Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.03.31, the SlowMist security team received the UXUY Protocol team's security audit application for UXUY Smart 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 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

UxuyWallet is a smart contract wallet system based on a simplified ERC4337 standard that allows users to create and manage non-custodial smart contract wallets, supporting batch transaction execution, meta-transaction processing, and gas fee sponsorship functionality. The system employs an upgradeable architecture, enabling users to flexibly execute transactions, delegate administrators (when authorized) to perform operations on their behalf, set time limits for administrator authorizations, and manage wallet ownership changes; simultaneously, through proxy contracts and a logic registry structure, it ensures wallet functionality can be upgraded at any time without asset migration, providing users with a convenient blockchain asset management solution, particularly suitable for blockchain application scenarios that require simplified user experiences and enhanced transaction flexibility.

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	Fixed
N2	Admin Authorization Bypass Enables Unauthorized Fund Draining	Design Logic Audit	Low	Acknowledged
N3	Redundant Computation of finalSalt	Others	Suggestion	Fixed
N4	Failure to follow the Checks-Effects-Interactions principle	Reentrancy Vulnerability	Suggestion	Fixed
N5	call() should be used instead of transfer()	Others	Information	Fixed
N6	Redundant ECDSA Implementation	Others	Information	Fixed
N7	Multiple Pending Owner Changes Possible	Others	Suggestion	Acknowledged
N8	Inconsistent Sender Address in Event Emission	Malicious Event Log Audit	Information	Fixed
N9	Missing ChainId in Signature Verification	Replay Vulnerability	Information	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

https://bscscan.com/address/0x188042d7a332D34bE995f9084bE0bEBda2567c8a

https://bscscan.com/address/0xE54B7fE7b1058BD21c5Dc237853707787f6C881a

https://bscscan.com/address/0x10743E5546f3bFa2cFEEb7c5E4E06866EBD6BA5D

Fixed Version:

https://github.com/uxuycom/smart_wallet_contracts

commit: 29f91a561bcef631d41b14c1f4c3d5f13454840b

Audit Scope:

```
./contracts
├── UxuySmartEntry.sol
├── UxuySmartLogic.sol
├── UxuySmartLogicRegistry.sol
├── UxuySmartProxy.sol
└── libs
    ├── IUxuySmartLogic.sol
    └── IUxuySmartLogicRegistry.sol
```

The main network address of the contract is as follows:

UxuyWalletEntry	
Contract Name	Contract Address
UxuySmartEntry	0x743CB7d6D8fBBF93806DeC9B7700743a2641dae2
UxuySmartLogic	0x68F2A9688bCfA153939d15E593Ac675B466A8146
UxuySmartLogicRegistry	0x9C92F675319Db2305d3d3586B9f6094747Db4B44

4.2 Visibility Description

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

UxuyWalletLogicRegistry			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
setDefaultLogic	External	Can Modify State	onlyOwner

UxuyWalletLogicRegistry			
getDefaultLogic	External	-	-
addLogic	Public	Can Modify State	onlyOwner
removeLogic	External	Can Modify State	onlyOwner
isAllowedLogic	External	-	-

UxuyWalletProxy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC1967Proxy

UxuyWalletLogic			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	initializer
pause	External	Can Modify State	onlyAdmin
unpause	External	Can Modify State	onlyAdmin
changeLogicMap	External	Can Modify State	whenNotPaused onlyAdmin
changeAdmin	External	Can Modify State	whenNotPaused onlyOwner
changeOwner	External	Can Modify State	whenNotPaused onlyOwner
confirmChangeOwner	External	Can Modify State	whenNotPaused onlyOwner
cancelChangeOwner	External	Can Modify State	whenNotPaused onlyOwner
_authorizeUpgrade	Internal	-	onlyOwner

UxuyWalletLogic			
upgradeTo	External	Can Modify State	whenNotPaused onlyOwner
executeBatch	External	Can Modify State	whenNotPaused onlyOwner nonReentrant
executeBatchByAdmin	External	Can Modify State	whenNotPaused onlyAdmin
executeBatchByEntry	External	Can Modify State	whenNotPaused onlyEntry
_deductGasFee	Internal	Can Modify State	-
setAdminAuthorization	External	Can Modify State	whenNotPaused onlyOwner
_call	Internal	Can Modify State	-
_recoverSigner	Internal	-	-
_splitSignature	Internal	-	-
<Receive Ether>	External	Payable	-

UxuyWalletEntry			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner
addPlatformAdmins	External	Can Modify State	onlyOwner
removePlatformAdmins	External	Can Modify State	onlyOwner
getAddress	Public	-	-
createAccount	Public	Can Modify State	whenNotPaused
execute	External	Can Modify State	whenNotPaused nonReentrant

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the UxuyWalletLogicRegistry contract, the owner is an EOA address. The owner role can modify the defaultLogic address and add other logic addresses through the setDefaultLogic and addLogic functions. The above modifications will affect the creation of new UxuyWallet and the subsequent contract updates of the original UxuyWallet.

Code location:

UxuyWalletLogicRegistry.sol#L19-L23, L29-L35

```
function setDefaultLogic(address logic) external onlyOwner {
     addLogic(logic);
     defaultLogic = logic;
     emit DefaultLogicUpdated(logic);
 }

 function addLogic(address logic) public onlyOwner {
     require(logic != address(0), "Invalid logic address");
     require(!allowedLogics[logic], "Logic already added");

     allowedLogics[logic] = true;
     emit LogicAdded(logic);
 }
```

2.In the UxuyWalletEntry contract, the owner is an EOA address, which is the same as the UxuyWalletLogicRegistry contract. The owner role can add or remove the platformAdmins through the addPlatformAdmins and removePlatformAdmins functions. The platformAdmins role can create UxuyWallet for the specified user through the execute function and become the admin role of the wallet. It can also call the executeBatchByEntry function of the wallet through the execute function to perform specified operations, or directly perform operations on UxuyWallet through the executeBatchByAdmin function.

Code location:

UxuyWalletEntry.sol#L86-L104

```
    function execute(IUxuyWalletLogic.ExecuteParams calldata params) external
whenNotPaused nonReentrant {
        address sender = params.sender;
        require(sender != address(0), "Invalid sender address");

        bool created = false;

        if (params.sender.code.length == 0) {
            sender = createAccount(params.user, params.salt, msg.sender);
            created = true;
        }

        if (params.dest.length > 0){
            require(platformAdmins[msg.sender] == true, "Not authorized platform
admin");

            IUxuyWalletLogic(sender).executeBatchByEntry(params);

            emit TransactionExecuted(params.user, params.sender, msg.sender, created);
        }
    }
```

Solution

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

Status

Fixed; After deployed, the Uxuy smart contract wallet system implements a rigorous multi-layered permission architecture, delegating core system control to a multisig wallet (0xEcc34657c79dcd4Ec6C986E8572e69E8e2a473E9) jointly managed by 3 EOA addresses, which controls ownership of both UxuySmartEntry and UxuySmartLogicRegistry contracts, ensuring that system upgrades and critical parameter changes; meanwhile, at the operational level, the system has designated 10 EOA addresses as Platform Admins through transaction 0xa1a5c824ff7b66e54b1c6655868e0897d9ec176e1e41e1c55649b3e5fbd104f8, these administrators can assist in

executing daily operations, but their authority is strictly limited by smart contract logic, requiring user signature authorization to execute asset-related transactions, thus creating a governance model that balances decentralization with operational efficiency. The project team stated that the Admin can pay Gas on behalf of users, but the user's signature is required for confirmation. Admin cannot transfer assets without user confirmation.

[N2] [Low] Admin Authorization Bypass Enables Unauthorized Fund Draining

Category: Design Logic Audit

Content

In the UxuyWalletLogic contract, the executeBatchByAdmin function allows an admin to execute transactions on behalf of the wallet owner. However, when adminAuthorizedUntil is set (which can be up to 30 days), an admin can bypass signature verification entirely and construct arbitrary transactions to drain all assets from the wallet. This is particularly dangerous because the admin can use the _deductGasFee function to transfer any token to any address without the owner's consent.

Code location:

UxuyWalletLogic.sol#L140-L173, L220-L230

```
   function executeBatchByAdmin(ExecuteParams calldata params) external whenNotPaused
onlyAdmin{
        ...
        bool isAdminAuthorized = block.timestamp <= adminAuthorizedUntil;

        if (!isAdminAuthorized) {
            bytes32 messageHash = keccak256(abi.encode(params.dest, params.value,
params.func, params.gasToken, params.gasFee, params.userNonce, address(this)));
            bytes32 ethSignedMessageHash = keccak256(abi.encodePacked("\x19Ethereum
Signed Message:\n32", messageHash));
            address recoveredSigner = _recoverSigner(ethSignedMessageHash,
params.ownerSignature);
            require(recoveredSigner == owner, "Invalid user signature");
        }

        if (params.gasDeductBefore) {
            _deductGasFee(params.gasToken, params.gasFee, params.gasReceive);
        }
        ...
        if (!params.gasDeductBefore) {
            _deductGasFee(params.gasToken, params.gasFee, params.gasReceive);
        }
```

```
        ...
    }

    function setAdminAuthorization(uint256 daysDuration) external whenNotPaused
onlyOwner {
        require(daysDuration <= MAX_ADMIN_AUTH_TIME, "Exceeds max authorization
time");

        if (daysDuration <= 0){
            adminAuthorizedUntil = 0;
            emit AdminAuthorizationUpdated(false, 0);
        } else {
            adminAuthorizedUntil = block.timestamp + (daysDuration * 1 days);
            emit AdminAuthorizationUpdated(true, adminAuthorizedUntil);
        }
    }
```

Solution

It's recommended to ensure all admin actions require explicit owner approval or be limited to non-fund-moving operations.

Status

Acknowledged; After communicating with the project team, they stated that this is a functional requirement. In order to enable the admin to pay for gas, the gas cost paid needs to be signed and confirmed by the user. The authorization function is to meet the needs of the admin to trade on behalf of the user. This will remind the user on the user side, and the authorization can be closed at any time.

[N3] [Suggestion] Redundant Computation of finalSalt

Category: Others

Content

In the UxuyWalletEntry contract, the same computation for finalSalt is performed in both the getAddress and createAccount functions. The finalSalt is calculated using keccak256 with the same parameters in both locations, which is inefficient and causes unnecessary gas consumption.

Code location:

UxuyWalletEntry.sol#L53-L84

```
    function getAddress(address user, uint256 salt, address admin) public view returns
(address) {
```

```
        bytes32 finalSalt = keccak256(abi.encodePacked(user, salt));
        …
    }

    function createAccount(address user, uint256 salt, address admin) public
whenNotPaused returns (address scw) {
        …
        address logicImpl = IUxuyWalletLogicRegistry(logicMapAddr).getDefaultLogic();
        bytes32 finalSalt = keccak256(abi.encodePacked(user, salt));
        UxuyWalletProxy proxy = new UxuyWalletProxy{salt: finalSalt}(
            logicImpl,
            abi.encodeCall(UxuyWalletLogic.initialize, (user, admin, address(this),
logicMapAddr))
        );
        …
    }
```

Solution

It is recommended to refactor the code to avoid redundant computation by having createAccount call getAddress to obtain the address, and then extract the finalSalt calculation into a separate internal function that both methods can use when needed.

Status

Fixed

[N4] [Suggestion] Failure to follow the Checks-Effects-Interactions principle

Category: Reentrancy Vulnerability

Content

In the UxuyWalletLogic contract, both executeBatchByAdmin and executeBatchByEntry functions violate the Checks-Effects-Interactions (CEI) pattern when params.gasDeductBefore is true. The functions first make an external call through _deductGasFee and only afterward increment the nonce, which is a state change that should occur before any external interactions. Code location:

UxuyWalletLogic.sol#L140-L205

```
   function executeBatchByAdmin(ExecuteParams calldata params) external whenNotPaused
onlyAdmin{
        …
        //gas
        if (params.gasDeductBefore) {
```

```
            _deductGasFee(params.gasToken, params.gasFee, params.gasReceive);
        }

        uint256 _nonce = nonce;
        nonce = _nonce + 1;
        …
    }

    function executeBatchByEntry(ExecuteParams calldata params) external
whenNotPaused onlyEntry{
        …
        if (params.gasDeductBefore) {
            _deductGasFee(params.gasToken, params.gasFee, params.gasReceive);
        }

        uint256 _nonce = nonce;
        nonce = _nonce + 1;

        …
    }
```

Solution

It is recommended to follow the CEI pattern by moving the nonce increment before any external calls.

Status

Fixed

[N5] [Information] call() should be used instead of transfer()

Category: Others

Content

The transfer() and send() functions forward a fixed amount of 2300 gas. Historically, it has often been recommended to use these functions for value transfers to guard against reentrancy attacks. However, the gas cost of EVM instructions may change significantly during hard forks which may break already deployed contract systems that make fixed assumptions about gas costs. For example. EIP 1884 broke several existing smart contracts due to a cost increase of the SLOAD instruction.

Code location:

UxuyWalletLogic.sol#L207-L218

```
    function _deductGasFee(address gasToken, uint256 gasFee, address gasReceive)
internal {
        if (gasFee > 0){
            if (gasToken == address(0)) {
                require(address(this).balance >= gasFee, "ETH balance not enough");
                payable(gasReceive).transfer(gasFee);
            } else {
                …
            }
            emit GasReceived(address(this), gasReceive, gasToken, gasFee);
        }
    }
```

Solution

It is recommended to use call() instead of transfer(), but be sure to respect the CEI pattern or add re-entrancy guards, and the return value should be checked. It also needs to refer to the N4 solution.

Status

Fixed

[N6] [Information] Redundant ECDSA Implementation

Category: Others

Content

In the UxuyWalletLogic contract, the OpenZeppelin ECDSA library is imported but not utilized. Instead, the contract implements its own signature verification logic with _recoverSigner and _splitSignature functions. This creates code duplication, increases gas costs instead of using the OpenZeppelin implementation.

Code location:

UxuyWalletLogic.sol#L5-L6

```
import "@openzeppelin/contracts/utils/cryptography/ECDSA.sol";
import "@openzeppelin/contracts/utils/cryptography/MessageHashUtils.sol";
```

Solution

It is recommended to replace the custom signature verification logic with the OpenZeppelin ECDSA library functions.

Status

Fixed

[N7] [Suggestion] Multiple Pending Owner Changes Possible

Category: Others

Content

In the UxuyWalletLogic contract, the changeOwner function allows the current owner to set multiple pending ownership transfers simultaneously without invalidating previous requests. Each new ownership transfer request is stored in the pendingOwnerChange mapping with a 7-day time lock. However, since there is no mechanism to cancel previous pending transfers immediately when creating a new one, this could lead to confusion or forgotten and potentially unintended ownership transfers when calling changeOwner multiple times, adding different pendingOwnerChanges.

Code location:

UxuyWalletLogic.sol#L91-108

```
    function changeOwner(address newOwner) external whenNotPaused onlyOwner {
        require(newOwner != address(0), "Invalid new owner");
        emit OwnerChangePending(owner, newOwner);
        pendingOwnerChange[newOwner] = block.timestamp + 7 days;
    }

    function confirmChangeOwner(address newOwner) external whenNotPaused onlyOwner {
        require(pendingOwnerChange[newOwner] != 0, "No pending owner change");
        require(block.timestamp >= pendingOwnerChange[newOwner], "Change not yet
effective");
        emit OwnerChanged(owner, newOwner);
        owner = newOwner;
        delete pendingOwnerChange[newOwner];
    }

    function cancelChangeOwner(address newOwner) external whenNotPaused onlyOwner {
        require(pendingOwnerChange[newOwner] != 0, "No pending change");
        delete pendingOwnerChange[newOwner];
    }
```

Solution

It is recommended to implement a mechanism that ensures only one pending ownership transfer can be active at any time.

Status

Acknowledged

[N8] [Information] Inconsistent Sender Address in Event Emission

Category: Malicious Event Log Audit

Content

In the UxuyWalletEntry contract, when a new account is created in the execute function, the sender variable is properly updated to reference the newly created account. However, when emitting the TransactionExecuted event, the code incorrectly uses the original params.sender rather than the updated sender value. This inconsistency may cause confusion when interpreting event logs, as the logged sender address could be different from the actual address used for transaction execution.

Code location:

UxuyWalletEntry.sol#L86-L104

```
    function execute(IUxuyWalletLogic.ExecuteParams calldata ) external whenNotPaused
nonReentrant {
        address sender = params.sender;
        require(sender != address(0), "Invalid sender address");

        bool created = false;

        if (params.sender.code.length == 0) {
            sender = createAccount(params.user, params.salt, msg.sender);
            created = true;
        }

        if (params.dest.length > 0){
            ...;
            emit TransactionExecuted(params.user, params.sender, msg.sender, created);
        }
    }
```

Solution

It is recommended to use the updated sender variable in the event emission to maintain consistency.

Status

Fixed

[N9] [Information] Missing ChainId in Signature Verification

Category: Replay Vulnerability

Content

In the UxuyWalletLogic contract, the signature verification process does not include the chain ID in the message hash construction. This omission could potentially enable cross-chain replay attacks if the same contract is deployed on multiple EVM-compatible blockchains. An attacker could take a valid signature from one chain and replay it on another chain where the same wallet contract exists.

Code location:

UxuyWalletLogic.sol#L148-L151, L181-L184

```
        bytes32 messageHash = keccak256(abi.encode(params.dest, params.value,
params.func, params.gasToken, params.gasFee, params.userNonce, address(this)));
         bytes32 ethSignedMessageHash = keccak256(abi.encodePacked("\x19Ethereum Signed
Message:\n32", messageHash));
         address recoveredSigner = _recoverSigner(ethSignedMessageHash,
params.ownerSignature);
         require(recoveredSigner == owner, "Invalid user signature");
```

Solution

It is recommended to include the chain ID in the message hash to prevent cross-chain replay attacks.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002504010002	SlowMist Security Team	2025.03.31 - 2025.04.01	Low Risk

Summary conclusion: The SlowMist security team uses a manual and the SlowMist team's analysis tool to audit the project. During the audit work, we found 1 medium risk, 1 low risk, 2 suggestions, and 5 information. All the findings were fixed or acknowledged. The project has been successfully deployed to the mainnet. The ownership of

UxuySmartEntry and UxuySmartLogicRegistry contracts has been transferred to a 2-of-3 multisig wallet (0xEcc34657c79dcd4Ec6C986E8572e69E8e2a473E9), and 10 EOA addresses have been properly configured as Platform Admins through transaction 0xa1a5c824ff7b66e54b1c6655868e0897d9ec176e1e41e1c55649b3e5fbd104f8. Due to this robust permission management structure and successful transfer of core role permissions, the risk assessment has been updated to low risk.

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.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist