Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.07.26, the SlowMist security team received the OTP Wallet team's security audit application for harmony otp wallet, 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:

https://github.com/hashmesan/harmony-

totp/blob/b38d90f1a930b32a4854daf12ee86f407c904e9e/contracts/otp_wallet.sol

Fixed Version:

https://github.com/hashmesan/harmony-

totp/tree/fdbf6831e05d12f0f43bc660cf11a3da8fa9ded2/contracts/otp_wallet.sol

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	Business logic error	Design Logic Audit	Low	Confirmed
N2	Unclear function implementation	Others	High	Fixed
N3	Excessive authority issues	Authority Control Vulnerability	Medium	Confirmed
N4	ConfirmMaterial leaked	Design Logic Audit	Medium	Fixed
N5	Incomplete function implementation	Others	Medium	Fixed
N6	Return value is not checked	"False top-up" Vulnerability	Medium	Fixed
N7	Signature replay issue	Replay Vulnerability	Low	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:

TOTPWallet			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-

TOTPWallet			
initialize	External	Can Modify State	-
registerENS	External	Can Modify State	onlyFromWalletOrOwnerWhenUnlocked
getRequiredSignatures	Public	-	-
executeMetaTx	External	Can Modify State	-
makeTransfer	External	Can Modify State	onlyFromWalletOrOwnerWhenUnlocked
getOwner	Public	-	-
getCounter	Public	-	-
setHashStorageId	External	Can Modify State	onlyFromWalletOrOwnerWhenUnlocked
getHashStorageId	Public	-	-
getRootHashes	Public	-	-
setDrainAddress	External	Can Modify State	onlyFromWalletOrOwnerWhenUnlocked
setDailyLimit	External	Can Modify State	onlyFromWalletOrOwnerWhenUnlocked
addGuardian	External	Can Modify State	onlyFromWalletOrOwnerWhenUnlocked
revokeGuardian	External	Can Modify State	onlyFromWalletOrOwnerWhenUnlocked
isGuardian	Public	-	-
getGuardians	Public	-	-
startRecoverGuardianOnly	External	Can Modify State	onlySelf
startRecoverCommit	External	Can Modify State	onlySelf

TOTPWallet			
startRecoveryReveal	External	Can Modify State	onlySelf onlyValidTOTP
isRecovering	External	-	-
cancelRecovery	External	Can Modify State	onlyFromWalletOrOwnerWhenUnlocked
getRecovery	External	-	-
finalizeRecovery	External	Can Modify State	-
upgradeMasterCopy	External	Can Modify State	onlySelf
getMasterCopy	Public	-	-
functionPrefix	Internal	-	-
<Receive Ether>	External	Payable	-

4.3 Vulnerability Summary

[N1] [Low] Business logic error

Category: Design Logic Audit

Content

The ownerSignatureRequirement is entered by the user. The code uses 'if' and 'else if' but does not use 'else'.

There have logical security issues:

1.When the conditions of `L109:if` and `L115:else if` are not met, the code will not “return false;” and will continue to execute.

2.When `L462:_isOwner` is not met, the code will not "return false;" and will continue to execute.

- contracts/features/metatx.sol#L108-L121

```
function validateSignatures(Core.Wallet storage wallet_, bytes32 _signHash, bytes
memory _signatures, Core.OwnerSignature ownerSignatureRequirement) internal view
returns (bool)
    {
        if (_signatures.length == 0) {
            return true;
        }
        address lastSigner = address(0);
        address[] memory guardians = wallet_.guardians;
        bool isGuardian;

        for (uint256 i = 0; i < _signatures.length / 65; i++) {
            address signer = recoverSigner(_signHash, _signatures, i);

            if (i == 0) {
                if (ownerSignatureRequirement == Core.OwnerSignature.Required) {
                    // First signer must be owner
                    if (_isOwner(wallet_, signer)) {
                        continue;
                    }
                    return false;
                } else if (ownerSignatureRequirement == Core.OwnerSignature.Optional)
{
                    // First signer can be owner
                    if (_isOwner(wallet_, signer)) {
                        continue;
                    }
                }
            }

            if (signer <= lastSigner) {
                return false; // Signers must be different
            }
            lastSigner = signer;
            isGuardian = isGuardianAddress(guardians, signer);
            if (!isGuardian) {
                return false;
            }
        }
        return true;
    }
```

Solution

It is recommended to use else to "return false;" those that do not meet the conditions.

Status

Confirmed; This is a feature, the code logic will use the getRequiredSignatures function to determine whether _isOwner or isGuardianAddress needs to be verified.

[N2] [High] Unclear function implementation

Category: Others

Content

The `MetaTx.executeMetaTx` is commented out, which will cause the parameters entered in the `executeMetaTx` function to not be used. This seems to be a bug, and you need to confirm the business logic here.

- contracts/otp_wallet.sol#L178

```
function executeMetaTx(
            bytes   calldata data,
            bytes   calldata signatures,
            uint256 nonce,
            uint256 gasPrice,
            uint256 gasLimit,
            address refundToken,
            address payable refundAddress
        ) external
    {
        uint gasLeft = gasleft();
        uint8 requiredSignatures;
        Core.SignatureRequirement memory sigRequirement;
        (sigRequirement.requiredSignatures, sigRequirement.ownerSignatureRequirement)
= getRequiredSignatures(data);

        //MetaTx.executeMetaTx(wallet, refundAddress, data, signatures, nonce,
gasPrice, gasLimit, refundAddress, sigRequirement);
        bool success;
        bytes memory returnData;
        (success, returnData) = address(this).call(data);

        if(gasPrice > 0 && success && refundAddress != address(0x0)) {
            uint gasUsed = gasLeft - gasleft() + 70000; //35k overhead
            refundAddress.transfer(gasUsed);
        }
        emit TransactionExecuted(success, returnData, 0x0);
    }
```

Solution

Need to confirm the business logic here with the project team.

Status

Fixed; This issue has been fixed in commit: d04f69c00ce4a230c7106290b973f7e759e05d53

[N3] [Medium] Excessive authority issues

Category: Authority Control Vulnerability

Content

The Owner and drainAddr can transfer ether from the contract to a new address.

- contracts/otp_wallet.sol#L190-L200

```
function makeTransfer(address payable to, uint amount) external
onlyFromWalletOrOwnerWhenUnlocked()
    {
        require(wallet.isUnderLimit(amount), "over withdrawal limit");
        require(address(this).balance >= amount, "not enough balance");

        wallet.spentToday += amount;
        //to.transfer(amount);
        to.call{value: amount, gas: 100000}("");

        emit WalletTransfer(to, amount);
    }
```

- contracts/otp_wallet.sol#L346-L354

```
receive() external payable {
        if (msg.value > 0) {
            if(msg.sender == wallet.drainAddr && msg.value == 1 ether) {
                uint amount = address(this).balance;
                wallet.drainAddr.call{value: amount, gas: 100000}("");
            }
            emit Deposit(msg.sender, msg.value);
        }
    }
```

Solution

It is recommended to confirm the design of the business logic and whether drainAddr is allowed to transfer the assets in the contract.

Status

Confirmed; harmony totp team response:

The owner address is a unique address only known by the user when the contract is created. Therefore the owner address security is unique per user. The drain address is also assigned by the user as a last resort method to recover the funds from the account by sending 1.0 ONE to the contract. It is optional.

[N4] [Medium] ConfirmMaterial leaked

Category: Design Logic Audit

Content

ConfirmMaterial will be stolen when it is submitted to the chain. and confirmMaterial can be used many times.

- contracts/otp_wallet.sol#L291-L300

```
function startRecoveryReveal(address newOwner, bytes32[] calldata confirmMaterial)
onlySelf() onlyValidTOTP(confirmMaterial) external {
        bytes32 hash = keccak256(abi.encodePacked(newOwner, confirmMaterial[0]));
        require(wallet.commitHash[hash], "NO COMMIT");

        //wallet.startRecovery(newOwner);
        wallet.owner = newOwner;

        delete wallet.commitHash[hash];
        wallet.counter = wallet.counter + 1;
    }
```

- contracts/otp_wallet.sol#L110-L128

```
 modifier onlyValidTOTP(bytes32[] memory confirmMaterial)
    {
        bytes32 reduced = Recovery._reduceConfirmMaterial(confirmMaterial);
        uint32 counterProvided = Recovery._deriveChildTreeIdx(wallet.merkelHeight,
confirmMaterial[confirmMaterial.length-1]);
        require(counterProvided >= wallet.counter, "Provided counter must be greater
or same");

        // Google Authenticator doesn't allow custom counter or change counter back;
so we must allow room to fudge
        // allow some room if the counters were skipped at some point
        require(counterProvided - wallet.counter  < 50, "Provided counter must not be
more than 20 steps");

        bool foundMatch = false;
        for (uint32 i = 0; i < wallet.rootHash.length; i++) {
            if(reduced==wallet.rootHash[i]) {
                foundMatch = true;
            }
        }
        require(foundMatch, "UNEXPECTED PROOF");
        _;
    }
```

Solution

Need to get more details about ConfirmMaterial to determine the scope of influence.

Status

Fixed; The issue has been fixed in commit: d04f69c00ce4a230c7106290b973f7e759e05d53

[N5] [Medium] Incomplete function implementation

Category: Others

Content

`ex` is a memory type variable, and `ex` is assigned in the `executeMetaTx` function, but `ex` is not stored in storage. `executeMetaTx` seems to be an incomplete function.

- contracts/features/metatx.sol#L15-L47

```
function executeMetaTx(
            Core.Wallet storage _wallet,
            address sw,
            bytes   calldata _data,
            bytes   calldata signatures,
            uint256 nonce,
            uint256 gasPrice,
            uint256 gasLimit,
            address refundAddress,
            Core.SignatureRequirement memory sigRequirement
        ) public
    {
        StackExtension memory ex;

        //require(sigRequirement.requiredSignatures > 0 ||
sigRequirement.ownerSignatureRequirement == Core.OwnerSignature.Anyone, "RM: Wrong
signature requirement");
        require(sigRequirement.requiredSignatures * 65 == signatures.length, "Wrong
number of signatures");

        ex.signHash = getSignHash(
            address(this),
            0,
            _data,
            nonce,
            gasPrice,
            gasLimit,
            address(0),
            refundAddress);

        require(validateSignatures(_wallet, ex.signHash, signatures,
sigRequirement.ownerSignatureRequirement), "RM: Invalid signatures");

        (ex.success, ex.returnData) = address(sw).call(_data);

        // refund
    }
```

Solution

Need to confirm the implementation logic of the code with the project team.

Status

Fixed; This issue has been fixed in commit: d04f69c00ce4a230c7106290b973f7e759e05d53

[N6] [Medium] Return value is not checked

Category: "False top-up" Vulnerability

Content

makeTransfer function do not check the return value. When the call return false, the function will not throw an error, but will continue to execute and record the event, and event log is wrong information.

- contracts/otp_wallet.sol#L197

```
function makeTransfer(address payable to, uint amount) external
onlyFromWalletOrOwnerWhenUnlocked()
    {
        require(wallet.isUnderLimit(amount), "over withdrawal limit");
        require(address(this).balance >= amount, "not enough balance");

        wallet.spentToday += amount;
        //to.transfer(amount);
        to.call{value: amount, gas: 100000}("");

        emit WalletTransfer(to, amount);
    }
```

- contracts/otp_wallet.sol#L346-L354

```
receive() external payable {
        if (msg.value > 0) {
            if(msg.sender == wallet.drainAddr && msg.value == 1 ether) {
                uint amount = address(this).balance;
                wallet.drainAddr.call{value: amount, gas: 100000}("");
            }
            emit Deposit(msg.sender, msg.value);
        }
```

```
    }
```

Solution

It is recommended to check the return value of call.

Status

Fixed; This issue has been fixed in commit: d04f69c00ce4a230c7106290b973f7e759e05d53

[N7] [Low] Signature replay issue

Category: Replay Vulnerability

Content

The getSignHash without setting the ChainID, when the chain split, the signed data can be replayed. The nonce is entered by the user. If the user incorrectly signs two transactions with the same nonce, it will cause replay issues.

- contracts/features/metatx.sol#39-69

```
function getSignHash(
        address _from,
        uint256 _value,
        bytes memory _data,
        uint256 _nonce,
        uint256 _gasPrice,
        uint256 _gasLimit,
        address _refundToken,
        address _refundAddress
    )
        internal
        pure
        returns (bytes32)
    {
        return keccak256(
            abi.encodePacked(
                "\x19Ethereum Signed Message:\n32",
                keccak256(abi.encodePacked(
                    bytes1(0x19),
                    bytes1(0),
                    _from,
                    _value,
                    _data,
                    uint(0), //block.chainid,
                    _nonce,
                    _gasPrice,
                    _gasLimit,
                    _refundToken,
                    _refundAddress))
        ));
    }
```

Solution

- It is recommended to check block.chainid in the contract.
- It is recommended to add a check for nonce in the contract.

Status

Fixed; This issue has been fixed in commit:fdbf6831e05d12f0f43bc660cf11a3da8fa9ded2;

harmony totp team response:

because it only allows previous + 1 with > (greater comparison). the project team uses the timestamp to generate the nonce. the project team prefers the client not have to query the smart contract or design where race conditions can happen.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0x002108060002	SlowMist Security Team	2021.07.26 - 2021.08.06	Low Risk

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 high risk, 4 medium risks, a low risk issues. A high risk has been fixed, 3 medium risks have been fixed, a low risk has been fixed. other issues have been confirmed, 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.