contract:

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Saffron smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1. Due diligence in assessing the overall code quality of the codebase.
2. Cross-comparison with other, similar smart contracts by industry leaders.
3. Testing contract logic against common and uncommon attack vectors.
4. Thorough, manual review of the codebase, line-by-line.

SUMMARY

Zokyo conducted a security audit for the following contracts: MockERC20.sol, SFIRewarder.sol, SaffronStakingV2.sol

The given contracts are in good condition. During the auditing process, Zokyo auditors haven’t found any issues with critical or high severity levels. All of the findings that had operational or security defects have been successfully fixed by the Saffron team.

Based on the given results, Zokyo auditors can state that the contracts are secure and bear no risk for the contact owner or end-user. The overall score of the contracts is set for 99.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. 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 ability of the contract to compile or operate in a significant way.

● 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

The contract owner has too many permissions

MEDIUM | RESOLVED

The rewardCutoff functionally allows the owner to change the last block number when the user can farm rewards. It means that the owner can add a new pool, wait for investors to invest and then change rewardCutoff to current block + 1. It may have unexpected behavior for potential investors.

Recommendation:
Make changes of rewardCutoff in a way when the owner can’t set less then initial value.

Re-audit:
They will use multisig for the owner, so the owner can’t remove all rewards by himself.

Function call may run out of gas

MEDIUM | RESOLVED

The mass update pool allows the contract to update all pools data in one call. It can be the cause of run out of gas if poolInfo.length will be a big number.

Recommendation:
There are few possible solutions:

- Make a comment to explain that poolInfo.length won't be a big number;
- Make additional requirements in the add function;
- Use updatePool with pool index instead of massUpdatePools.

Re-audit:
Issue is resolved by using updatePool with pool id.

Withdraw function can be called with 0 value

INFORMATIONAL | NOT VALID

Withdraw function can be called with 0 amount of token to withdraw, and function doesn’t have equirement:

```
require(amount > 0, "can't withdraw 0 tokens");
```

It causes that function to do more work than it needs.

Recommendation:
Add requirement in withdraw function.

	MockERC20	SFIRewarder	SaffronStakingV2
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Saffron team

Code Coverage

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

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
contracts\	100.00	91.68	100.00	98.80	
MockERC20.sol	100.00	100.00	100.00	100.00	
SFIRewarder.sol	100.00	50.00	100.00	88.89	74
SaffronStakingV2.sol	100.00	100.00	100.00	100.00	
contracts\interface\	100.00	100.00	100.00	100.00	
ISFIRewarder.sol	100.00	100.00	100.00	100.00	
All files	**100.00**	**91.18**	**100.00**	**98.80**	

Test Results

```
SaffronStakingV2 contract test
  ✓ should transfer to rewarder correctly 115ms)
  ✓ should set the staking address (215ms)
  ✓ should set rewardCutoff correctly (284ms)
  ✓ should set rewardCutoff and sfiPerBlock correctly (261ms)
  ✓ should add pools correctly (210ms)
  ✓ should deposit correctly (535ms)
  ✓ should revert when attempting to withdraw more than user owns (155ms)
  ✓ should withdraw 0 to claim the rewards before reward period ends (3813ms)
  ✓ should withdraw 50% of principal to claim the rewards before reward period ends (3437ms)
  ✓ should withdraw remaining principal to claim the rewards after the reward period ends (4989ms)
  ✓ shouldn't have any rewards after reward end block (6958ms)
```

```
SaffronStakingV2 function coverage
  ✓ should deploy SFIRewarder contract correctly (144ms)
  ✓ should deploy SaffronStakingV2 contract correctly (266ms)
  ✓ should set SFIRewarder staking address correctly (78ms)
  ✓ should have SFIRewarder rewardUser work correctly (139ms)
  ✓ should have SFIRewarder emergencyWithdraw work correctly (94ms)
  ✓ should have SaffronStakingV2 setRewarder work correctly (64ms)
  ✓ should have SaffronStakingV2 setRewardPerBlock work correctly (47ms)
  ✓ should have SaffronStakingV2 setRewardCutoff work correctly (79ms)
  ✓ should have SaffronStakingV2 add pool work correctly (145ms)
  ✓ should have SaffronStakingV2 add pool withUpdate work correctly (104ms)
  ✓ should have SaffronStakingV2 poolLength work correctly (59ms)
  ✓ should have SaffronStakingV2 set pool work correctly (96ms)
  ✓ should have SaffronStakingV2 pendingSFI correctly (127ms)
  ✓ should have SaffronStakingV2 massUpdatePools work correctly (212ms)
  ✓ should have SaffronStakingV2 updatePool work correctly (128ms)
  ✓ should have SaffronStakingV2 emergencyWithdraw work correctly (164ms)

SaffronStakingV2 deposits/withdrawals
  ✓ users that deposit first should have more rewards and total rewards match
    rewarder total (7157ms)
  ✓ should have users multiple deposits and withdrawals not cause math errors (39476ms)
  ✓ should have users alternate multiple deposits and withdrawals not cause math errors (44592ms)
  ✓ should have users deposits spanning reward cutoff block not lose funds (6201ms)
  ✓ should have users deposits for same amount of blocks get same reward
    if deposits don't overlap' (916ms)

SaffronStakingV2 multiple transactions per block
  ✓ should have user deposit and withdrawing in same block not get rewards (239ms)

multiple consecutive deposit/withdraw tests
  ✓ should have random users depositing and withdrawing randomly return
    correct amounts (234787ms)
  ✓ should have random users depositing and withdrawing randomly return correct amounts
    correct amounts (277750ms)

35 passing (11m)
```

Tests written by Zokyo Security team

As part of our work assisting Saffron in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the Saffron contract requirements for details about issuance amounts and how the system handles these.

Code Coverage

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

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
contracts\	100.00	97.06	100.00	100.00	
MockERC20.sol	100.00	100.00	100.00	100.00	
SFIRewarder.sol	100.00	100.00	100.00	100.00	
SaffronStakingV2.sol	100.00	96.43	100.00	100.00	
contracts\interface\	100.00	100.00	100.00	100.00	
ISFIRewarder.sol	100.00	100.00	100.00	100.00	
All files	**100.00**	**97.06**	**100.00**	**100.00**	

Test Results

```
Contract: SaffronStakingV2 and SFIRewarder test
  Methods
    SaffronStakingV2 constructor
      ✓ should fail if transfer wrong address of SFIRewarder (285ms)
      ✓ should fail if transfer wrong block number as cutOff (929ms)
    SFIRewarder constructor
      ✓ should fail if transfer zero address as address of reward token (278ms)
      ✓ should work correctly (194ms)
```

```
SFIRewarder setStakingAddress
  ✓ should fail if caller isn't owner (175ms)
  ✓ should fail if transfer zero address as address of staking (289ms)
SFIRewarder rewardUser
  ✓ should fail if caller isn't staking (221ms)
SFIRewarder emergencyWithdraw
  ✓ should fail if caller isn't owner (207ms)
  ✓ should fail if transfer zero address as address of user (289ms)
  ✓ should fail if transfer zero address as address of reward token (303ms)
  ✓ should work correctly (696ms)
SaffronStakingV2 setRewarder
  ✓ should fail if transfer zero address (213ms)
  ✓ should fail if caller isn't owner (303ms)
  ✓ should work correclty (381ms)
SaffronStakingV2 setRewardPerBlock
  ✓ should fail if caller isn't owner (240ms)
  ✓ should work correctly (1070ms)
SaffronStakingV2 setRewardCutoff
  ✓ should fail if caller isn't owner (255ms)
  ✓ should fail if transfer wrong block number (405ms)
  ✓ should work correctly (353ms)
SaffronStakingV2 setRewardPerBlockAndRewardCutoff
  ✓ should fail if caller isn't owner (172ms)
  ✓ should fail if transfer wrong rewardCutoff value (234ms)
  ✓ should work correctly (1167ms)
SaffronStakingV2 add
  ✓ should fail if caller isn't owner (207ms)
  ✓ should fail if transfer zero address as rewardToken address (209ms)
  ✓ should fail if call this function after cutOff (1116ms)
  ✓ should work correctly (1572ms)
SaffronStakingV2 set
  ✓ should fail if caller isn't owner (267ms)
  ✓ should fail if transfer wrong pool id (209ms)
  ✓ should fail if transfer max value of uint256 (1116ms)
  ✓ should work correctly (1572ms)
SaffronStakingV2 deposit
  ✓ should fail if transfer wrong pool id (295ms)
  ✓ should fail if user not approved (933ms)
```

```
      ✓ should fail if user doesn't have enough tokens on his balance (906ms)
      ✓ should work correctly (2695ms)
    SaffronStakingV2 withdraw
      ✓ should fail if transfer wrong pool id (253ms)
      ✓ should fail if user try to withdraw more than he has (1235ms)
      ✓ should fail if transfer negative number as amount for withdraw (892ms)
      ✓ should work correctly (1645ms)
    SaffronStakingV2 emergencyWithdraw
      ✓ should fail if transfer wrong pool Id (256ms)
      ✓ should fail if caller didn't deposit tokens (622ms)
      ✓ should work correctly (1702ms)
    SaffronStakingV2 pendingSFI
      ✓ should fail if transfer wrong pool Id (98ms)
      ✓ should return == 0 if lpSupply == 0 (512ms)
      ✓ should work correctly (1261ms)

  44 passing (35s)
```

We are grateful to have been given the opportunity to work with the Saffron team.

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

Zokyo's Security Team recommends that the Saffron team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.