Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.07.05, the SlowMist security team received the OneWallet team's security audit application for OneWallet, 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

Audit Version Code：

https://github.com/polymorpher/one-

wallet/blob/9eed5823d5ab26649ed9295bf31b2bc6229ec448/code/contracts/ONEWallet.sol

Fixed Version Code：

https://github.com/polymorpher/one-

wallet/blob/2de6387e3a7f1277795357973f44d3775403c495/code/contracts/ONEWallet.sol

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	DoS risks and lack of access permission check	Authority Control Vulnerability	Suggestion	Fixed
N2	No check return value	Others	Low	Fixed
N3	Timestamp Dependence	Block data Dependence Vulnerability	Suggestion	Confirmed
N4	Race Conditions Vulnerability	Race Conditions Vulnerability	Critical	Fixed
N5	Business logic error	Others	Low	Fixed
N6	Business logic check bypass	Reentrancy Vulnerability	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:

- https://github.com/polymorpher/one-wallet/blob/9eed5823d5/code/contracts/ONEWallet.sol

ONEWallet			
Function Name	Visibility	Mutability	Modifiers

ONEWallet			
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
retire	External	Can Modify State	-
getInfo	External	-	-
getVersion	External	-	-
getCurrentSpending	External	-	-
getNonce	Public	-	-
getCommits	Public	-	-
commit	External	Can Modify State	-
revealTransfer	External	Can Modify State	isCorrectProof
revealRecovery	External	Can Modify State	isCorrectProof
revealSetLastResortAddress	External	Can Modify State	-
_drain	Internal	Can Modify State	-
_findCommit	Internal	-	-
_cleanupCommits	Internal	Can Modify State	-
_isRevealTimely	Internal	-	-
_revealPreCheck	Internal	-	-
_completeReveal	Internal	Can Modify State	-
_cleanupNonces	Internal	Can Modify State	-
_incrementNonce	Internal	Can Modify State	-

- https://github.com/polymorpher/one-wallet/blob/c0185b1de8/code/contracts/ONEWallet.sol

ONEWallet			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
retire	External	Can Modify State	-
getInfo	External	-	-
getVersion	External	-	-
getCurrentSpending	External	-	-
getNonce	External	-	-
getCommits	External	-	-
getAllCommits	External	-	-
findCommit	External	-	-
lookupCommit	External	-	-
commit	External	Can Modify State	-
_drain	Internal	Can Modify State	-
_transfer	Internal	Can Modify State	-
_recover	Internal	Can Modify State	-
_setRecoveryAddress	Internal	Can Modify State	-
_transferToken	Internal	Can Modify State	-
_getRevealHash	Internal	-	-

ONEWallet			
reveal	External	Can Modify State	-
_isCorrectProof	Internal	-	-
_cleanupCommits	Internal	Can Modify State	-
_isRevealTimely	Internal	-	-
_verifyReveal	Internal	-	-
_completeReveal	Internal	Can Modify State	-
_cleanupNonces	Internal	Can Modify State	-
_incrementNonce	Internal	Can Modify State	-

- https://github.com/polymorpher/one-wallet/blob/c0185b1de8/code/contracts/TokenTracker.sol

TokenTracker			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
onERC1155Received	External	Can Modify State	-
onERC1155BatchReceived	External	Can Modify State	-
supportsInterface	External	-	-
onERC721Received	External	Can Modify State	-
getTrackedTokens	External	-	-
_trackToken	Internal	Can Modify State	-
_untrackToken	Internal	Can Modify State	-

TokenTracker			
_overrideTrack	Internal	Can Modify State	-
_overrideTrackWithBytes	Internal	Can Modify State	-
_multiTrack	Internal	Can Modify State	-
_multiUntrack	Internal	Can Modify State	-
_asByte32	Internal	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] DoS risks and lack of access permission check

Category: Authority Control Vulnerability

Content

When the commit function is called, there is no permission check, and the size of the commits is not limited. Any user can add useless data to increase the length of the commits because when using commits, a for loop is used to traverse. When the length of commits is longer, it will cause a DoS issue due to the depth of the call in evm.

- https://github.com/polymorpher/one-wallet/blob/9eed5823d5ab26649ed9295bf31b2bc6229ec448/code/contracts/ONEWallet.sol#L80-L87

```
    function commit(bytes32 hash) external
     {
        //        require(!commitLocked, "Cleanup in progress. Queue is temporarily
locked. Please resubmit.");
        _cleanupCommits();
        (uint32 ct, bool completed) = _findCommit(hash);
        require(ct == 0 && !completed, "Commit already exists");
        Commit memory nc = Commit(hash, uint32(block.timestamp), false);
        commits.push(nc);
     }
```

Solution

It is recommended to add permission checks and limit the length of commits.

Status

Fixed; The issue has been fixed in this commit: c0185b1de8a2b5ac6b37d9ca4f81888f34dd1e2d

[N2] [Low] No check return value

Category: Others

Content

The send function is used to send the underlying asset, and the return value is not checked.

- https://github.com/polymorpher/one-wallet/blob/9eed5823d5ab26649ed9295bf31b2bc6229ec448/code/contracts/ONEWallet.sol

```
    function _drain() internal returns (bool) {
        return lastResortAddress.send(address(this).balance);
     }
```

Solution

It is recommended to use transfer to send the underlying assets or check the return value of send, and ensure that the return value is true

Status

Fixed; The issue has been fixed in this commit: 4b6a9d460e683fa21d8a183a76f2bc927817d4b7.

[N3] [Suggestion] Timestamp Dependence

Category: Block data Dependence Vulnerability

Content

It depends on the time stamp on the chain to obtain nonce. The nonce obtained by the same block height is the same.

- https://github.com/polymorpher/one-wallet/blob/9eed5823d5ab26649ed9295bf31b2bc6229ec448/code/contracts/ONEWallet.sol

```
function getNonce() public view returns (uint8) {
        uint32 index = uint32(block.timestamp) / interval - t0;
        return nonces[index];
    }
```

Solution

It is not recommended to use on-chain timestamps to generate data.

Status

Confirmed; Currently, the getNonce function is not used in the contract, it is only used for the client to get the data.

[N4] [Critical] Race Conditions Vulnerability

Category: Race Conditions Vulnerability

Content

When calling revealTransfer, revealRecovery and revealSetLastResortAddress functions, needs to input neighbors, indexWithNonce, and eotp parameters. When the transaction is pending, the data can be publicly accessed on the blockchain.

The attacker can steal the values of neighbors, indexWithNonce, and eotp, and then modify other parameters to construct a new commit hash to submit. In this way, the requested data can be constructed arbitrarily, and pack in advance by paying more gasprice or cooperating with miners to sort transactions, so assets in the contract will be stolen.

- https://github.com/polymorpher/one-wallet/blob/9eed5823d5ab26649ed9295bf31b2bc6229ec448/code/contracts/ONEWallet.sol

```
function commit(bytes32 hash) external
    {
        //        require(!commitLocked, "Cleanup in progress. Queue is temporarily
locked. Please resubmit.");
        _cleanupCommits();
        (uint32 ct, bool completed) = _findCommit(hash);
        require(ct == 0 && !completed, "Commit already exists");
        Commit memory nc = Commit(hash, uint32(block.timestamp), false);
        commits.push(nc);
    }

    function revealTransfer(bytes32[] calldata neighbors, uint32 indexWithNonce,
bytes32 eotp, address payable dest, uint256 amount) external
    isCorrectProof(neighbors, indexWithNonce, eotp)
    returns (bool) {
        //        bytes memory packedNeighbors = _pack(neighbors);
        bytes memory packed = bytes.concat(neighbors[0],
            bytes32(bytes4(indexWithNonce)), eotp, bytes32(bytes20(address(dest))),
bytes32(amount));
        bytes32 commitHash = keccak256(bytes.concat(packed));
        //        emit CheckingCommit(packed, commitHash);
        _revealPreCheck(commitHash, indexWithNonce);
        _completeReveal(commitHash);
        uint32 day = uint32(block.timestamp / SECONDS_PER_DAY);
        if (day > lastTransferDay) {
            spentToday = 0;
            lastTransferDay = day;
        }
        if (spentToday + amount > dailyLimit) {
            emit ExceedDailyLimit(amount, dailyLimit, spentToday, dest);
            return false;
        }
        if (address(this).balance < amount) {
            emit InsufficientFund(amount, address(this).balance, dest);
            return false;
        }
        bool success = dest.send(amount);
        // we do not want to revert the whole transaction if this operation fails,
since EOTP is already revealed
        if (!success) {
            emit UnknownTransferError(dest);
            return false;
        }
        spentToday += amount;
        return true;
    }

    function revealRecovery(bytes32[] calldata neighbors, uint32 indexWithNonce,
bytes32 eotp) external
    isCorrectProof(neighbors, indexWithNonce, eotp)
    returns (bool) {
        //        bytes memory packedNeighbors = _pack(neighbors);
        bytes memory packed = bytes.concat(
            neighbors[0],
            bytes32(bytes4(indexWithNonce)),
            eotp
        );
        bytes32 commitHash = keccak256(bytes.concat(packed));
        _revealPreCheck(commitHash, indexWithNonce);
        _completeReveal(commitHash);
        if (lastResortAddress == address(0)) {
            emit LastResortAddressNotSet();
            return false;
        }
        return _drain();
    }

    function revealSetLastResortAddress(bytes32[] calldata neighbors, uint32
indexWithNonce, bytes32 eotp, address payable lastResortAddress_) external
    {
        require(lastResortAddress == address(0), "Last resort address is already
set");
        bytes memory packed = bytes.concat(
            neighbors[0],
            bytes32(bytes4(indexWithNonce)),
            eotp,
            bytes32(bytes20(address(lastResortAddress_)))
        );
        bytes32 commitHash = keccak256(bytes.concat(packed));
        _revealPreCheck(commitHash, indexWithNonce);
        _completeReveal(commitHash);
        lastResortAddress = lastResortAddress_;
    }
```

Solution

It is recommended to package and record the request parameters when calling the commit function, And when calling the revealTransfer, revealRecovery and revealSetLastResortAddress functions, verify the input parameters and the parameter records in the commit to ensure that the parameters are consistent before calling.

Status

Fixed; The issue has been fixed in this commit: c0185b1de8a2b5ac6b37d9ca4f81888f34dd1e2d

[N5] [Low] Business logic error

Category: Others

Content

lastResortAddress can call receive function to excute _drain(), so it can bypass this check

`require(uint32(block.timestamp / interval)-t0> lifespan, "Too early to retire");`

In the case of `msg.sender == lastResortAddress`, lastResortAddress cannot be address(0), so `if (lastResortAddress == address(0)) { return;}` is a redundant check.

- https://github.com/polymorpher/one-wallet/blob/1ed4417b9ba7c6d6cf57b3a0a311023fca800b9a/code/contracts/ONEWallet.sol#L73-L98

```
    receive() external payable {
        emit PaymentReceived(msg.value, msg.sender);
        if (msg.value != AUTO_RECOVERY_TRIGGER_AMOUNT) {
            return;
        }
        if (msg.sender != lastResortAddress) {
            return;
        }
        if (lastResortAddress == address(0)) {
            return;
        }
        if (msg.sender == address(this)) {
            return;
        }
        emit AutoRecoveryTriggered(msg.sender);
        require(_drain());
    }


    function retire() external returns (bool)
    {
        require(uint32(block.timestamp / interval) - t0 > lifespan, "Too early to
retire");
```

```
        require(lastResortAddress != address(0), "Last resort address is not set");
        require(_drain(), "Recovery failed");
        return true;
    }
```

Solution

It is recommended to check the specific business logic and delete redundant codes.

Status

Fixed; The issue has been fixed in this commit: 2de6387e3a7f1277795357973f44d3775403c495

[N6] [High] Business logic check bypass

Category: Reentrancy Vulnerability

Content

The _transfer function uses a call to transfer assets and then updates the value of spentToday, which is a reentrant vulnerability. The issue can bypass this check `spentToday + amount > dailyLimit`.

- https://github.com/polymorpher/one-wallet/blob/1ed4417b9ba7c6d6cf57b3a0a311023fca800b9a/code/contracts/ONEWallet.sol#L205

```
function _transfer(address payable dest, uint256 amount) internal returns (bool) {
        uint32 day = uint32(block.timestamp / SECONDS_PER_DAY);
        if (day > lastTransferDay) {
            spentToday = 0;
            lastTransferDay = day;
        }
        if (spentToday + amount > dailyLimit) {
            emit ExceedDailyLimit(amount, dailyLimit, spentToday, dest);
            return false;
        }
        if (address(this).balance < amount) {
            emit InsufficientFund(amount, address(this).balance, dest);
            return false;
        }
        (bool success,) = dest.call{value : amount}("");
        // we do not want to revert the whole transaction if this operation fails,
since EOTP is already revealed
```

```
        if (!success) {
            emit UnknownTransferError(dest);
            return false;
        }
        spentToday += amount;
        emit PaymentSent(amount, dest);
        return true;
    }
```

Solution

It is recommended to use the Checks-Effects-Interactions coding standard, and limit the gas limit of the call method.

Status

Fixed; The issue has been fixed in this commit: c0185b1de8a2b5ac6b37d9ca4f81888f34dd1e2d

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0x002107130001	SlowMist Security Team	2021.07.05 - 2021.07.13	Passed

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found a critical risk, a high risk, two low risks, two suggestions, and a suggestion was confirmed, All the other issues have been fixed. The codes were 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.