SLOWMIST

Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.01.12, the SlowMist security team received the DAS team's security audit application for Das Contract Reverse, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

Level	Description
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Short Address Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Uninitialized Storage Pointers Vulnerability
- Arithmetic Accuracy Deviation Vulnerability
- tx.origin Authentication Vulnerability

- "False top-up" Vulnerability
- Variable Coverage Vulnerability
- Gas Optimization Audit
- Malicious Event Log Audit
- Redundant Fallback Function Audit
- Unsafe External Call Audit
- Explicit Visibility of Functions State Variables Audit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Audit Version:

https://github.com/DeAccountSystems/das-contract-reverse

commit: ac8e4fca6d472c1f6aa5857f2832fa3c84800aed

Fixed Version:

https://github.com/DeAccountSystems/das-contract-reverse

commit: d8252fd54762fe08a865454d7c64e8ebc512026e

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status

NO	Title	Category	Level	Status
N1	Partial logic not implemented	Design Logic Audit	Suggestion	Fixed
N2	Not checking if `reverseName` already exists	Design Logic Audit	Medium	Ignored
N3	Potential risk of arbitrarily setting reverse name	Design Logic Audit	High	Fixed

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

ReverseLogic			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
setReverseName	External	Can Modify State	-
setReverseNameInternal	Internal	Can Modify State	-
getReverseName	Public	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Partial logic not implemented

Category: Design Logic Audit

Content

In the setReverseName function of the ReverseLogic contract, after the previous check, the specific check logic when the owner is still 0 address is not implemented.

Code location: reverse.sol

```
    function setReverseName(string memory reverseName, address account)
        external
    {
        if (msg.sender != account) {
            address owner = address(0);
            try IOwner(account).owner() {
                owner = IOwner(account).owner();
                require(owner == msg.sender, "need owner to set reverse");
            } catch {
                                 try IOwner(account).getOwner() {
                                         owner = IOwner(account).getOwner();
                                         require(owner == msg.sender, "need getOwner
to set reverse");
                } catch {}
                         }
            if (owner == address(0)) {
                // Other check
            }
            require(
                owner != address(0),
                "only allow owner to set reverse name"
            );
        }

        setReverseNameInternal(reverseName, account);
    }
```

Solution

It is recommended to complete the check as much as possible. (For example: the owner of the compound cToken contract is called admin)

Status

Fixed

[N2] [Medium] Not checking if `reverseName` already exists

Category: Design Logic Audit

Content

In the ReverseLogic contract, the user can set the reverse name through the setReverseName function, but it does not check whether the reverse name passed in by the user has been registered. Allowing the same reverse name to be registered may lead to a phishing risk.

Code location: reverse.sol

```
    function setReverseName(string memory reverseName, address account)
        external
    {
        if (msg.sender != account) {
            address owner = address(0);
            try IOwner(account).owner() {
                owner = IOwner(account).owner();
                require(owner == msg.sender, "need owner to set reverse");
            } catch {
                                 try IOwner(account).getOwner() {
                                         owner = IOwner(account).getOwner();
                                         require(owner == msg.sender, "need getOwner
to set reverse");
                } catch {}
                         }
            if (owner == address(0)) {
                // Other check
            }
            require(
                owner != address(0),
                "only allow owner to set reverse name"
            );
        }

        setReverseNameInternal(reverseName, account);
    }

    function setReverseNameInternal(string memory _name, address _account)
        internal
    {
        require(Address.isContract(_account), "EOA account not allowed.");
        names[_account] = _name;
        emit ReverseNameChanged(_account, _name);
    }
```

Solution

It is recommended to check whether the reverse name passed in by the user is already registered.

Status

Ignored; After communicating with the project side, the repetition check of the reverse name will be carried out at the SDK layer.

[N3] [High] Potential risk of arbitrarily setting reverse name

Category: Design Logic Audit

Content

In the ReverseLogic contract, the user can set the reverse name through the setReverseName function, which allows the contract to set itself. However, some contracts have the feature of arbitrary external calls, which will allow any user to set the reverse name of the contract.

Code location: reverse.sol

```
    function setReverseName(string memory reverseName, address account)
        external
    {
        if (msg.sender != account) {
            address owner = address(0);
            try IOwner(account).owner() {
                owner = IOwner(account).owner();
                require(owner == msg.sender, "need owner to set reverse");
            } catch {
                                 try IOwner(account).getOwner() {
                                         owner = IOwner(account).getOwner();
                                         require(owner == msg.sender, "need getOwner
to set reverse");
                } catch {}
                         }
            if (owner == address(0)) {
                // Other check
            }
            require(
                owner != address(0),
                "only allow owner to set reverse name"
            );
        }

        setReverseNameInternal(reverseName, account);
    }
```

Solution

It is recommended to prohibit such contracts from setting their own reverse name.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002201140001	SlowMist Security Team	2022.01.12 - 2022.01.14	Passed

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 high risk, 1 medium risk, 1 suggestion vulnerabilities. And 1 medium risk vulnerabilities were ignored; All other findings were fixed. The code was not deployed to the mainnet.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

SLOWMIST

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist