Swiss
Smart Contract Audit

The BlockChain Auditor

Prepared by: Roger Blackstone

Version: 1
Date: Nov. 13, 2020

Summary of Findings

This document expresses all security concerns of the Swiss Smart Contracts as expressed by Roger Blackstone. I took care to attempt to find as many ways to improve the security, code efficiency, best practices, and overall function of the smart contracts.

Contract Status: Ready for Deployment

0 Critical Issues were found in the farm.sol that need to be resolved.
0 Medium Level Issues were found in farm.sol that need to be resolved.
0 Low Level Issues were found in farm.sol that need to be resolved.
19 Informational Issues were found in farm.sol that should be taken into consideration.

Code Coverage

Swiss	Industry Standard
0.0%	95.0%

This audit should be seen as one step in the development process with the intent of raising awareness on the meticulous work involved in secure development and making no material statements or guarantees to the operational state of the smart contract(s) once they are deployed. This document is not an endorsement of the reliability or effectiveness of the smart contracts. This is an assessment of the smart contract logic, implementation, and best practices. I cannot take responsibility for any potential consequences of the deployment or use of the smart contract(s) related to the audit.

Table of Contents

Table of Contents

Audit Methodology & Techniques

The BlockChain Auditor has the following auditing process:

1. Our audits include
   a. Review of the specifications, source code, and instructions provided to the BlockchainAuditors to clearly identify the desired functionality of the smart contract(s).
   b. Manual line by line review of contract code to spot potential vulnerabilities.
   c. Identification of deviations between desired functionality expressed to the BlockchainAuditors and what the smart contract(s) are doing.
2. Automated static and symbolic analysis, as well as verifying testing coverage using the provided test suite.
   a. Automated static and symbolic analysis help determine what inputs cause each part of the smart contract to execute. Analysis of how much of the code base is tested and comparison to industry standard.
3. Examination of smart contracts and development process as a whole, ensuring best practices are followed, allowing improved efficiency and security based on established industry and academic practices.
4. Specific, itemized, and actionable recommendations to assist in securing the smart contract(s) in question.

Contract Checklist

farm.sol

Contract Vulnerability	
Integer Overflow	Pass
Race Condition	Pass
Denial of Service	Pass
Logical Vulnerability	Pass
Hardcoded Address	Pass
Function Input Parameter Check	Pass
Function Access Control Check	Pass
Random Number Generation	N/A
Random Number Use	N/A
Contract Specification	
Solidity Compiler Version	Pass
Event Use	Pass
Fallback Function Use	Pass
Contructor Use	Pass
Function Visibility Declaration	Fail
Variable Storage Declaration	Fail
Deprecated Keyword Use	Pass
ERC20/223 Standard	Pass
ERC721 Standard	N/A
Business Risk	
Able to Arbitratrily Create Token	N/A
Able to Arbitrarily Destroy Token	N/A
Can Suspend Transactions	N/A
Short Address Attack	Pass
Gas Optimization	
assert()/require()/revert() misused	Pass
Loop Optimization	Pass
Storage Optimization	Fail

Detailed Analysis

5.1 addContractBalance()

farm.sol 380

Optimization

Explanation:

public functions that are never called by the contract should be declared external to save gas.

Recommendation:

Use the external attribute for functions never called from the contract.

5.2 getEstimatedPendingDivs()

farm.sol 412

Optimization

Explanation:

public functions that are never called by the contract should be declared external to save gas.

Recommendation:

Use the external attribute for functions never called from the contract.

5.3 getNumberOfHolders()

farm.sol 427

Optimization

Explanation:

public functions that are never called by the contract should be declared external to save gas.

Recommendation:

Use the external attribute for functions never called from the contract.

5.4 deposit()

farm.sol 432

Optimization

Explanation:

public functions that are never called by the contract should be declared external to save gas.

Recommendation:

Use the external attribute for functions never called from the contract.

5.5 withdraw()

farm.sol 275

Optimization

Explanation:

public functions that are never called by the contract should be declared external to save gas.

Recommendation:

Use the external attribute for functions never called from the contract.

5.6 emergencyWithdraw()

farm.sol 466

Optimization

Explanation:

public functions that are never called by the contract should be declared external to save gas.

Recommendation:

Use the external attribute for functions never called from the contract.

5.7 claim()

farm.sol 486

Optimization

Explanation:

public functions that are never called by the contract should be declared external to save gas.

Recommendation:

Use the external attribute for functions never called from the contract.

5.8 getDepositorsList()

farm.sol 529

Optimization

Explanation:

public functions that are never called by the contract should be declared external to save gas.

Recommendation:

Use the external attribute for functions never called from the contract.

5.9 transferAnyERC20Token() farm.sol 558

Optimization

Explanation:

public functions that are never called by the contract should be declared external to save gas.

Recommendation:

Use the external attribute for functions never called from the contract.

5.10 transferAnyOldERC20Token() farm.sol 567

Optimization

Explanation:

public functions that are never called by the contract should be declared external to save gas.

Recommendation:

Use the external attribute for functions never called from the contract.

5.11 transferOwnership()

farm.sol 309

Optimization

Explanation:

public functions that are never called by the contract should be declared external to save gas.

Recommendation:

Use the external attribute for functions never called from the contract.

5.12 adminCanClaimAfter

farm.sol 346

Optimization

Explanation:

Constant state variables should be declared constant to save gas.

Recommendation:

Add the constant attributes to state variables that never change.

5.13 disburseAmount

farm.sol 340

Optimization

Explanation:

Constant state variables should be declared constant to save gas.

Recommendation:

Add the constant attributes to state variables that never change.

5.14 disburseDuration

farm.sol 342

Optimization

Explanation:

Constant state variables should be declared constant to save gas.

Recommendation:

Add the constant attributes to state variables that never change.

5.15 disbursePercentX100

farm.sol 350

Optimization

Explanation:

Constant state variables should be declared constant to save gas.

Recommendation:

Add the constant attributes to state variables that never change.

5.16 pointMultiplier

farm.sol 378

Optimization

Explanation:

Constant state variables should be declared constant to save gas.

Recommendation:

Add the constant attributes to state variables that never change.

5.17 totalTokensDisbursed

farm.sol 372

Optimization

Explanation:

This variable is not being used anywhere after being initialized to zero so it would be best to be removed. Otherwise, constant state variables should be declared constant to save gas.

Recommendation:

Add the constant attributes to state variables that never change.

5.18 trustedDepositToken-

farm.sol 336

Optimization

Explanation:

Constant state variables should be declared constant to save gas.

Recommendation:

Add the constant attributes to state variables that never change.

5.19 trustedRewardTokenAddress

farm.sol 337

Optimization

Explanation:

Constant state variables should be declared constant to save gas.

Recommendation:

Add the constant attributes to state variables that never change.

Notable Concerns & Suggestions

Overall Thoughts

After a careful examination, I believe this farming contract is ready to be deployed. One initial security concern is the way tokens are transferred to and from trustedDepositTokenAddress and trustedRewardTokenAddress. However, this was found to be acceptable because these addresses are assumed to be under the control of Swiss Finance and as such safe from reentrancy attacks.

Appendix A

File Fingerprints

```
farm.sol                3d8df797b6b11c01cec6025f98849ad8
```

The Blockchain Auditor is honored to have the opportunity to be partnered with Swiss Finance and help provide enhanced security and efficiency for the Swiss Finance platform.

The Swiss Finance Farming protocol was excellently implemented and we are looking forward to seeing what else the team that developed DeCash will develop in the future.

The BlockChain Auditor

- Roger Blackstone