0xCommit

Security Assessment Report

Biconomy - Smart Wallet Contracts

Version: Final

Date: 18 Jul 2023

Table of Contents

Licence

Disclaimer

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/tree/SCW-V2-Modular-SA-Kawach-Audit

Date	Version	Commit hash
26/06/2023	Initial Codebase	872dd4b1a9010a83e51ebb491b680c840ead83d6

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	Un-restricted use of account by session keys - (Design Issue)	High	Resolved

Detailed Findings

Un-restricted use of account by session keys - Design Issue

Session keys are granted to the session validation module which can be built by Dapps for application specific userOperation via session key manager module. In the current implementation, the session key manager module checks if the session key and session data is valid by comparing it to the merkle root for the respective account. However session data is not derived from userOperation which allows session keys to access functions which are out of bound for session keys.

Severity: **High**

Category: Privilege Escalation / Design Issue

Description

Session keys once issued have unlimited power on smart accounts. Since the session key manager module does not enforce strict restrictions on access to functionality of the smart accounts. In the current scheme of things, a session key is associated with session data. Session key manager module validates session data with the merkle root of the respective account. Session data is passed externally along with userOperation (userOperation is one which gets executed). Current validation of session data does not include data derived from userOperation. This can lead to a malicious Session validation module to access more functions than it is permitted. In such a situation the attacker can gain access to the following core functionality of the smart Account but not limited to it.

- Transfer Owner
- Renounce Owner
- Set Merkle Root
- Enabling Module
- Disabling Module
- Update Implementation

Also EIP4337 Specification requires strict control on validateUserOps to disallow **arbitrary account hijacking**.

Vulnerable Code

Unset

```
// SessionKeyManagerModule.sol#L66-L91 - Key area of issue.
               bytes32 leaf = keccak256(
                   abi.encodePacked(
                       validUntil,
                       validAfter,
                       sessionValidationModule,
                       sessionKeyData // <- Actual vulnerable code as it does not take
information from userOp.callData
                   )
               );
               if (
                   !MerkleProof.verify(merkleProof, sessionKeyStorage.merkleRoot, leaf)
               ) {
                   revert("SessionNotApproved");
               }
               return
                   _packValidationData(
             //_packValidationData expects true if sig validation has failed, false otherwise
                       !ISessionValidationModule(sessionValidationModule)
                           .validateSessionUserOp(
                               userOp,
                               userOpHash,
                               sessionKeyData,
                               sessionKeySignature
                           ),
                       validUntil,
                       validAfter
                   );
```

Attack / Threat Scenarios

Removal of Dapp Specific sessionKeyData Checks

In this attack scenario the attacker / malicious Dapp developer will update the Session Validation module which does signature verification but does not check any session data against userOperation and rather simply passes it through. In this situation the attacker can pass any userOperation and it will be executed. Thus the attacker can have full access to the account via the session key granted to him.

Refer to test cases "AttackSssionValidation_0.Module.specs.ts"

As for the formal threat specification it falls under - **Assertion redirect attacks** where an attacker can redirect the use of session keys to some other functionality which he does not have access to. (Refer - NIST 800-63C specification relating to digital identity federation methods.)

Root Cause Analysis

Principle of least privilege not applied to session keys

Applying the principle of least privilege is essential for session keys. By default, the access scope of a session key should be limited. The current access control mechanism applied in session key manager grants full access, which is not ideal. Instead, session keys should only be granted access to pre-specified addresses and functions. In other words, the access should be restricted to specific contracts and function selectors, ensuring that session keys have minimal privileges required for their intended purpose, and this data to be included in the merkle root of session manager.

Remediation

For remediation the following data points are to be included in leaf data apart from session key data.

- Type of call to be done. (i.e. Execute call)
- Address to be called.
- Selector to be called.

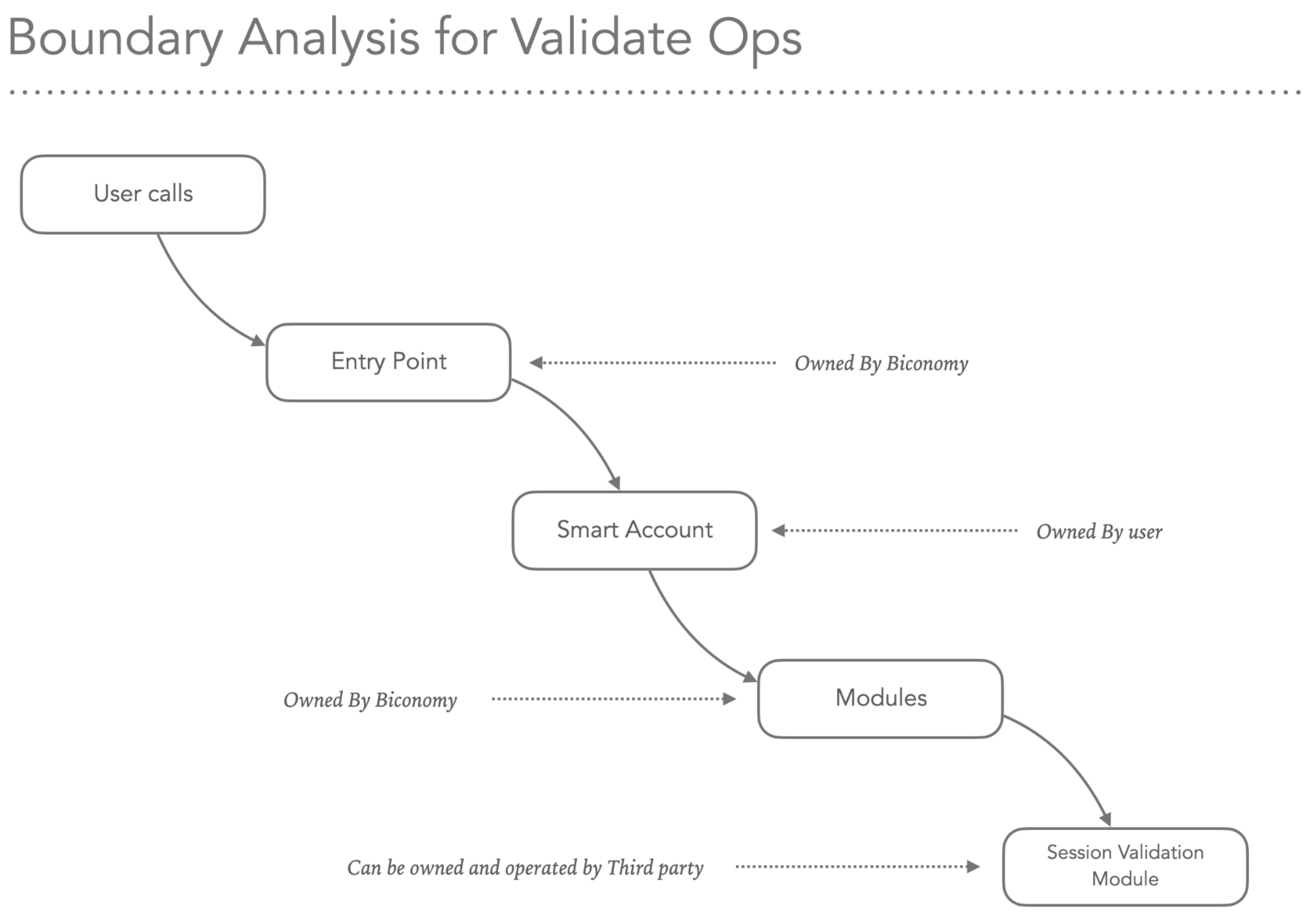
(Note :- The above information is to be extracted from userOp.calldata not from session key data, by this virtue the scope of access is limited by default)

Remediated code

Unset
```
// SessionKeyManagerModule.sol - Remediated code

       /**
       * @dev validates userOperation
       * @param userOp User Operation to be validated.
       * @param userOpHash Hash of the User Operation to be validated.
       * @return sigValidationResult 0 if signature is valid, SIG_VALIDATION_FAILED
otherwise.
       */
       function validateUserOp(
              UserOperation calldata userOp,
              bytes32 userOpHash
       ) external view virtual returns (uint256) {
              (bytes memory moduleSignature, ) = abi.decode(
                  userOp.signature,
                  (bytes, address)
              );




              (
                  uint48 validUntil,
                  uint48 validAfter,
                  address sessionValidationModule,
                  bytes memory sessionKeyData,
                  bytes32[] memory merkleProof,
                  bytes memory sessionKeySignature
              ) = abi.decode(
                      moduleSignature,
                      (uint48, uint48, address, bytes, bytes32[], bytes)
                  );
              ( address _caller , bytes4 _selector) = _extract( userOp.callData );

              bytes32 leaf = keccak256(
                  abi.encodePacked(
                      validUntil,
                      validAfter,
                      sessionValidationModule,
                      _extractSelector(userOp.callData),
                      _caller,
                      _selector
                  )
              );
              if (
                  !MerkleProof.verify(merkleProof, _getSessionData(msg.sender).merkleRoot,
leaf)
              ) {
                  revert("SessionNotApproved");
              }
              return
                  _packValidationData(
                      //_packValidationData expects true if sig validation has failed, false
otherwise
                      !ISessionValidationModule(sessionValidationModule)
                          .validateSessionUserOp(
                              userOp,
                              userOpHash,
                              sessionKeyData,
                              sessionKeySignature
                          ),
                      validUntil,
                      validAfter
                  );
          }

          // Note :- extracting _caller and _selector from userOp.callData this function can
be added into userOp library also
          function _extract(bytes calldata _calldata ) internal view returns ( address ,
bytes4) {
              bytes4 _selector;
              (address _caller, ,bytes memory _data ) = abi.decode(
                  _calldata[4:], // skip selector
                  (address, uint256, bytes)
              );
              _selector = _extractSelector(_data);
              return ( _caller , _selector );
          }

          function _extractSelector( bytes calldata _calldata ) internal view returns (
bytes4 ){
              bytes4 _selector;
              if ( _calldata.length >= 4 ) {
                  assembly {
                  // Load the first 4 bytes of 'data' into 'result'
                      _selector := mload(add(_calldata, 32))
                  }
              }
              return _selector;
          }
```

Ancillary Analysis

Boundary analysis


Boundary Analysis for Validate Ops
User calls
Entry Point
Owned By Biconomy
Smart Account
Owned By user
Owned By Biconomy
Modules
Can be owned and operated by Third party
Session Validation Module

References

https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63c.pdf - section 8.1 Federation threats.

Biconomy’s Remarks on Remediation

As per Biconomy there are two ways of mitigating this issue :

1) On-chain: add function selector(s) checks to the SessionKeyManager module.

2) Off-chain: ensure that no vulnerable Session Validation Module (SVM) can be enabled via Biconomy SDK.

Since Biconomy will be reconstructing the Merkle Tree when a new session key is added on our side, it can ensure that every SVM is coming from the list of pre-approved SVMs.

Of course, SA owner can always enable something not via SDK, but the same is currently true for regular Validation Modules.

Biconomy Assumes:

- At this stage, Biconomy wants to give more flexibility to the customers to implement various SVMs

- Biconomy is planning for the Modular Smart Account v3 with hooks support which will allow for a more efficient and reliable way of limiting Validation modules and SVMs permissions on-chain

- At this stage, all dApps most probably will use Biconomy backend infra to rebuild the Merkle Tree while adding/removing Session Keys

Based on these assumptions, future plans and current trajectory of events Biconomy concludes that for this version it will be more acceptable for us to go with the **2nd (off-chain)** way to mitigate this issue.

Real time functioning:

- Biconomy has a list of reviewed and approved SVMs

- When there is a request by a dApp for a new Merkle tree root to use it into the userOp which updated the root on-chain, Biconomy backend checks that all the leaves in the new Tree contain the SVM from the list above and thus are safe

- If there is a non-approved SVM Biconomy doesn’t provide a new root

Conclusion:

0xCommit (previously Kawach) is in agreement with the conclusions and remediation taken thereupon by Biconomy. Other than the issue mentioned in the report most of the modules in scope work as they are supposed to and thus the Biconomy team must be commended for the quality of the code.