HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: ACryptoS

Date: February 18th, 2021

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities fixed - upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for ACryptoS
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Reward pool
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/acryptos/acryptos-protocol/blob/main/farms/ACryptoSFarmV2.sol
Commit	https://github.com/acryptos/acryptos-protocol/commit/8d68ce017f5644b6cd4cd0aa1157bfce6da0e0b1
Deployed contract	
Timeline	15 FEB 2021 – 18 FEB 2021
Changelog	18 FEB 2021 – INITIAL AUDIT

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by ACryptoS (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer's smart contract and its code review conducted between February 15th, 2021 – February 18th, 2021.

Scope

The scope of the project is smart contracts in the repository:

Contract deployment address:
Repository: https://github.com/acryptos/acryptos-protocol/blob/main/farms/ACryptoSFarmV2.sol
Commit:8d68ce017f5644b6cd4cd0aa1157bfce6da0e0b1
Files: ACryptoSFarmV2.sol

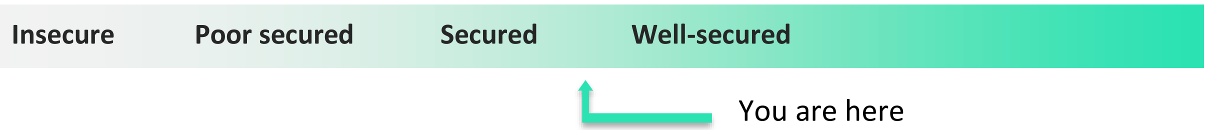
We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency
Functional review	▪ Business Logics Review

	■ Functionality Checks ■ Access Control & Authorization ■ Escrow manipulation ■ Token Supply manipulation ■ Assets integrity ■ User Balances manipulation ■ Data Consistency manipulation ■ Kill-Switch Mechanism ■ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart has some issues that should be fixed.

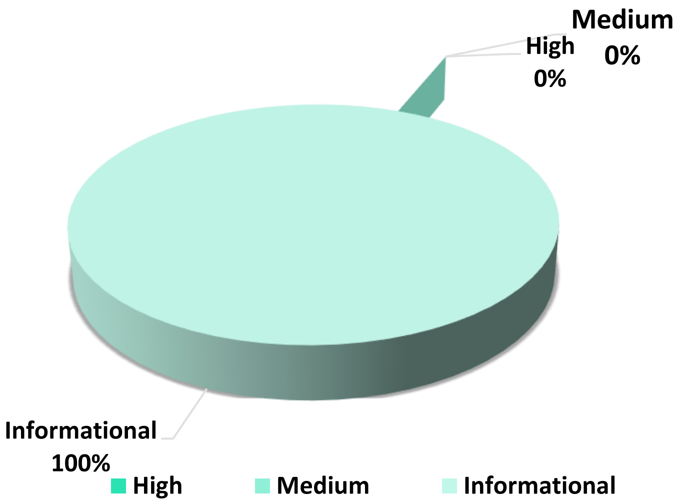


Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section, and all found issues can be found in the Audit overview section.

Security engineers found 2 informational issues during the audit.

Notice: the audit scope contains 1 contract: ACryptoSFarmV2.sol. Resulting score may not be considered as score for the whole project.

Graph 1. The distribution of vulnerabilities.



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution
Informational / Code Style / Best Practice	Informational vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

AS-IS overview

ACryptoSFarmV2.sol

Description

ACryptoSFarmV2 is a contract used to introduce a pools management and reward distribution.

Inheritance

ACryptoSFarmV2 contract is OwnableUpgradeable.

Usages

ACryptoSFarmV2 contract has following usages:

- using SafeMathUpgradeable for uint256;
- using SafeERC20Upgradeable for IERC20Upgradeable;

Structs

ACryptoSFarmV2 contract has following structures:

- UserInfo: struct to store data about user and his rewards.
- PoolInfo: struct to store data about pool and its variables.
- AdditionalReward: struct to store data about additional mint parameters for special rewards.

Enums

- *ACryptoSFarmV2* contract has no custom enums.

Events

ACryptoSFarmV2 contract has following custom events:

- Deposit: emit when new deposit has been done.
- Withdraw: emit when user withdraw his funds.

Modifiers

ACryptoSFarmV2 has following modifiers:

- onlyStrategist – checks whether a message sender is the *strategist* address or owner address.

Fields and constants

ACryptoSFarmV2 contract has following fields:

- ERC20Mintable public sushi
- uint256 public sushiPerBlock
- address public strategist
- address public harvestFeeAddress
- uint256 public harvestFee
- uint256 public maxBoost
- uint256 public boostFactor
- address public boostToken
- AdditionalReward[] public additionalRewards
- mapping (address => PoolInfo) public poolInfo
- mapping (address=> mapping (address => UserInfo)) public userInfo
- uint256 public totalAllocPoint

ACryptoSFarmV2 contract has following constants:

- uint256 public constant REWARD_DENOMINATOR = 10000

Functions

ACryptoSFarmV2 has following functions:

- ***pendingSushi***
  Description
  View function to see pending SUSHIs on frontend.
  Visibility
  External view
  Input parameters
  - address _lpToken,
  - address _user

  Constraints
  None
  Events emit
  None
  Output
  - uint256
- ***setBoostFactor***
  Description
  Set boost factor.

Visibility
External
Input parameters
  - uint256 _boostFactor

Constraints
onlyStrategist
Events emit
None
Output
None

- ***setMaxBoost***
  Description
  Set max boost factor.
  Visibility
  External
  Input parameters
    - uint256 _boostFactor

  Constraints
    - onlyStrategist

  Events emit
  None
  Output
  None
- ***setHarvestFee***
  Description
  Set harvest fee.
  Visibility
  External
  Input parameters
    - uint256 _harvestFee

  Constraints
    - onlyStrategist

  Events emit
  None
  Output
  None
- ***setHarvestFeeAddress***
  Description
  Set Harvest Fee Address.
  Visibility

External

Input parameters

  - uint256 _harvestFeeAddress

Constraints

onlyStrategist

Events emit

None

Output

None

- ***deleteAdditionalRewards***

  Description

  Delete Additional Rewards.

  Visibility

  External

  Input parameters

  None

  Constraints

  - onlyStrategist

  Events emit

  None

  Output

  None

- ***addAdditionalRewards***

  Description

  Add Additional Rewards.

  Visibility

  External

  Input parameters

  - address _to,
  - uint256 _reward

  Constraints

  - onlyStrategist

  Events emit

  None

  Output

  None

- ***setStrategist***

  Description

  Set Strategist address.

  Visibility

External

Input parameters

- address _strategist

Constraints

- onlyStrategist

Events emit

None

Output

None

- ***setSushiPerBlock***

Description

Set SushiPer Block.

Visibility

External

Input parameters

- uint256 _sushiPerBlock

Constraints

- onlyStrategist

Events emit

None

Output

None

- ***updatePool***

Description

Update reward variables of the given pool

Visibility

public

Input parameters

- address _lpToken

Constraints

None

Events emit

None

Output

None

- ***calculateWeight***

Description

Returns weight of a *user*.

Visibility

public view

Input parameters

  - address _lpToken,
  - address _user

Constraints

None

Events emit

None

Output

uint256

- **deposit**

  Description

  Deposit LP tokens to MasterChef for SUSHI allocation.

  Visibility

  public

  Input parameters

  - address _lpToken,
  - uint256 _amount

  Constraints

  None

  Events emit

  - Deposit

  Output

  None

- ***withdraw***

  Description

  Withdraw LP tokens from MasterChef.

  Visibility

  public

  Input parameters

  - address _lpToken
  - uint256 _amount

  Constraints

  - require(user.amount >= _amount, "withdraw: not good");

  Events emit

  - Withdraw

  Output

  None

- ***harvest***

  Description

  Withdraw LP harvest tokens from MasterChef.

Visibility
public
Input parameters
  - address _lpToken

Constraints
None
Events emit
None
Output
None

- ***set***

  Description
  Update the given pool's SUSHI allocation point
  Visibility
  public
  Input parameters
    - address _lpToken
    - uint256 _allocPoint
    - uint256 _withdrawalFee

  Constraints
    - onlyStrategist modifier

  Events emit
  None
  Output
  None

- ***safeSushiTransfer***

  Description
  Safe sushi transfer function.
  Visibility
  Internal
  Input parameters
    - address _to,
    - uint256 _amount

  Constraints
    - onlyStrategist

  Events emit
  None
  Output
  None

Audit overview

🟥🟥🟥🟥 Critical

No critical issues were found.

🟨🟨🟨 High

1. The *addAdditionalRewards* function allows owners to mint any amount of tokens to any address unlimitedly.

This behavior is described in the security-and-risks page and is not an issue.

🟩🟩 Medium

1. User weight is a one of the basic parameters to calculate reward. It depends on total pool size and user funds amount. It is updated only on withdraw and deposit functions calls. As a result, when pool amount is small, a user can get a large weight value. And when the pool become bigger, weight of the user will not be changed. But reward credit value will be calculated based on this value.

   We recommend updating a user weight before calculating a reward sum.

   This actually will never happen because the “boost weight” is limited by the % of the pool. For example, if user has %1 share of acsACS (boostToken), his maximum boost will be 1.5 * 1% = 1.5% of the pool. So if the pool is small, say 10 ETH, his maximum boost will be 0.15 ETH. When the pool becomes big, say 1000 ETH, his maximum boost will be 15 ETH, but only up to 1.5X his stake (amount) in the pool. So, there should be no way this can be exploited.

🟦 Low

No low severity issues were found.

🟦 Informational/ Code style / Best Practice

1. The code contains a lot of duplicates lines that could be extracted to separate function. For example:

   a. Reward credit calculation

   b. Pool weight update calculation

   c. Reward dept calculation

   d. Sushi reward value calculation

   e. “accSushiPerShare” value calculation

2. Some code-style issues were found by the static code analyzer.

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract, high-level description of functionality was presented in As-Is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found **2** informational issues during the audit.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.