SLOWMIST

Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.03.22, the SlowMist security team received the Open Social team's security audit application for Open Social - Abstract Account, 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.
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:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

Audit Version:

https://gitlab.com/Keccak256-evg/opensocial/abstractaccount/-/tree/osp_master

commit: e10d262cab5b4c2e30135b439fa61792b63abb07

Fixed Version:

https://github.com/Open-Social-Protocol/abstract-account

commit: 1dc8ec789ab2a19ed8be5fcacd519fa0c238265f

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N2	Ordinary users with permitCalls may use the owner privilege through arbitrary contract calls	Design Logic Audit	Suggestion	Fixed
N3	Missing the event records	Others	Suggestion	Fixed
N4	Missing zero address validation	Others	Suggestion	Fixed
N5	ERC777 reentrancy risk reminder	Unsafe External Call Audit	Suggestion	Fixed

4 Code Overview

4.1 Contracts Description

These are Open Social Osp AA Wallet (Account Abstraction Wallet) and paymaster parts. The Osp smart contract wallet comprises two primary contracts: OspAccount and OspAccountFactory. The OspAccount contract provides a versatile account structure, supporting standard execution, session key management, token callback handling, and setting recovery addresses. It allows users to authorize various transactions, upgrade and recover accounts. The OspAccountFactory contract is responsible for creating instances of OspAccount. Users can create new OspAccount instances via this contract, establishing upgradeable.

For the paymaster part, the VerifyingSingletonPaymaster contract facilitates the use of an external service to decide whether to pay for user operations' gas fees. It relies on an external signer to sign transactions, where the user must first pass their operation to this external signer for various off-chain verifications before the signature. The WhitelistOperationVerifyingPaymaster contract, based on the eth-infinitism sample VerifyingPaymaster, introduces a whitelist management system for operations, controlled by an administrator. And the GasInBox contract, functioning as a gas station model.

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:

OspAccountFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
createAccount	Public	Can Modify State	-
getCreate2Address	Public	-	-
setAccountImplementation	External	Can Modify State	onlyRole
withdrawStake	External	Can Modify State	onlyRole
addStake	External	Payable	onlyRole
unlockStake	External	Can Modify State	onlyRole
_authorizeLog	Internal	-	-

BaseAccount			
Function Name	Visibility	Mutability	Modifiers
getNonce	Public	-	-
entryPoint	Public	-	-
validateUserOp	External	Can Modify State	-
_requireFromEntryPoint	Internal	-	-
_validateSignature	Internal	Can Modify State	-
_validateNonce	Internal	-	-

BaseAccount			
_payPrefund	Internal	Can Modify State	-

BasePaymaster			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
validatePaymasterUserOp	External	Can Modify State	-
_validatePaymasterUserOp	Internal	Can Modify State	-
postOp	External	Can Modify State	-
_postOp	Internal	Can Modify State	-
deposit	Public	Payable	-
withdrawTo	Public	Can Modify State	onlyRole
addStake	External	Payable	onlyRole
getDeposit	Public	-	-
unlockStake	External	Can Modify State	onlyRole
withdrawStake	External	Can Modify State	onlyRole
_requireFromEntryPoint	Internal	Can Modify State	-

GasInBox			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
nativeTopUp	External	Payable	checkToken
erc20TopUp	External	Payable	checkToken
erc20PermitTopUp	External	Payable	checkToken

GasInBox

_increaseGas	Internal	Can Modify State	-
withdraw	External	Can Modify State	onlyRole
whitelistToken	External	Can Modify State	onlyRole
getBalance	External	-	-

VerifyingPaymaster

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BasePaymaster
getHash	Public	-	-
_validatePaymasterUserOp	Internal	-	-
_validateCallData	Internal	-	-
parsePaymasterAndData	Public	-	-

VerifyingSingletonPaymaster

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Payable	BasePaymaster
depositFor	External	Payable	nonReentrant
getBalance	External	-	-
deposit	Public	Payable	-
withdrawTo	Public	Can Modify State	nonReentrant
setUnaccountedEPGasOverhead	External	Can Modify State	onlyRole
getHash	Public	-	-
_validatePaymasterUserOp	Internal	-	-

VerifyingSingletonPaymaster			
parsePaymasterAndData	Public	-	-
getGasPrice	Internal	-	-
min	Internal	-	-
_postOp	Internal	Can Modify State	-

WhitelistOperationVerifyingPaymaster			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	VerifyingPaymaster
whitelistOperation	External	Can Modify State	onlyRole
unWhitelistOperation	External	Can Modify State	onlyRole
switchWhitelistOperation	External	Can Modify State	onlyRole
getAllOperationWhitelisted	External	-	-
_validateCallData	Internal	-	-

OspAccount			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
version	External	-	-
entryPoint	Public	-	-
<Receive Ether>	External	Payable	-
initialize	External	Can Modify State	onlyProxy
getImplementation	External	-	-
isValidSignature	External	-	-

OspAccount			
setOwner	External	Can Modify State	-
setRecoveryAddress	External	Can Modify State	-
revokePendingRecoveryAddress	External	Can Modify State	-
recoverOwner	External	Can Modify State	-
setPermitCall	External	Can Modify State	-
revokeSessionKey	External	Can Modify State	-
_validateSignature	Internal	Can Modify State	-
_validateAuthorizeFromSessionKey	Internal	-	-
_validateAuthorize	Internal	-	-
_authorizeUpgrade	Internal	-	-
_authorizeStandardExecutor	Internal	-	-
_runtimeValidateIfNotFromEntrypoint	Internal	-	-
_runtimeValidateIfNotFromEntrypointOrAccount	Internal	-	-
getDeposit	Public	-	-
addDeposit	Public	Payable	-
withdrawDepositTo	External	Can Modify State	-

StandardExecutor			
Function Name	Visibility	Mutability	Modifiers
execute	Public	Payable	-
executeBatch	Public	Payable	-

StandardExecutor

_authorizeStandardExecutor	Internal	Can Modify State	-
_exec	Internal	Can Modify State	-

TokenCallbackHandler

Function Name	Visibility	Mutability	Modifiers
tokensReceived	External	-	-
onERC721Received	External	-	-
onERC1155Received	External	-	-
onERC1155BatchReceived	External	-	-
supportsInterface	External	-	-

ExecutionAuthorizer

Function Name	Visibility	Mutability	Modifiers
_getExecutionAuthorizerStorage	Internal	-	-
_setPermitCall	Internal	Can Modify State	-
isPermitCall	Public	-	-
permitCallsLength	External	-	-
permitCalls	External	-	-

MultiSigner

Function Name	Visibility	Mutability	Modifiers
_getMultiSignerStorage	Internal	-	-
_setOwner	Internal	Can Modify State	-
_revokePendingRecoveryAddress	Internal	Can Modify State	-

MultiSigner			
_setRecoveryAddress	Internal	Can Modify State	-
_recoverOwner	Internal	Can Modify State	-
isOwner	Public	-	-
ownersLength	Public	-	-
owners	External	-	-
recoveryAddress	Public	-	-
pendingRecoveryAddress	Public	-	-
lastChangeRecovery	Public	-	-
lastRecovery	Public	-	-

SessionKey			
Function Name	Visibility	Mutability	Modifiers
_getSessionKeyStorage	Internal	-	-
_revokeSessionKey	Internal	Can Modify State	-
_isRevokedSessionKey	Private	-	-
_hashSessionKey	Private	-	-
_verifySessionKey	Private	-	-
_getSessionKeyFromUserOperationSignature	Internal	-	-

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the VerifyingSingletonPaymaster contract, the `DEFAULT_ADMIN_ROLE` can arbitrarily set the `unaccountedEPGasOverhead` parameters. If this parameter is set too high, paymasterIdBalances may be consumed maliciously.

Code location:

paymaster/VerifyingSingletonPaymaster.sol#105-109

```
    function setUnaccountedEPGasOverhead(uint256 value) external
onlyRole(DEFAULT_ADMIN_ROLE) {
        uint256 oldValue = unaccountedEPGasOverhead;
        unaccountedEPGasOverhead = value;
        emit EPGasOverheadChanged(oldValue, value);
    }
```

2.In the WhitelistOperationVerifyingPaymaster contract, the `ADMIN` role can add the whitelist destination contracts and selectors for VerifyingPaymaster's PatmasterUserOp operation validation. And the `ADMIN` role can choose to switch to start or turn off this validation at any time.

Code location:

paymaster/WhitelistOperationPaymaster.sol#37-61

```
    function whitelistOperation(OperationInPut[] memory operations) external
onlyRole(ADMIN) {
        for (uint256 i = 0; i < operations.length; i++) {
            OperationInPut memory operation = operations[i];
            for (uint256 j = 0; j < operation.selectors.length; j++) {
                operationWhitelisted.add(
                    bytes32(bytes.concat(bytes20(operation.dest),
operation.selectors[j]))
                );
            }
        }
    }

    function unWhitelistOperation(OperationInPut[] memory operations) external
onlyRole(ADMIN) {
        for (uint256 i = 0; i < operations.length; i++) {
            OperationInPut memory operation = operations[i];
            for (uint256 j = 0; j < operation.selectors.length; j++) {
                operationWhitelisted.remove(
                    bytes32(bytes.concat(bytes20(operation.dest),
operation.selectors[j]))
```

```
                );
            }
        }
    }

    function switchWhitelistOperation(bool _enableWhitelistOperation) external
onlyRole(ADMIN) {
        enableWhitelistOperation = _enableWhitelistOperation;
    }
```

3.In the OspAccountFactory contract, the `DEFAULT_ADMIN_ROLE` can modify the `accountImplementation` contract address through the setAccountImplementation function and upgrade the contract by upgradeToAndCall function. Upgrading the contract and modifying the `accountImplementation` may introduce new risks.

Code location:

wallet/OspAccount.sol#367-373

wallet/OspAccountFactory.sol#85-89

```
    function setAccountImplementation(
        address newImplementation
    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
        accountImplementation = newImplementation;
    }

    function _authorizeUpgrade(address newImplementation) internal view override {
        _runtimeValidateIfNotFromEntrypointOrAccount();
        require(
            newImplementation ==
IAccountFactory(_accountFactory).accountImplementation(),
            'account: newImplementation is illegality'
        );
    }
```

Solution

In the short term, transferring owner ownership to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. The authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds.

Status

Acknowledged; After communicating with the project team, they split the DEFAULT_ADMIN_ROLE permission between the ADMIN role and the CASHIER role, and the ADMIN role and CASHIER role will be transferred to the safe multisig wallet control.

[N2] [Suggestion] Ordinary users with permitCalls may use the owner privilege through arbitrary contract calls

Category: Design Logic Audit

Content

The execute and executeBatch functions allow ordinary users to call the contracts which in the permitCalls with arbitrary calldata, can control their native token or ERC20 token assets by these functions. However, this functionality can be abused and users calling the contract itself (OspAccount) through the EntryPoint contract will then be able to call functions like setOwner, setRecoveryAddress, setPermitCall, and revokeSessionKey functions, thus overstepping their authority.

Code location:

wallet/OspAccount.sol

wallet/extensions/StandardExecutor.sol

```
    function execute(
        Execution calldata execution
    ) public payable virtual returns (bytes memory result) {
        _authorizeStandardExecutor();
        result = _exec(execution.target, execution.value, execution.data);
    }

    function executeBatch(
        Execution[] calldata executions
    ) public payable virtual returns (bytes[] memory results) {
        _authorizeStandardExecutor();
        …
        for (uint256 i = 0; i < executionsLength; ) {
            results[i] = _exec(executions[i].target, executions[i].value,
executions[i].data);
            …
        }
    }
    function _authorizeStandardExecutor() internal virtual;

    function _exec(
        …
    ) internal returns (bytes memory result) {
        bool success;
        (success, result) = target.call{value: value}(data);

        if (!success) {
            …
        }
    }

    function _runtimeValidateIfNotFromEntrypoint() internal view {
        if (msg.sender != address(entryPoint())) {
            require(
                _validateAuthorize(msg.data, msg.sender),
                'account: runtimeValidateAuthorize fail'
            );
        }
    }

    function _runtimeValidateIfNotFromEntrypointOrAccount() internal view {
        if (!(msg.sender == address(entryPoint()) || msg.sender == address(this))) {
            require(
                _validateAuthorize(msg.data, msg.sender),
                'account: runtimeValidateAuthorize fail'
            );
        }
    }

    function _validateAuthorize(
        bytes calldata callData,
        address signer
    ) internal view returns (bool) {
        // Get the selector of the user op.
        bytes4 selector = bytes4(callData);
        …
        if (selector == IStandardExecutor.execute.selector) {
            // Decode the execution data.
            IStandardExecutor.Execution memory exec = abi.decode(
                callData[4:],
                (IStandardExecutor.Execution)
            );

            // Check if the call is authorized.
            if (!isPermitCall(exec.target, signer)) {
                return false;
            }
            return true;
            // If the selector is the executeBatch function, then validate the
executions.
        } else if (selector == IStandardExecutor.executeBatch.selector) {
            // Decode the execution data.
            IStandardExecutor.Execution[] memory execs = abi.decode(
                callData[4:],
                (IStandardExecutor.Execution[])
            );
            uint length = execs.length;
            for (uint i; i < length; ) {
                IStandardExecutor.Execution memory exec = execs[i];
                // Check if the call is authorized.
                if (!isPermitCall(exec.target, signer)) {
                    return false;
                }
                unchecked {
                    ++i;
                }
            }
            return true;
        }
        return false;
    }
```

Solution

It is recommended to strictly confirm the permitCalls authorization and perform permission checks on authentication that can be bypassed by `address(this)`.

Status

Fixed

[N3] [Suggestion] Missing the event records

Category: Others

Content

In the WhitelistOperationVerifyingPaymaster and the OspAccountFactory contracts, the `ADMIN` and `DEFAULT_ADMIN_ROLE` can arbitrarily modify `OperationInPut`, `enableWhitelistOperation`, and `accountImplementation` parameters, but there are no event logs in these functions.

Code location:

paymaster/WhitelistOperationPaymaster.sol#37-61

```
    function whitelistOperation(OperationInPut[] memory operations) external 
onlyRole(ADMIN) {
        for (uint256 i = 0; i < operations.length; i++) {
            OperationInPut memory operation = operations[i];
            for (uint256 j = 0; j < operation.selectors.length; j++) {
                operationWhitelisted.add(
                    bytes32(bytes.concat(bytes20(operation.dest), 
operation.selectors[j]))
                );
            }
        }
    }

    function unWhitelistOperation(OperationInPut[] memory operations) external 
onlyRole(ADMIN) {
        for (uint256 i = 0; i < operations.length; i++) {
            OperationInPut memory operation = operations[i];
            for (uint256 j = 0; j < operation.selectors.length; j++) {
                operationWhitelisted.remove(
                    bytes32(bytes.concat(bytes20(operation.dest), 
operation.selectors[j]))
                );
            }
        }
    }

    function switchWhitelistOperation(bool _enableWhitelistOperation) external 
onlyRole(ADMIN) {
        enableWhitelistOperation = _enableWhitelistOperation;
    }
```

Solution

It is recommended to record events when sensitive parameters are modified for self-inspection or community review.

Status

Fixed

[N4] [Suggestion] Missing zero address validation

Category: Others

Content

In the OspAccount and the OspAccountFactory contracts, it lacks a zero-check when setting addresses.

Code location:

wallet/OspAccount.sol#69-78

wallet/OspAccountFactory.sol#30-34

```
    constructor(address admin, address aEntrypoint) {
        accountImplementation = address(new OspAccount(IEntryPoint(aEntrypoint),
address(this)));
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        entryPoint = IEntryPoint(payable(aEntrypoint));
    }

    function initialize(address anOwner) external onlyProxy {
        require(msg.sender == _accountFactory);
        _setOwner(anOwner, true);
        ...
    }
```

Solution

It is recommended to add zero address validation.

Status

Fixed

[N5] [Suggestion] ERC777 reentrancy risk reminder

Category: Unsafe External Call Audit

Content

ERC777 tokens are vulnerable to reentrancy attacks due to a design flaw. In the TokenCallbackHandler contract, the deprecated ERC777 standard tokensReceived has been introduced into the contract. If there is any need to deal with ERC77 tokens in the project, strict attention needs to be paid to whether there is reentrancy risk.

Code location:

wallet/extensions/TokenCallbackHandler.sol#9, 16-23

```
import {IERC777Recipient} from
'@openzeppelin/contracts/interfaces/IERC777Recipient.sol';
    function tokensReceived(
        address,
        address,
        address,
        uint256,
        bytes calldata,
        bytes calldata
    ) external pure override {}
```

Solution

It is recommended to strictly check or discard the ERC777 standard when using it.

Status

Fixed; Removed the tokensReceived hook and IERC777Recipient from the TokenCallbackHandler contract.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002403290001	SlowMist Security Team	2024.03.22 - 2024.03.29	Medium 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 1 medium risk, and 4 suggestions. 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