Zellic

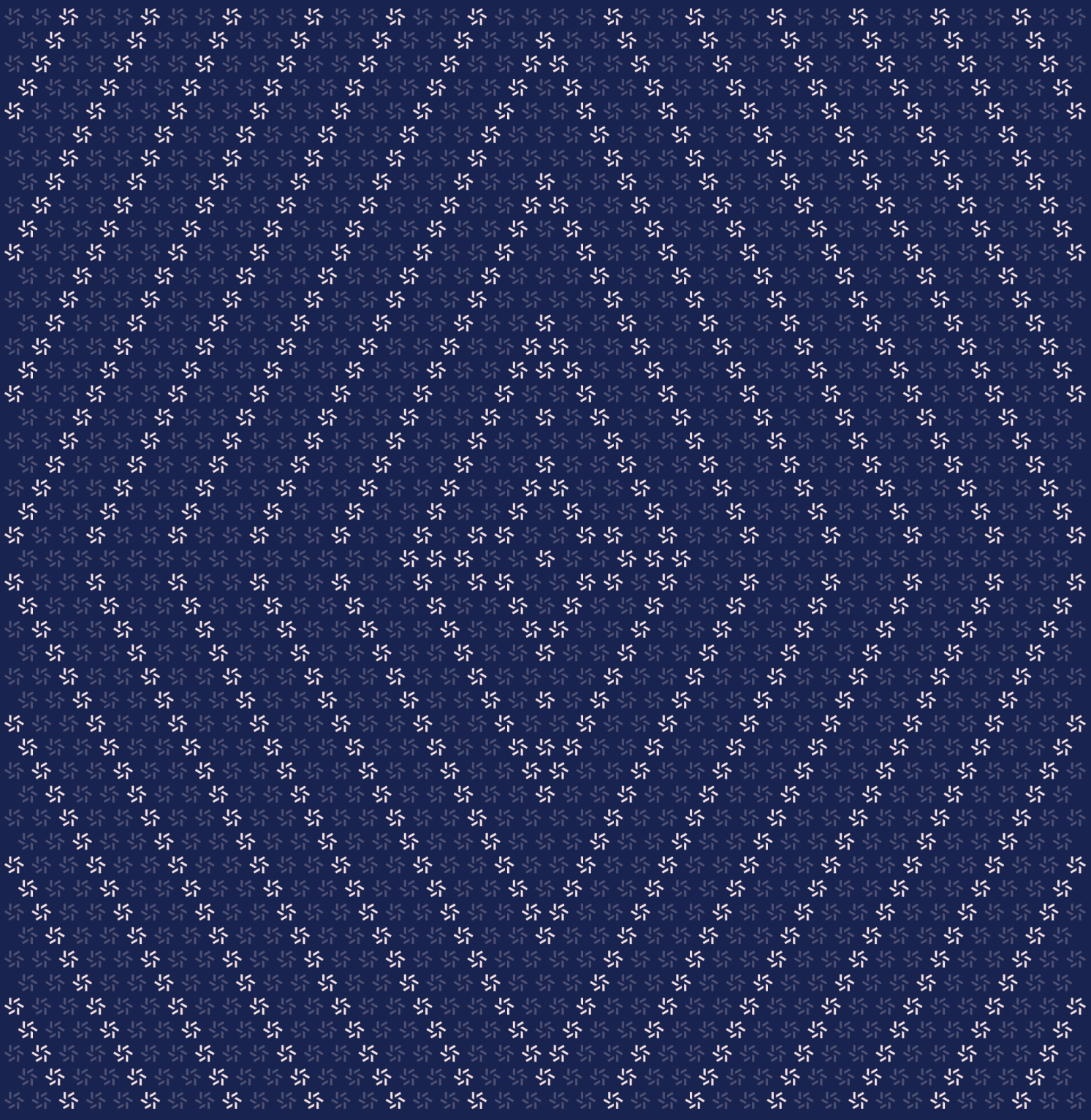
Prepared for
Adam Boudjemaa
Biconomy Labs

Prepared by
Syed Faraz Abrar
Katerina Belotskaia
Zellic

February 14, 2024

Biconomy Smart Account

Smart Contract Patch Review



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security patch review for Biconomy Labs from January 29th to February 7th, 2024. During this engagement, Zellic reviewed Biconomy Smart Account's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. 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 question:

- Were any bugs introduced during recent optimizations, refactoring, or updates?

1.3. 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.

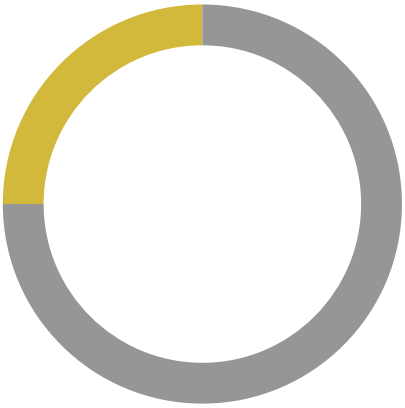
1.4. Results

During our assessment on the scoped Biconomy Smart Account contracts, we discovered four findings. No critical issues were found. One finding was of medium impact and the other findings were informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for Biconomy Labs'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	0
Informational	3



2. Introduction

Biconomy SCW project focuses on enhancing modular smart accounts. Our project primarily involves developing smart contract wallets (SCW) that are highly customizable and secure, using account abstraction principles to optimize user experiences in decentralized applications.

We were asked to review several patches to Biconomy Smart Account for optimization, updates, and refactoring purposes. The purpose of this review was to focus exclusively on these new changes, evaluating them for any potential security vulnerabilities or inconsistencies. In section 4.1. ↗, we have provided an overview of notable changes.

2.1. Scope

The engagement involved a review of the following targets:

Biconomy Smart Account Contracts

Repository	https://github.com/bcnmy/scw-contracts ↗
Version	scw-contracts: `5e97846a633a9bd9be320f63abd66e036dc9abef`
Programs	• BaseSmartAccount.sol • SmartAccount.sol • ModuleManager.sol • SmartAccountFactory.sol • EcdsaOwnershipRegistryModule.sol • MultichainECDSAValidator.sol • Secp256r1.sol • BatchedSessionRouterModule.sol • SessionKeyManagerModule.sol • ERC20SessionValidationModule.sol • PasskeyRegistryModule.sol • AddressResolver.sol • AuthorizationModulesConstants.sol • BaseAuthorizationModule.sol
Type	Solidity
Platform	EVM-compatible

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:

Syed Faraz Abrar
Engineer
faith@zellic.io

Katerina Belotskaia
Engineer
kate@zellic.io

2.2. 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.

3. Detailed Findings

3.1. Session key `maxAmount` parameter is not stateful

Target	ERC20SessionValidationModule		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	Low	**Impact**	Medium

Description

One of the parameters in a session key used by the `ERC20SessionValidationModule` smart contract is the `maxAmount` parameter. This parameter determines the maximum amount of tokens the caller can transfer out of the smart account to a recipient (who is also authorized by the session key).

Within the `validateSessionUserOp()` function, the ERC20 Session Validation Module verifies that the user operation is not attempting to transfer more than session key's `maxAmount`. It will fail validation if it does.

However, the issue is that this validation is not stateful. The session key does not track the amount of tokens that have already been transferred using the session key.

Impact

This allows a malicious session key holder to continuously transfer out `maxAmount` of tokens over multiple user operations.

For example, if the `maxAmount` for a session key is 100 tokens, the expectation is that the session key becomes invalid or unusable after 100 tokens have been transferred using it.

However, in the current codebase, the user can keep using the session key across multiple user operations to transfer out `maxAmount` of tokens as many times as they would want. They only need to ensure that the session key has not expired.

Since this leads to loss of funds of the smart account, the severity is Critical. However, because session keys are intended to be issued to trusted parties, we have decided that the likelihood of this occurring is Low. Therefore, we conclude that the final impact of this vulnerability is Medium.

Recommendations

Consider adding an additional parameter to the session key that tracks how many tokens have already been transferred. This can then be validated in `validateSessionUserOp()` as well.

Remediation

This issue has been acknowledged by Biconomy Labs, and a fix was implemented in commit `1a11bd4f` ↗.

3.2. No `isContract` check in `initForSmartAccount()`

Target	EcdsaOwnershipRegistryModule		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

In the EcdsaOwnershipRegistryModule smart contract, we noted that the `transferOwnership()` function checks to ensure the new owner is an EOA.

However, the same check does not exist in the `initForSmartAccount()` function, which sets the initial owner.

Impact

This would allow a user to accidentally initialize the EcdsaOwnershipRegistryModule smart contract with a non-EOA owner. This would effectively make the module unusable, as a smart contract cannot produce valid signatures that will pass the `isValidSignature()` function checks.

Recommendations

We were initially going to recommend adding in the `isContract()` checks, but we learned that the `extcodesize` opcode is banned in user operations that have nonempty `initCode`. Since this would be the case for a user operation that is calling `initForSmartAccount()`, this is not a valid recommendation.

Instead, we recommend that Biconomy extensively document that the module owner cannot be an EOA. The function parameter is already named `eoaOwner`, but we would also recommend adding a comment above the function as well as adding it to any documentation related to this module.

Remediation

The Biconomy team have acknowledged this finding, stating that users will be expected to use the Biconomy SDK, which will perform off-chain `isContract()` checks. They have also stated that they will document this behavior extensively to make it extremely unlikely for users to make this mistake.

3.3. The `isValidSignatureUnsafe()` is redundant

Target	PasskeyRegistryModule		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

In the PasskeyRegistryModule, both the `isValidSignature()` and `isValidSignatureUnsafe()` functions have the same code. They both call `isValidSignatureForAddress()`, which will add the "`\x19Ethereum Signed Message:\n52`" prefix to the data hash and smart account address prior to hashing it and then verifying the signature against it.

This seems to be a copy-paste error, as in the EcdsaOwnershipRegistryModule, the `isValidSignatureUnsafe()` function calls `isValidSignatureForAddressUnsafe()`, which does not perform the same prefixing plus hashing.

Impact

The `isValidSignatureUnsafe()` function is currently redundant, as it does the same thing as the `isValidSignature()` function. This does not affect any core functionality of the module, and thus it is an Informational finding.

Recommendations

Consider porting over the `isValidSignatureForAddressUnsafe()` function from EcdsaOwnershipRegistryModule, and then having `isValidSignatureUnsafe()` call this function instead.

Remediation

This issue has been acknowledged by Biconomy Labs, and a fix was implemented in commit `34a75fa2` ↗.

3.4. Some interfaces are missing functions

Target	IAccountRecoveryModule, IPasskeyRegistryModule		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The `IAccountRecoveryModule` interface is missing the following external functions:

- `resetModuleForCaller()`
- `setAllowedRecoveries()`
- `executeRecovery()`

The `IAddressResolver` interface is missing the following external functions:

- `resolveAddressesV1()`
- `resolveAddresses()`
- `resolveAddressesFlexibleForV2()`

The `IPasskeyRegistryModule` is missing the `getOwner()` external function.

Impact

Users or smart contracts that make use of these interfaces will not be able to call these functions.

Recommendations

Consider adding these functions to the interfaces.

Remediation

This issue has been acknowledged by Biconomy Labs, and a fix was implemented in commit `34a75fa2` ↗.

4. Patch Review

This section documents the changes between the contracts BaseSmartAccount.sol, SmartAccount.sol, ModuleManager.sol, SmartAccountFactory.sol, and EcdsaOwnershipRegistryModule.sol in the previous audit commit 96bb6655 ↗; the contract MultichainECDSAValidator.sol in the previous audit commit b8d4e08e ↗; the contract Secp256r1.sol in the previous audit commit 43525074 ↗; the contracts BatchedSessionRouterModule.sol, SessionKeyManagerModule.sol, and ERC20SessionValidationModule.sol in the previous audit commit 3bf128e9 ↗; and the contract PasskeyRegistryModule.sol in the previous audit commit 5c5a6bfe ↗ and the latest commit 5e97846a ↗.

4.1. Notable changes

New function `isValidSignatureUnsafe`

This view function was introduced in SmartAccount.sol contract to validate an EIP-1271 signature by calling the function of the same name, `isValidSignatureUnsafe`, in a trusted module. In a successful validation, the return value will be `EIP1271_MAGIC_VALUE`, while a failed validation will return `bytes4(0xffffffff)`. This is in accordance with the standard EIP-1271.

In turn, this new `isValidSignatureUnsafe` function was implemented in EcdsaOwnershipRegistryModule and PasskeyRegistryModule modules' contracts. This function, unlike the implementation of the `isValidSignature`, does not add a prefix "`\x19Ethereum Signed Message:\n52`" to the hash of the data to be validated by `_verifySignature` to check that this data was signed by the signature of the expected signer. Also, the new `isValidSignatureForAddressUnsafe` function was introduced in EcdsaOwnershipRegistryModule, which allows to verify signatures like the `isValidSignatureUnsafe` but for any smart account address. The modules BatchedSessionRouterModule and SessionKeyManager do not support the `isValidSignatureUnsafe` function and will revert when called with any arguments.

4.2. Minor differences

BaseSmartAccount.sol

1. The internal function `_validateNonce` has been removed.
2. An argument `uint192 key` has been added to the `nonce` function. This argument is passed to `entryPoint().getNonce(address(this), key);`. Previously, a constant `0` was passed instead of `key`.

SmartAccount.sol

1. The contract no longer inherits from the IERC165 interface.
2. The functions `executeCall` and `executeBatchCall` have been renamed to `execute` and `executeBatch`, respectively.
3. The function `validationModule` has been optimized for gas efficiency using assembly

code. Additionally, the call to the internal `_validateNonce` has been removed.

4. Functions `executeCall_s1m` and `executeBatchCall_4by` have been renamed to `execute_ncC` and `executeBatch_y6U`.

ModuleManager.sol

1. The `execTransactionFromModule` and `execBatchTransactionFromModule` functions have become payable.
2. A new argument `txGas` has been added to the `execTransactionFromModuleReturnData` function. The previous version of this function without `txGas` now calls `execTransactionFromModuleReturnData` with a zero `txGas` amount.
3. The contract uses a linked list of allowed modules. The first module is always the auth module, which is necessary for validating user operations and to not brick the contract. All other modules are linked to it. The `_disableModule()` function prevents the removal of the first auth module.
4. The `execBatchTransactionFromModule` function, instead of calling the `execute` function from the Executor.sol library, calls the internal `_executeFromModule`. This function, in turn, calls `_execute` from Executor.sol and emits events in case of successful and unsuccessful execution.

SmartAccountFactory.sol

1. The contract is inherited from the Stakeable.sol contract.
2. The Proxy contract has been renamed to BiconomyMSAProxy. The contract itself remains unchanged.
3. In the `deployAccount` function, the check `if (initializer.length > 0)` has been deleted.

EcdsaOwnershipRegistryModule.sol

1. The internal `_verifySignature` function has been updated to check if the signature was made over `dataHash.toEthSignedMessageHash()`.
2. The new internal `_transferOwnership` function has been added.
3. The new view function `getOwner` has been added, which returns the `owner` address for `smartAccount`.
4. New `renounceOwnership` function has been added, which set zero `owner` address for `msg.sender`.

BatchedSessionRouterModule.sol

1. Added `uint256 private constant MODULE_SIGNATURE_OFFSET = 96;`.
2. Instead of using a two-step process for decoding data from the signature, the `validateUserOp` function directly decodes data from `userOp.signature[MODULE_SIGNATURE_OFFSET:]`.
3. The `validateUserOp` function was updated by verification that the `sessionKeyManager` module is an enabled module.
4. The function `isValidSignatureUnsafe` is not supported.

SessionKeyManagerModule.sol

1. The function `isValidSignatureUnsafe` is not supported.
2. Instead of using a two-step process for decoding data from the signature, the `validateUserOp` function directly decodes data from `userOp.signature[MODULE_SIGNATURE_OFFSET:]`.

PasskeyRegistryModule.sol

1. A new view function named `getOwner` has been added, which returns `smartAccountPasskey[smartAccount]`.
2. The `_verifySignature` function has been updated to get the new `address smartAccount` argument. Previously, this function used the `msg.sender` address.
3. In the `isValidSignature` function, the `msg.sender` is now passed to the `isValidSignatureForAddress` function, which in turn calls the updated `_verifySignature`.

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 Smart Account contracts, we discovered four findings. No critical issues were found. One finding was of medium 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.