Zellic

Prepared for
Fil Makarov
Biconomy

Prepared by
Nipun Gupta
Vlad Toie
Zellic

December 4, 2023

Biconomy Multi Owned ECDSA

Smart Contract Security Assessment

Contents

About Zellic **4**

1. Executive Summary **4**

1.1. Goals of the Assessment 5

1.2. Non-goals and Limitations 5

1.3. Results 5

2. Introduction **6**

2.1. About Biconomy Multi Owned ECDSA 7

2.2. Methodology 7

2.3. Scope 9

2.4. Project Overview 9

2.5. Project Timeline 10

3. Detailed Findings **10**

3.1. Removal of all owners can underflow 11

3.2. Wrong parameter used in revert message 13

3.3. Entries of `eoaOwners` not checked 14

4. Discussion **15**

4.1. General additional checks 16

5. Threat Model **16**

5.1. Module: MultiOwnedECDSAModule.sol 17

6.	**Assessment Results**	**20**
6.1.	Disclaimer	21

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 from November 30th to December 1st, 2023. During this engagement, Zellic reviewed Biconomy Multi Owned ECDSA'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:

- How does the Multi Owned ECDSA Module verify signatures?
- Does the codebase respect the ERC-4337 standard for Smart Accounts?
- Is it possible for Unauthorized EOAs to perform actions on behalf of the Smart Accounts?

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
- The rest of the contracts in the codebase

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

During this assessment, the limited scope of the audit prevented us from performing a comprehensive threat model for the entire codebase.

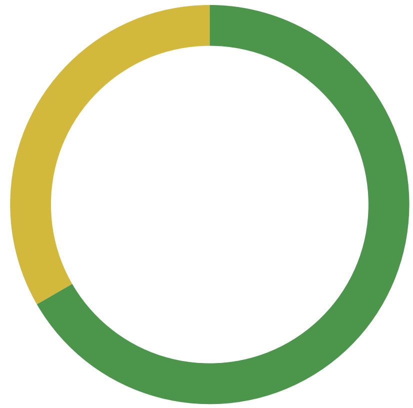
1.3. Results

During our assessment on the scoped Biconomy Multi Owned ECDSA contracts, we discovered three findings. No critical issues were found. One finding was of medium impact and two were of low impact.

Additionally, Zellic recorded its notes and observations from the assessment for Biconomy's benefit in the Discussion section (4. ↗) at the end of the document.

Breakdown of Finding Impacts

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



2. Introduction

2.1. About Biconomy Multi Owned ECDSA

The Multi Owner ECDSA Module allows having multiple owners for a smart account.

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 contracts.

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, upgradability 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 contracts itself. These observations — found in the Discussion (4. ↗) 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 Multi Owned ECDSA Contracts

Repository	https://github.com/bcnmy/scw-contracts/ ↗
Version	scw-contracts: `7f73eed4a4e2152e53764b2e0de8b6ee88f5b5e1`
Program	MultiOwnedECDSAModule.sol
Type	Solidity
Platform	EVM-compatible

2.4. Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of two person-days. The assessment was conducted over the course of two 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:

Nipun Gupta
Engineer
nipun@zellic.io ↗

Vlad Toie
Engineer
vlad@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

November 30, 2023	Start of primary review period
December 1, 2023	End of primary review period

3. Detailed Findings

3.1. Removal of all owners can underflow

Target	MultiOwnedECDSAModule		
Category	Business Logic	**Severity**	Medium
Likelihood	Low	**Impact**	Medium

Description

The `removeOwner` function facilitates the removal of owners from one's smart account.

```
function removeOwner(address owner) external override {

    if (!_smartAccountOwners[owner][msg.sender])
        revert NotAnOwner(owner, msg.sender);
    _transferOwnership(msg.sender, owner, address(0));
    unchecked {
        --numberOfOwners[msg.sender];
    }
}
```

It first checks whether the `owner` to be removed actually belongs to the `smartAccountOwners` mapping for the given `msg.sender`, and then it transfers the ownership from `owner` to `address(0)`.

```
function _transferOwnership(
    address smartAccount,
    address owner,
    address newOwner
) internal {
    _smartAccountOwners[owner][smartAccount] = false;
    _smartAccountOwners[newOwner][smartAccount] = true;
    emit OwnershipTransferred(smartAccount, owner, newOwner);
}
```

This is a workaround solution, as, by default, `address(0)`'s entry within the `smartAccountOwners` mapping would point to `false`. The setting of `address(0)`'s entry to true allows `msg.sender` to keep calling `removeOwner`, which will do the same transfer of ownership to `address(0)` potentially over and over again and would eventually underflow the `numberOfOwner` mapping, as the operations performed there are under `unchecked`.

```
function removeOwner(address owner) external override {

    if (!_smartAccountOwners[owner][msg.sender])
        revert NotAnOwner(owner, msg.sender);
    _transferOwnership(msg.sender, owner, address(0));
    unchecked {
        --numberOfOwners[msg.sender];
    }
}
```

Impact

The `numberOfOwners` mapping will eventually underflow. Moreover, the `_smartAccountOwners[address(0)][msg.sender]` will be set to `true`, which is an override of the intended and expected value of `false`, as `address(0)` should never be an owner.

Recommendations

We recommend performing the `delete` operation when removing the `_smartAccountOwners` entry. On top of that, we recommend removing the `unchecked` operation, as the gas savings are limited over the potential downside of underflows, as exemplified in this issue.

```
function removeOwner(address owner) external override {

    if (!_smartAccountOwners[owner][msg.sender])
        revert NotAnOwner(owner, msg.sender);
-   _transferOwnership(msg.sender, owner, address(0));
+   _smartAccountOwners[owner][msg.sender] = false;

    unchecked {
        --numberOfOwners[msg.sender];
    }
}
```

Remediation

This issue has been acknowledged by Biconomy, and a fix was implemented in commit bb275bf7 ↗.

3.2. Wrong parameter used in revert message

Target	MultiOwnedECDSAModule		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The `NotEOA` revert message is used when the given parameter is not an EOA. In the `transfer-Ownership` function, the wrong parameter is used as a revert message.

```
function transferOwnership(
        address owner,
        address newOwner
) external override {
    if (_isSmartContract(newOwner)) revert NotEOA(owner);
```

Impact

The revert message will point to irrelevant data.

Recommendations

We recommend changing the revert message to `revert NotEOA(newOwner);` as that reflects the actually verified parameter.

Remediation

This issue has been acknowledged by Biconomy, and a fix was implemented in commit `bb275bf7` ↗.

3.3. Entries of `eoaOwners` not checked

Target	MultiOwnedECDSAModule		
Category	Business Logic	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The `initForSmartContract` initializes the owners for a specific smart account.

```
function initForSmartAccount(
    address[] calldata eoaOwners
) external returns (address) {
    if (numberOfOwners[msg.sender] != 0) {
        revert AlreadyInitedForSmartAccount(msg.sender);
    }
    uint256 ownersToAdd = eoaOwners.length;
    for (uint256 i; i < ownersToAdd; ) {
        if (eoaOwners[i] == address(0))
        // ...
    }
    // ...
}
```

The entries of `eoaOwners` are not checked against being smart contracts. The owners should never be a smart contract as that is the general constraint that the contract adheres to.

Impact

The `eoaOwners` could point to an address that is a smart contract.

Recommendations

We recommend checking that none of the `eoaOwners` entries are smart contracts.

```
function initForSmartAccount(
    address[] calldata eoaOwners
) external returns (address) {
    if (numberOfOwners[msg.sender] != 0) {
```

```
            revert AlreadyInitedForSmartAccount(msg.sender);
        }
        uint256 ownersToAdd = eoaOwners.length;
        for (uint256 i; i < ownersToAdd; ) {
            if (_isSmartContract(ownersToAdd[i]))
        revert NotEOA(ownersToAdd[i]);
            if (eoaOwners[i] == address(0))
            // ...

        }
        // ...
    }
```

Remediation

This issue has been acknowledged by Biconomy.

Biconomy provided the following response:

> Unfortunately , the `extcodesize` opcode is banned for the `userOps` with initcode > 0, thus it is not possible to apply this check at the time of the deployment of smart account with this module as a first validator module.
>
> However, it is not an issue, as it won't be possible to sign => validate such userOp => such smart account can't be deployed. If this method is called when the given module is not the only validator, the situation won't be critical as it will be possible to change the owner thru other validator.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. General additional checks

There are a few additional checks that we recommend adding in the contract:

1. The `msg.sender` for the functions `initForSmartAccount`, `transferOwnership`, `addOwner`, and `removeOwner` is a smart account and therefore a smart contract. Currently, anyone (EOA or smart contract) can call these functions, but as the expected caller is a smart contract, the function should check if the value of `_isSmartContract(msg.sender)` is true. Currently, this missing check does not cause any security issues.

2. The length of `eoaOwners.length` should be checked and reverted if it is zero. Otherwise, the function `initForSmartAccount` could be called again.

These issues have been acknowledged by Biconomy, and a fix for issue 2 was implemented in commit `0bcc21d5` ↗

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts 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.

5.1. Module: MultiOwnedECDSAModule.sol

Function: `addOwner(address owner)`

The function is used to add a new owner for the smart account.

Inputs

- `owner`
  - **Control**: Fully controlled by caller.
  - **Constraints**: Should not be `address(0)`.
  - **Impact**: The `owner` becomes a new owner of the smart account.

Branches and code coverage

Intended branches

- The mapping `_smartAccountOwners` is set to true for the `owner`.
  - ☐ Test coverage
- The value of `numberOfOwners` mapping increases for `msg.sender`.
  - ☐ Test coverage

Negative behavior

- Revert if `owner` is a smart contract.
  - ☐ Negative test
- Revert if `owner` is `address(0)`.
  - ☐ Negative test
- Revert if `owner` is already an owner of the smart account.
  - ☐ Negative test

Function: `initForSmartAccount(address[] eoaOwners)`

Initializes the module for a smart account.

Inputs

- `eoaOwners`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: No constraints.
    - **Impact**: These addresses become the owners of the smart account.

Branches and code coverage

Intended branches

- Update the `_smartAccountOwners` mapping for all the `eoaOwners`.
    - [x] Test coverage
- Update the `numberOfOwners` mapping for `msg.sender`.
    - [x] Test coverage

Negative behavior

- Revert if any of the `eoaOwners` is `address(0)`.
    - [ ] Negative test
- Revert if `numberOfOwners[msg.sender]` is nonzero.
    - [ ] Negative test
- Revert if any of the `eoaOwners` is already an owner for the smart account.
    - [ ] Negative test

Function: `removeOwner(address owner)`

The function is used to remove an owner.

Inputs

- `owner`
    - **Control**: Fully controlled by caller.
    - **Constraints**: Should not be `address(0)`.
    - **Impact**: The address to remove from ownership.

Branches and code coverage

Intended branches

- The mapping `_smartAccountOwners` is set to false for `owner`.
    - [x] Test coverage
- The value of the `numberOfOwners` mapping is decreased for the `msg.sender`.
    - [x] Test coverage

Negative behavior

- Revert if `owner` is not an owner of the smart account.
  - [ ] Negative test

Function call analysis

- `this._transferOwnership(msg.sender, owner, newOwner)`
  - **What is controllable?** `msg.sender`, `owner`, and `newOwner`.
  - **If the return value is 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.

Function: `transferOwnership(address owner, address newOwner)`

The function is used to transfer the ownership of a smart account.

Inputs

- `owner`
  - **Control**: Fully controlled by caller.
  - **Constraints**: Should not be `address(0)`.
  - **Impact**: This is the previous owner to be removed.
- `newOwner`
  - **Control**: Fully controlled by caller.
  - **Constraints**: Should not be `address(0)`.
  - **Impact**: This is the new owner to be transferred ownership to.

Branches and code coverage

Intended branches

- Set `_smartAccountOwners` for `owner` as false.
  - [x] Test coverage
- Set `_smartAccountOwners` for `newOwner` as true.
  - [x] Test coverage

Negative behavior

- Revert if `newOwner` is a smart contract.
  - [ ] Negative test
- Revert if `newOwner` or `owner` is `address(0)`.

- ☐ Negative test
- Revert if `owner` and `newOwner` are same addresses.
  - ☐ Negative test
- Revert if `owner` is not an owner of the smart account.
  - ☐ Negative test
- Revert if `newOwner` is already an owner of the smart account.
  - ☐ Negative test

Function call analysis

- `this._transferOwnership(msg.sender, owner, newOwner)`
  - **What is controllable?** `msg.sender`, `owner`, and `newOwner`.
  - **If the return value is 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.

6. 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 Multi Owned ECDSA contracts, we discovered three findings. No critical issues were found. One finding was of medium impact and two were of low impact. Biconomy acknowledged all findings and implemented fixes.

6.1. 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.