Zellic

Biconomy Batched Session Router Module

Smart Contract Security Assessment

October 2, 2023

Prepared for:

Sachin Tomar, Chirag, and Aman

Biconomy Labs

Prepared by:

SeungHyeon Kim, Sina Pilehchiha, and Yuhang Wu

Zellic Inc.

Contents

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please contact us at hello@zellic.io.

1 Executive Summary

Zellic conducted a security assessment for Biconomy Labs from September 18th to September 22nd, 2023. During this engagement, Zellic reviewed Biconomy Batched Session Router Module's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.1 Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Could any potential bugs lead to passing the validation to the arbitrary Session Key Manager?
- Could any potential bugs lead to passing the validation to the arbitrary Session Validation Modules?
- Are there any potential vulnerabilities or incorrect implementations related to Merkle proof verification?
- Are there potential data-exposure risks, especially concerning session key data and parameters?
- Can a user potentially exploit the batching mechanism to execute unauthorized or malicious operations?
- Can the session data be tampered with, either before or during the operation-validation process?
- Does the contract handle the comparison of `validUntil` and `validAfter` timestamps in a safe and expected manner? Are there edge cases where session validation can be bypassed?

1.2 Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

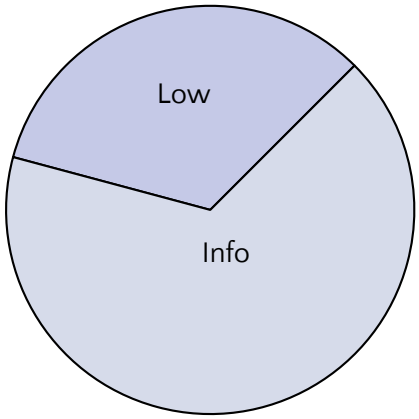
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.3 Results

During our assessment on the scoped Biconomy Batched Session Router Module modules, we discovered three findings. No critical issues were found. One finding was of low impact and the other findings were informational in nature.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	1
Informational	2



2 Introduction

2.1 About Biconomy Batched Session Router Module

Biconomy Batched Session Router Module allows the batching of several session-key-signed operations, which should be validated by different Session Validation modules into one User Operation, and executes them atomically.

2.2 Methodology

During a security assessment, Zellic works through standard phases of security auditing including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the modules.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality

standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped modules itself. These observations — found in the Discussion section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3 Scope

The engagement involved a review of the following targets:

Biconomy Batched Session Router Module Modules

Repository	https://github.com/bcnmy/scw-contracts
Version	scw-contracts: `3141c376c814ee7fcc7b0a8fb8f64c5a4aec9097`
Programs	SessionKeyManagerModule ERC20SessionValidationModule BatchedSessionRouterModule
Type	Solidity
Platform	EVM-compatible

2.4 Project Overview

Zellic was contracted to perform a security assessment with three consultants for a total of 10 person-days. The assessment was conducted over the course of five calendar days.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald, Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

SeungHyeon Kim, Engineer
seunghyeon@zellic.io

Sina Pilehchiha, Engineer
sina@zellic.io

Yuhang Wu, Engineer
yuhang@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

September 18, 2023	Kick-off call
September 18, 2023	Start of primary review period
September 22, 2023	End of primary review period

3 Detailed Findings

3.1 Add Length Validation for `callData` in `validateSessionUserOp`

- **Target**: ERC20SessionValidationModule
- **Category**: Coding Mistakes
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

In the `validateSessionUserOp` function, the length check for `op.callData` is incomplete, and there may be `callData` with illegal length.

```
function validateSessionUserOp(
    UserOperation calldata _op,
    bytes32 _userOpHash,
    bytes calldata _sessionKeyData,
    bytes calldata _sessionKeySignature
) external pure override returns (bool) {

    ...

    // working with userOp.callData
    // check if the call is to the allowed recepient and amount is not
more than allowed
    bytes calldata data;
    {
        uint256 offset
= uint256(bytes32(_op.callData[4 + 64:4 + 96]));
        uint256 length = uint256(
            bytes32(_op.callData[4 + offset:4 + offset + 32])
        );
        //we expect data to be the `IERC20.transfer(address, uint256)`
calldata
        data = _op.callData[4 + offset + 32:4 + offset + 32 + length];
    }
    if (address(bytes20(data[16:36])) ≠ recipient) {
        revert("ERC20SV Wrong Recipient");
```

```
        }
        if (uint256(bytes32(data[36:68])) > maxAmount) {
            revert("ERC20SV Max Amount Exceeded");
        }
        return
            ECDSA.recover(
                ECDSA.toEthSignedMessageHash(_userOpHash),
                _sessionKeySignature
            ) == sessionKey;
    }
```

Impact

Data without length restrictions may lead to issues such as hash collisions. A hash collision may lead to the ability to arbitrarily forge user messages.

Recommendations

Use `abi.decode` to get the message length and add a maximum length check on `op.callData`.

Remediation

This issue has been acknowledged by Biconomy Labs, and a fix was implemented in commit 3bf128e9.

3.2 Missing element count check of `sessionData` in `validateUserOp`

- **Target**: BatchedSessionRouter
- **Category**: Coding Mistakes
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

The function `validateUserOp` decodes an array named `sessionData` and iterates over it to perform various validations and computations. However, there is no explicit check in the code to ensure that the `sessionData` array contains at least one element.

```
 uint256 length = sessionData.length;
 (
        address sessionKeyManager,
        SessionData[] memory sessionData,
        bytes memory sessionKeySignature
) = abi.decode(moduleSignature, (address, SessionData[], bytes));
     ...

uint256 length = sessionData.length;

// iterate over batched operations
 for (uint i; i < length; ) {

      ...

}

return (
            _packValidationData(
                false, // sig validation failed = false; if we are here,
    it is valid
                earliestValidUntil,
                latestValidAfter
            )
);
```

Impact

The absence of a check for the array length could lead to potential logical errors or undesired behaviors in the case where the `sessionData` array is empty.

Recommendations

Implement the array length check and make sure the length of `sessionData` is equal to the length of `destinations`.

Remediation

This issue has been acknowledged by Biconomy Labs, and a fix was implemented in commit 3bf128e9.

3.3 Missing test suite code coverage

- **Target**: BatchedSessionRouter, ERC20SessionValidationModule, SessionKeyManagerModule
- **Category**: Code Maturity
- **Likelihood**: Low
- **Severity**: Low
- **Impact**: Low

Description

In our assessment of Biconomy Batched Session Router Module's test suite, we observed that while it provides adequate coverage for many aspects of the codebase, there are specific branches and codepaths that appear to be under-tested or not covered at all.

Some functions in the smart contract are not covered by any unit or integration tests, to the best of our knowledge. The following functions do not have full test coverage:

BatchedSessionRouter.sol: `validateUserOp`.

ERC20SessionValidationModule.sol: `validateSessionParams`.

SessionKeyManagerModule.sol: `validateSessionKey`.

Impact

Because correctness is so critical when developing smart contracts, we always recommend that projects strive for 100% code coverage. Testing is an essential part of the software development life cycle. No matter how simple a function may be, untested code is always prone to bugs.

Recommendations

Expand the test suite so that all functions are covered by unit or integration tests.

Remediation

This issue has been acknowledged by Biconomy Labs, and a fix was implemented in commit 12037aff.

4 Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the modules and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

4.1 Module: BatchedSessionRouterModule.sol

Function: `validateUserOp(UserOperation userOp, byte[32] userOpHash)`

Validates `userOperation`.

Inputs

- `userOp`
  - **Control**: Full.
  - **Constraints**: Needs to contain valid `selector`.
  - **Impact**: User Operation to be validated. If invalid, the function will revert or return a failure code.
- `userOpHash`
  - **Control**: Full.
  - **Constraints**: Must be a valid 32-byte-hash representation of the corresponding `userOp`.
  - **Impact**: Hash of the User Operation to be validated. Acts as a unique identifier or checksum of the User Operation.

Branches and code coverage (including function calls)

Intended branches

- Function returns `SIG_VALIDATION_SUCCESS` for a valid `UserOp` and valid `userOpHash`.
  ☑ Test coverage
- Function returns `SIG_VALIDATION_FAILED` if the userOp was signed with an improper session key.
  ☑ Test coverage

Negative behavior

- Function reverts when `userOp.sender` is an unregistered smart contract.
  ☑ Negative test
- Function reverts when the length of `user.signature` is less than 65.
  ☑ Negative test

4.2 Module: ERC20SessionValidationModule.sol

Function: `validateSessionParams(address destinationContract, uint256 callValue, byte[] _funcCallData, byte[] _sessionKeyData, byte[] None)`

This validates that the call (`destinationContract`, `callValue`, and `funcCallData`) complies with the Session Key permissions represented by `sessionKeyData`.

Inputs

- `destinationContract`
  - **Control**: Full.
  - **Constraints**: Must match the token address specified in `_sessionKeyData`.
  - **Impact**: The address of the contract to be called.
- `callValue`
  - **Control**: Full.
  - **Constraints**: Must be zero in value, as nonzero values will result in a revert.
  - **Impact**: The value to be sent with the call.
- `_funcCallData`
  - **Control**: Full.
  - **Constraints**: Must adhere to the ERC-20 standard.
  - **Impact**: The data for the call. It is parsed inside the SVM.
- `_sessionKeyData`
  - **Control**: Full.
  - **Constraints**: Must contain valid session key data that represents session key permissions.
  - **Impact**: SessionKey data that describes sessionKey permissions.
- `None`
  - **Control**: Full.
  - **Constraints**: N/A.
  - **Impact**: N/A.

Branches and code coverage (including function calls)

Intended branches

- Function returns the session key for a valid `destinationContract`, `callValue`, and `_funcCallData` that matches `_sessionKeyData`.
  ☑ Test coverage

Negative behavior

- Function reverts with `ERC20SV Invalid Token` when `destinationContract` does not match the token address in `_sessionKeyData`.
  ☑ Negative test
- Function reverts with `ERC20SV Non Zero Value` when a nonzero `callValue` is provided.
  ☑ Negative test
- Function reverts with `ERC20SV Wrong Recipient` when the recipient in `_funcCallData` does not match the intended recipient from `_sessionKeyData`.
  ☑ Negative test
- Function reverts with `ERC20SV Max Amount Exceeded` when the amount specified in `_funcCallData` exceeds the `maxAmount` described in `_sessionKeyData`.
  ☑ Negative test

4.3 Module: SessionKeyManagerModule.sol

Function: `setMerkleRoot(byte[32] _merkleRoot)`

Sets the Merkle root of a tree containing session keys for `msg.sender`.

Inputs

- `_merkleRoot`
  - **Control**: Full.
  - **Constraints**: N/A.
  - **Impact**: The Merkle root of a tree that contains session keys with their permissions and parameters.

Branches and code coverage (including function calls)

Intended branches

- Should successfully set the Merkle root of a tree containing session keys for `msg.sender`.
  ☑ Test coverage

Function: `validateSessionKey(address smartAccount, uint48 validUntil, uint48 validAfter, address sessionValidationModule, byte[] sessionKeyData, byte[32][] merkleProof)`

Validates that Session Key and parameters are enabled by being included into the Merkle tree.

Inputs

- `smartAccount`
  - **Control**: Full.
  - **Constraints**: Must be a valid Ethereum address.
  - **Impact**: The `smartAccount` for which the session key is being validated.
- `validUntil`
  - **Control**: Full.
  - **Constraints**: N/A.
  - **Impact**: The timestamp when the session key expires.
- `validAfter`
  - **Control**: Full.
  - **Constraints**: N/A.
  - **Impact**: The timestamp when the session key becomes valid.
- `sessionValidationModule`
  - **Control**: Full.
  - **Constraints**: Must be a valid contract address.
  - **Impact**: The address of the Session Validation Module.
- `sessionKeyData`
  - **Control**: Full.
  - **Constraints**: N/A.
  - **Impact**: The session parameters (limitations/permissions).
- `merkleProof`
  - **Control**: Full.
  - **Constraints**: N/A.
  - **Impact**: The Merkle proof for the leaf that represents this session key and params.

Branches and code coverage (including function calls)

Intended branches

- Function successfully fetches the session key storage for the provided smart account.

☑ Test coverage

Negative behavior

- Function reverts with `SessionNotApproved` due to invalid session key (data).
  ☑ Negative test

Function call analysis

- `rootFunction → verify(bytes32[], bytes32, bytes32)`
  - **What is controllable?**: `merkleProof`, `smartAccount`, `validUntil`, `validAfter`, `sessionValidationModule`, and `sessionKeyData`.
  - **If return value controllable, how is it used and how can it go wrong?**: It is used to verify the proof.
  - **What happens if it reverts, reenters, or does other unusual control flow?**: N/A.

Function: `validateUserOp(UserOperation userOp, byte[32] userOpHash)`

Validates `userOperation`.

Inputs

- `userOp`
  - **Control**: Full.
  - **Constraints**: N/A.
  - **Impact**: User Operation to be validated.
- `userOpHash`
  - **Control**: Full.
  - **Constraints**: Must be a valid 32-byte-hash representation of the corresponding `userOp`.
  - **Impact**: Hash of the User Operation to be validated.

Branches and code coverage (including function calls)

Intended branches

- Function is successfully invoked.
  ☑ Test coverage

Negative behavior

- Function reverts with `SIG_VALIDATION_FAILED`.
  ☑ Negative test

- Should revert with wrong session key data.
  ☑ Negative test
- Should revert with the wrong session validation module address.
  ☑ Negative test
- Should revert if session key is already expired.
  ☑ Negative test
- Should revert if session key is not yet valid.
  ☑ Negative test
- Should revert with wrong `validAfter`.
  ☑ Negative test
- Should revert with wrong `validUntil`.
  ☑ Negative test
- Should revert if signed with the session key that is not in the Merkle tree.
  ☑ Negative test

Function call analysis

- `rootFunction → _getSessionData(address)`
  - **What is controllable?**: N/A.
  - **If return value controllable, how is it used and how can it go wrong?**: N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?**: N/A.
- `rootFunction → validateSessionKey(address, uint48, uint48, address, byte[], byte[32][])`
  - **What is controllable?**: `userOp`.
  - **If return value controllable, how is it used and how can it go wrong?**: N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?**: N/A.
- `rootFunction → _packValidationData(bool, unit48, uint48)`
  - **What is controllable?**: `userOp` and `userOpHash`.
  - **If return value controllable, how is it used and how can it go wrong?**: True for signature failure, false for success.
  - **What happens if it reverts, reenters, or does other unusual control flow?**: N/A.

5 Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Ethereum Mainnet.

During our assessment on the scoped Biconomy Batched Session Router Module modules, we discovered three findings. No critical issues were found. One finding was of low impact and the other findings were informational in nature. Biconomy Labs acknowledged all findings and implemented fixes.

5.1 Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.