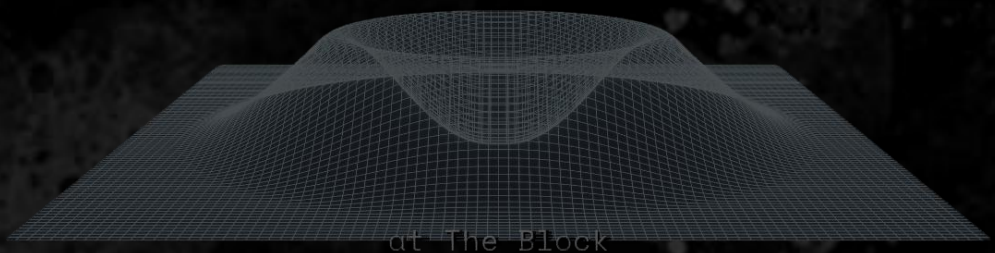
CHAINLION

FarmReward

Smart Contract Audit Report

OCT 09th,2022

NO.OC002210090002



CATALOGUE

1. PROJECT SUMMARY 3

2. AUDIT SUMMARY 3

3. VULNERABILITY SUMMARY 3

4. EXECUTIVE SUMMARY 4

5. DIRECTORY STRUCTURE 5

6. VULNERABILITY DISTRIBUTION 5

7. AUDIT CONTENT 6

7.1. CODING SPECIFICATION 6

7.1.1. Compiler Version【security】 6

7.1.2. Return value verification【security】 6

7.1.3. Constructor writing【security】 7

7.1.4. Key event trigger【security】 8

7.1.5. Address non-zero check【security】 8

7.1.6. Code redundancy check【security】 8

7.2. CODING DESIGN 9

7.2.1. Shaping overflow detection【security】 9

7.2.2. Reentry detection【security】 10

7.2.3. Rearrangement attack detection【security】 10

7.2.4. Replay Attack Detection【security】 11

7.2.5. False recharge detection【security】 11

7.2.6. Access control detection【security】 12

7.2.7. Denial of service detection【security】 13

7.2.8. Conditional competition detection【security】 13

7.2.9. Consistency detection【security】 14

7.2.10.Variable coverage detection【security】 14

7.2.11.Random number detection【security】 15

7.2.12.Numerical operation detection【security】 15

7.2.13.Call injection detection【security】 16

8. APPENDIX:ANALYSIS TOOLS 16

8.1. SOLGRAPH 16

8.2. SOL2UML 16

8.3. REMIX-IDE 16

8.4. ETHERSPLAY 17

8.5. MYTHRIL 17

8.6. ECHIDNA 17

9. DISCLAIMERS.... 17

1. PROJECT SUMMARY

Entry type	Specific description
Entry name	FarmReward
Project type	DEFI
Application platform	BSC
DawnToken	0xDC06a99c74DF8849074a3261A7163bEe20d8C906

2. AUDIT SUMMARY

Entry type	Specific description
Project cycle	OCT/08/2022-OCT/09/2022
Audit method	Black box test、White box test、Grey box test
Auditors	ONE

3. VULNERABILITY SUMMARY

Audit results are as follows:

Entry type	Specific description
Serious vulnerability	0
High risk vulnerability	0
Moderate risk	0
Low risk vulnerability	0

Security vulnerability rating description：

1) **Serious vulnerability：** Security vulnerabilities that can directly cause token contracts or user capital losses， For example: shaping overflow vulnerability、

Fake recharge vulnerability、Reentry attacks, vulnerabilities, etc.

2) **High risk vulnerability：** Security vulnerabilities that can directly cause the contract to fail to work normally, such as reconstructed smart contract caused by constructor design error, denial of service vulnerability caused by unreasonable design of require / assert detection conditions, etc.

3) **Moderate risk：** Security problems caused by unreasonable business logic design, such as accuracy problems caused by unreasonable numerical operation sequence design, variable ambiguous naming, variable coverage, call injection, conditional competition, etc.

4) **Low risk vulnerability：** Security vulnerabilities that can only be triggered by users with special permissions, such as contract backdoor vulnerability, duplicate name pool addition vulnerability, non-standard contract coding, contract detection bypass, lack of necessary events for key state variable change, and security vulnerabilities that are harmful in theory but have harsh utilization conditions.

4. EXECUTIVE SUMMARY

This report is prepared for **FarmReward** smart contract， The purpose is to find the security vulnerabilities and non-standard coding problems in the smart contract through the security audit of the source code of the smart contract. This audit mainly involves the following test methods:

White box test

Conduct security audit on the source code of smart contract and check the

security issues such as coding specification, DASP top 10 and business logic design

Grey box test

Deploy smart contracts locally and conduct fuzzy testing to check function robustness, function call permission and business logic security

Black box test

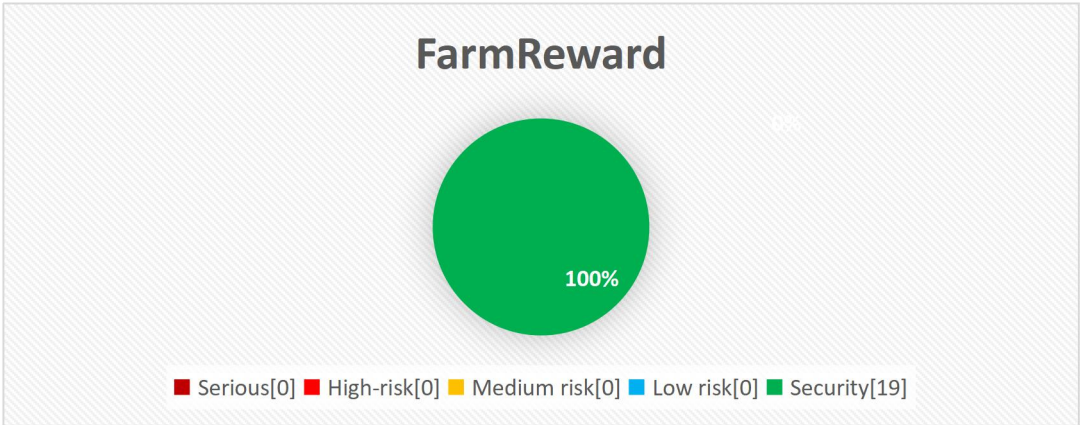
Conduct security test attacks on smart contracts from the perspective of attackers, combined with black-and-white and testing techniques, to check whether there are exploitable vulnerabilities.

This audit report is subject to the latest contract code provided by the current project party, does not include the newly added business logic function module after the contract upgrade, does not include new attack methods in the future, and does not include web front-end security and server-side security.

5. Directory structure

FarmReward.sol

6. Vulnerability distribution



7. Audit content

7.1. Coding specification

Smart contract supports contract development in programming languages such as solid, Vyper, C + +, Python and rust. Each programming language has its own coding specification. In the development process, the coding specification of the development language should be strictly followed to avoid security problems such as business function design defects.

7.1.1. Compiler Version【security】

Audit description： The compiler version should be specified in the smart contract code. At the same time, it is recommended to use the latest compiler version. The old version of the compiler may cause various known security problems. At present, the latest version is v 0.8 x. And this version has been protected against shaping overflow.

Audit results： According to the audit, the compiler version used in the smart contract code is 0.8.x, so there is no such security problem.

Safety advice：NONE.

7.1.2. Return value verification【security】

Audit description： Smart contract requires contract developers to strictly follow EIP / tip and other standards and specifications during contract development. For

transfer, transferfrom and approve functions, Boolean values should be returned to feed back the final execution results. In the smart contract, the relevant business logic code often calls the transfer or transferfrom function to transfer. In this case, the return value involved in the transfer operation should be strictly checked to determine whether the transfer is successful or not, so as to avoid security vulnerabilities such as false recharge caused by the lack of return value verification.

Audit results： According to the audit, there is no embedded function calling the official standards transfer and transferfrom in the smart contract, so there is no such security problem.

Safety advice： NONE.

7.1.3. Constructor writing【security】

Audit description： In solid v0 The smart contract written by solidity before version 4.22 requires that the constructor must be consistent with the contract name. When the constructor name is inconsistent with the contract name, the constructor will become an ordinary public function. Any user can call the constructor to initialize the contract. After version V 0.4.22, The constructor name can be replaced by constructor, so as to avoid the coding problems caused by constructor writing.

Audit results： After audit, the constructor in the smart contract is written correctly, and there is no such security problem.

```
32  contract ERC20 is Context, IERC20, IERC20Metadata {
33      mapping(address => uint256) private _balances;
34
35      mapping(address => mapping(address => uint256)) private _allowances;
36
37      uint256 private _totalSupply;
38
39      string private _name;
40      string private _symbol;
41
42
43      constructor(string memory name_, string memory symbol_,uint256 amount_) {
44          _name = name_;
45          _symbol = symbol_;
46          _mint(msg.sender, amount_);
47      }
48
```

Safety advice：NONE.

7.1.4. Key event trigger【security】

Audit description： Most of the key global variable initialization or update operations similar to setXXX exist in the smart contract. It is recommended to trigger the corresponding event through emit when operating on similar key events.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

7.1.5. Address non-zero check【security】

Audit description： The smart contract initializes the key information of the contract through the constructor. When it comes to address initialization, the address should be non-zero checked to avoid irreparable economic losses.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

7.1.6. Code redundancy check【security】

Audit description： The deployment and execution of smart contracts need to

consume certain gas costs. The business logic design should be optimized as much as possible, while avoiding unnecessary redundant code to improve efficiency and save costs.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

7.2. Coding design

DASP top 10 summarizes the common security vulnerabilities of smart contracts. Smart contract developers can study smart contract security vulnerabilities before developing contracts to avoid security vulnerabilities during contract development. Contract auditors can quickly audit and check the existing security vulnerabilities of smart contracts according to DASP top 10.

7.2.1. Shaping overflow detection【security】

Audit description： Solid can handle 256 digits at most. When the number is unsigned, the maximum value will overflow by 1 to get 0, and 0 minus 1 will overflow to get the maximum value. The problem of shaping overflow often appears in the relevant logic code design function modules such as transaction transfer, reward calculation and expense calculation. The security problems caused by shaping overflow are also very serious, such as excessive coinage, high sales and low income, excessive distribution, etc. the problem of shaping overflow can be solved by using solid V 0.8 X version or by using the safemath library officially provided by openzenppelin.

Audit results： According to the audit, the smart contract is applicable to the compiler of version 0.8.0, and the safemath library is used for numerical operation, which better prevents the problem of shaping overflow.

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

interface IERC20 {
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address to, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address from, address to, uint256 amount) external returns (bool);
}
```

Safety advice：NONE.

7.2.2. Reentry detection【security】

Audit description： The in solidity provides call Value(), send(), transfer() and other functions are used for transfer operation. When call When value() sends ether, it will send all gas for transfer operation by default. If the transfer function can be called recursively again through call transfer, it can cause reentry attack.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

7.2.3. Rearrangement attack detection【security】

Audit description：Rearrangement attack means that miners or other parties try to compete with smart contract participants by inserting their information into the list or mapping, so that attackers have the opportunity to store their information in the contract.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

7.2.4. Replay Attack Detection【security】

Audit description： When the contract involves the business logic of delegated management, attention should be paid to the non reusability of verification to avoid replay attacks. In common asset management systems, there are often delegated management businesses. The principal gives the assets to the trustee for management, and the principal pays a certain fee to the trustee. In similar delegated management scenarios, it is necessary to ensure that the verification information will become invalid once used.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

7.2.5. False recharge detection【security】

Audit description： When a smart contract uses the transfer function for transfer, it should use require / assert to strictly check the transfer conditions. It is not recommended to use if Use mild judgment methods such as else to check, otherwise it will misjudge the success of the transaction, resulting in the security problem of false recharge.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

7.2.6. Access control detection【security】

Audit description：Solid provides four function access domain Keywords: public, private, external and internal to limit the scope of function. In the smart contract, the scope of function should be reasonably designed to avoid the security risk of improper access control. The main differences of the above four keywords are as follows:

1．public： The marked function or variable can be called or obtained by any account, which can be a function in the contract, an external user or inherit the function in the contract

2．external：The marked functions can only be accessed from the outside and cannot be called directly by the functions in the contract, but this can be used Func() calls this function as an external call

3．private： Marked functions or variables can only be used in this contract (Note: the limitation here is only at the code level. Ethereum is a public chain, and anyone can directly obtain the contract status information from the chain)

4．internal: It is generally used in contract inheritance. The parent contract is marked as an internal state variable or function, which can be directly accessed and called by the child contract (it cannot be directly obtained and called externally)

Audit results：After audit, there is no such security problem.

Safety advice： NONE.

7.2.7. Denial of service detection【security】

Audit description：Denial of service attack is a DoS attack on Ethereum contract, which makes ether or gas consume a lot. In more serious cases, it can make the contract code logic unable to operate normally. The common causes of DoS attack are: unreasonable design of require check condition, uncontrollable number of for cycles, defects in business logic design, etc.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

7.2.8. Conditional competition detection【security】

Audit description： The Ethereum node gathers transactions and forms them into blocks. Once the miners solve the consensus problem, these transactions are considered effective. The miners who solve the block will also choose which transactions from the mine pool will be included in the block. This is usually determined by gasprice transactions. Attackers can observe whether there are transactions in the transaction pool that may contain problem solutions, After that, the attacker can obtain data from this transaction, create a higher-level transaction gasprice, and include its transaction in a block before the original, so as to seize the original solution.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

7.2.9. Consistency detection【security】

Audit description： The update logic in smart contract (such as token quantity update, authorized transfer quota update, etc.) is often accompanied by the check logic of the operation object (such as anti overflow check, authorized transfer quota check, etc.), and when the update object is inconsistent with the check object, the check operation may be invalid, Thus, the conditional check logic is ignored and unexpected logic is executed. For example, the authorized transfer function function function transfer from (address _from, address _to, uint256 _value) returns (bool success) is used to authorize others to transfer on behalf of others. During transfer, the permission [_from] [MSG. Sender] authorized transfer limit will be checked, After passing the check, the authorized transfer limit will be updated at the same time of transfer. When the update object in the update logic is inconsistent with the check object in the check logic, the authorized transfer limit of the authorized transfer user will not change, resulting in that the authorized transfer user can transfer all the assets of the authorized account.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

7.2.10. Variable coverage detection【security】

Audit description： Smart contracts allow inheritance relationships, in which the child contract inherits all the methods and variables of the parent contract. If a global variable with the same name as the parent contract is defined in the child contract, it

may lead to variable coverage and corresponding asset losses.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

7.2.11. Random number detection【security】

Audit description： Random numbers are often used in smart contracts. When designing the random number generation function, the generation and selection of random seeds should avoid the data information that can be queried on the blockchain, such as block Number and block Timestamp et al. These data are vulnerable to the influence of miners, resulting in the predictability of random numbers to a certain extent.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

7.2.12. Numerical operation detection【security】

Audit description： Solidity supports addition, subtraction, multiplication, division and other conventional numerical operations, but solidty does not support floating-point types. When multiplication and division operations exist at the same time, the numerical operation order should be adjusted reasonably to reduce the error as much as possible.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

7.2.13. Call injection detection【security】

Audit description：In the solid language, you can call a contract or a method of a local contract through the call method. There are roughly two ways to call: < address > Call (method selector, arg1, arg2,...) or < address > Call (bytes). When using call call, we can pass method selectors and parameters by passing parameters, or directly pass in a byte array. Based on this function, it is recommended that strict permission check or hard code the function called by call when using call function call.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

8. Appendix:Analysis tools

8.1.Solgraph

Solgraph is used to generate a graph of the call relationship between smart contract functions, which is convenient for quickly understanding the call relationship between smart contract functions.

Project address：https://github.com/raineorshine/solgraph

8.2.Sol2uml

Sol2uml is used to generate the calling relationship between smart contract functions in the form of UML diagram.

Project address：https://github.com/naddison36/sol2uml

8.3.Remix-ide

Remix is a browser based compiler and IDE that allows users to build contracts and debug transactions using the solid language.

Project address：http://remix.ethereum.org

8.4.Ethersplay

Etherplay is a plug-in for binary ninja. It can be used to analyze EVM bytecode and graphically present the function call process.

Project address：https://github.com/crytic/ethersplay

8.5.Mythril

Mythril is a security audit tool for EVM bytecode, and supports online contract audit.

Project address：https://github.com/ConsenSys/mythril

8.6.Echidna

Echidna is a security audit tool for EVM bytecode. It uses fuzzy testing technology and supports integrated use with truss.

Project address：https://github.com/crytic/echidna

9. DISCLAIMERS

Chainlion only issues this report on the facts that have occurred or existed before the issuance of this report, and assumes corresponding responsibilities. For the facts occurring or existing after the issuance, chainlion cannot judge the security status of its smart contract, and is not responsible for it. The security audit analysis and other contents in this report are only based on the documents and materials provided by the information provider to chainlion as of the issuance of this report.

Chainlion assumes that the information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed or reflected inconsistent with the actual situation, chainlion shall not be liable for the losses and adverse effects caused thereby. Chainlion only conducted the agreed safety audit on the safety of the project and issued this report. Chainlion is not responsible for the background and other conditions of the project