Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.06.07, the SlowMist security team received the MaskWallet team's security audit application for Mask InitialTwitterOffering, 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 party 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 Aduit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Type: Token + ITO Module

Code lines: 565 (including external introduction)

Complexity: Medium

Workload: 6 working days, smart contract security audit standards, related instructions and post-audit certificate query: https://www.slowmist.com/service-smart-contract-security-audit.html

Project code:

https://github.com/DimensionDev/InitialTwitterOffering

Audit based branch: https://github.com/DimensionDev/InitialTwitterOffering/tree/master

Initial audit commit: 6b95eca79e0436ff16f01610c2543e7a227cbbb0

Last audit commit: e05d299e80c1910fadd90bf021c52ced8b13f891

Please confirm the above project address is correct.

Before issuing the report, the project party needs to open source verification in the block browser and give the relevant contract address.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	The password verification can be bypassed in swap function	Design Logic Audit	Suggestion	Confirmed

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

https://etherscan.io/address/0xBe63b9ee7055Ea8A5691e93D31E949C03A3e2F54#code

4.2 Visibility Description

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

IQLF			
Function Name	Visibility	Mutability	Modifiers
supportsInterface	external	-	-

IQLF			
ifQualified	external	-	-
logQualified	external	can modify state	-
supportsInterface	external	-	-

HappyTokenPool			
Function Name	Visibility	Mutability	Modifiers
constructor	public	can modify state	-
fill_pool	public	payable	-
swap	public	payable	-
check_availability	external	-	-
claim	external	can modify state	-
setUnlockTime	public	can modify state	-
destruct	public	can modify state	-
withdraw	public	can modify state	-

Context			
Function Name	Visibility	Mutability	Modifiers
_msgSender	internal	-	-
_msgData	internal	-	-

Ownable

Ownable			
Function Name	Visibility	Mutability	Modifiers
_msgSender	internal	-	-
_msgData	internal	-	-
constructor	internal	can modify state	-
owner	public	-	-
renounceOwnership	public	can modify state	onlyOwner
transferOwnership	public	can modify state	onlyOwner

QLF			
Function Name	Visibility	Mutability	Modifiers
constructor	internal	can modify state	-
owner	public	-	-
renounceOwnership	public	can modify state	onlyOwner
transferOwnership	public	can modify state	onlyOwner
_msgSender	internal	-	-
_msgData	internal	-	-
ifQualified	external	-	-
logQualified	external	can modify state	-
supportsInterface	external	-	-
supportsInterface	external	-	-

QLF			
constructor	public	can modify state	-
get_start_time	public	-	-
set_start_time	public	can modify state	onlyOwner
ifQualified	public	-	-
logQualified	public	can modify state	-
supportsInterface	external	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] The password verification can be bypassed in swap function

Category: Design Logic Audit

Content

contracts/ito.sol#178-255,

```
function swap (bytes32 id,
               bytes32 verification,
               uint256 exchange_addr_i,
               uint128 input_total,
               bytes32[] memory data)
public payable returns (uint256 swapped) {

    Pool storage pool = pool_by_id[id];
    Packed1 memory packed1 = pool.packed1;
    Packed2 memory packed2 = pool.packed2;
    Packed3 memory packed3 = pool.packed3;
    {
        bool qualified;
        string memory errorMsg;
        (qualified, errorMsg) =
IQLF(packed1.qualification_addr).logQualified(msg.sender, data);
        require(qualified, errorMsg);
    }
    require (packed3.start_time + base_time < block.timestamp, "Not started.");
    require (packed3.end_time + base_time > block.timestamp, "Expired.");
    // sha3(sha3(passowrd)[:40] + msg.sender) so that the raw password will never
appear in the contract
    require (verification == keccak256(abi.encodePacked(uint256(packed1.password),
msg.sender)),
             'Wrong Password');

    // revert if the pool is empty
    require (packed2.total_tokens > 0, "Out of Stock");

    address exchange_addr = pool.exchange_addrs[exchange_addr_i];
    uint256 ratioA = pool.ratios[exchange_addr_i*2];
    uint256 ratioB = pool.ratios[exchange_addr_i*2 + 1];
    // check if the input is enough for the desired transfer
    if (exchange_addr == DEFAULT_ADDRESS) {
        require(msg.value == input_total, 'No enough ether.');
    }

    uint128 swapped_tokens =
SafeCast.toUint128(SafeMath.div(SafeMath.mul(input_total, ratioB), ratioA));
    require(swapped_tokens > 0, "Better not draw water with a sieve");

    if (swapped_tokens > packed2.limit) {
        // don't be greedy - you can only get at most limit tokens
        swapped_tokens = packed2.limit;
        input_total = SafeCast.toUint128(SafeMath.div(SafeMath.mul(packed2.limit,
ratioA), ratioB));           // Update input_total
    } else if (swapped_tokens > packed2.total_tokens ) {
        // if the left tokens are not enough
        swapped_tokens = packed2.total_tokens;
        input_total =
SafeCast.toUint128(SafeMath.div(SafeMath.mul(packed2.total_tokens , ratioA),
ratioB));    // Update input_total
        // return the eth
        if (exchange_addr == DEFAULT_ADDRESS)
            payable(msg.sender).transfer(msg.value - input_total);
    }
    require(swapped_tokens <= packed2.limit);
// make sure again
    pool.exchanged_tokens[exchange_addr_i] =
SafeCast.toUint128(SafeMath.add(pool.exchanged_tokens[exchange_addr_i],
                                                                  input_total));
// update exchanged
```

```
    // penalize greedy attackers by placing duplication check at the very last
    require (pool.swapped_map[msg.sender] == 0, "Already swapped");

    // update the remaining tokens and swapped token mapping
    pool.packed2.total_tokens = SafeCast.toUint128(SafeMath.sub(packed2.total_tokens,
swapped_tokens));
    pool.swapped_map[msg.sender] = swapped_tokens;

    // transfer the token after state changing
    // ETH comes with the tx, but ERC20 does not - INPUT
    if (exchange_addr != DEFAULT_ADDRESS) {
        IERC20(exchange_addr).safeTransferFrom(msg.sender, address(this),
input_total);
    }

    // Swap success event
    emit SwapSuccess(id, msg.sender, exchange_addr, packed3.token_address,
input_total, swapped_tokens);

    // if unlock_time == 0, transfer the swapped tokens to the recipient address
(msg.sender) - OUTPUT
    // if not, claim() needs to be called to get the token
    if (packed3.unlock_time == 0) {
        IERC20(packed3.token_address).safeTransfer(msg.sender, swapped_tokens);
        emit ClaimSuccess(id, msg.sender, block.timestamp, swapped_tokens,
packed3.token_address);
    }

    return swapped_tokens;
}
```

Here is the part about password verification:

```
// sha3(sha3(passowrd)[:40] + msg.sender) so that the raw password will never appear
in the contract
require (verification == keccak256(abi.encodePacked(uint256(packed1.password),
msg.sender)),
         'Wrong Password');
```

packed1.password can be getten after fill_pool called, so an user can use it to pass the verification.

Solution

Such judgments can only prevent participants who use the Mask plug-in for transactions, and cannot effectively defend against malicious users. The defense can be enhanced by means such as whitelisting.

Status

Confirmed; Whitelists and random numbers are now used to strengthen defenses.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002106150003	SlowMist Security Team	2021.06.07 - 2021.06.15	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 suggestion vulnerabilities. And 1 suggestion vulnerabilities were confirmed and being fixed; All other findings were fixed.

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.