0xCommit

Security Assessment Report

Biconomy - Smart Wallet Contracts - Multichain Module

Version: Final

Date: 3 Oct 2023

Table of Contents

Licence

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

Introduction

Purpose of this report

0xCommit (previously Kawach) has been engaged by **Biconomy** to perform a security audit of several Smart Account components.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behaviour.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebases Submitted for the Audit

The audit has been performed on the following GitHub repositories:

Repository :
https://github.com/bcnmy/scw-contracts/blob/SCW-V2-Modular-SA/contracts/smart-account/modules/MultichainECDSAValidator.sol

Version	Commit hash
Initial Codebase	d45733e04253d3c19ac0a12c180187b4df24d5c6

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
High	An attacker can successfully execute an attack that clearly results in operational issues for the service. This also includes any value loss of unclaimed funds permanently or temporary.
Medium	The service may be susceptible to an attacker carrying out an unintentional action, which could potentially disrupt its operation. Nonetheless, certain limitations exist that make it difficult for the attack to be successful.
Low	The service may be vulnerable to an attacker executing an unintended action, but the impact of the action is negligible or the likelihood of the attack succeeding is very low and there is no loss of value.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary

The status of an issue can be one of the following: **Pending**, **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Overview

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
    d. Access Control Issues
    e. Boundary Analysis
4. Report preparation

Functionality Overview

Biconomy is an extensible Account Abstraction protocol, which allows users to create Smart wallets.

Summary of Findings

Sr. No.	Description	Severity	Status
1	Replication of the merkle tree can lead to arbitrage opportunities for bots	Low	Resolved
2	Generation of multiple isolated Merkle trees	Informatio...	Acknowledged

Detailed Findings

1. Replication of the merkle tree can lead to arbitrage opportunities for bots

Severity: Low

Description

Multichain Module uses Merkle trees which are computed off chain. UserOps for each chain are added as leaves to the merkle tree, and the root of the tree is signed and used as the signature for on-chain verification. User will then send the specific userOp with the respective merkle tree proof and root's signature to the chains. Here the signatures on each chain will remain the same as it is the same merkle tree's root.

Here the issue arises if there is a pattern in the user's smart wallet transactions, and an attacker picks up the pattern. Then they can possibly replicate the merkle tree on their own and relay the user's transactions on other chains even before the user himself. Through this the attacker can do an arbitrage like a sandwich attack very easily.

Remediation

The offchain merkle trees can include some random leaves which are never published on-chain making it impossible for an attacker to replicate the merkle tree. Another solution can be to introduce pseudo-random validUntil/validAfter for every userOp signature, which makes it difficult to predict the pattern for an attacker.

Status

Acknowledged

2. Generation of multiple isolated Merkle trees

In multichain Module there is no hard enforcement of a singular Merkle tree for each account, thereby leading to creation of multiple Merkle trees leaving room for attack vectors for malicious actors.

Severity: Informational

Category: Exposed method or function

Description

Multichain Module uses Merkle trees which are computed off chain. However there is no data component of the Merkle trees kept on chain leading to existence of multiple valid Merkle trees being maintained off chain. (i.e. - Each transaction can be associated with a different or a new Merkle tree). This opens up an attack vector for the account where a malicious actor can compromise the account using social engineering means.

Vulnerable Code

Unset

```
// SessionKeyManagerModule.sol#L66-L91 - Key area of issue.
        (
            uint48 validUntil,
            uint48 validAfter,
            bytes32 merkleTreeRoot,
            bytes32[] memory merkleProof, <- Vulnerable code
            bytes memory multichainSignature
        ) = abi.decode(
                moduleSignature,
                (uint48, uint48, bytes32, bytes32[], bytes)
            );

        //make a leaf out of userOpHash, validUntil and validAfter
        bytes32 leaf = keccak256(
            abi.encodePacked(validUntil, validAfter, userOpHash)
        );

        if (!MerkleProof.verify(merkleProof, merkleTreeRoot, leaf)) {
            revert("Invalid UserOp");
        }
```

Remediation

For remediation the following are the action points.

- Implement incremental Merkle trees for each account, such that each account has a single Merkle tree and Merkle root for each account is maintained on chain.

(Refer - @zk-kit for standardised implementation of incremental Merkle trees)

Status

Acknowledged