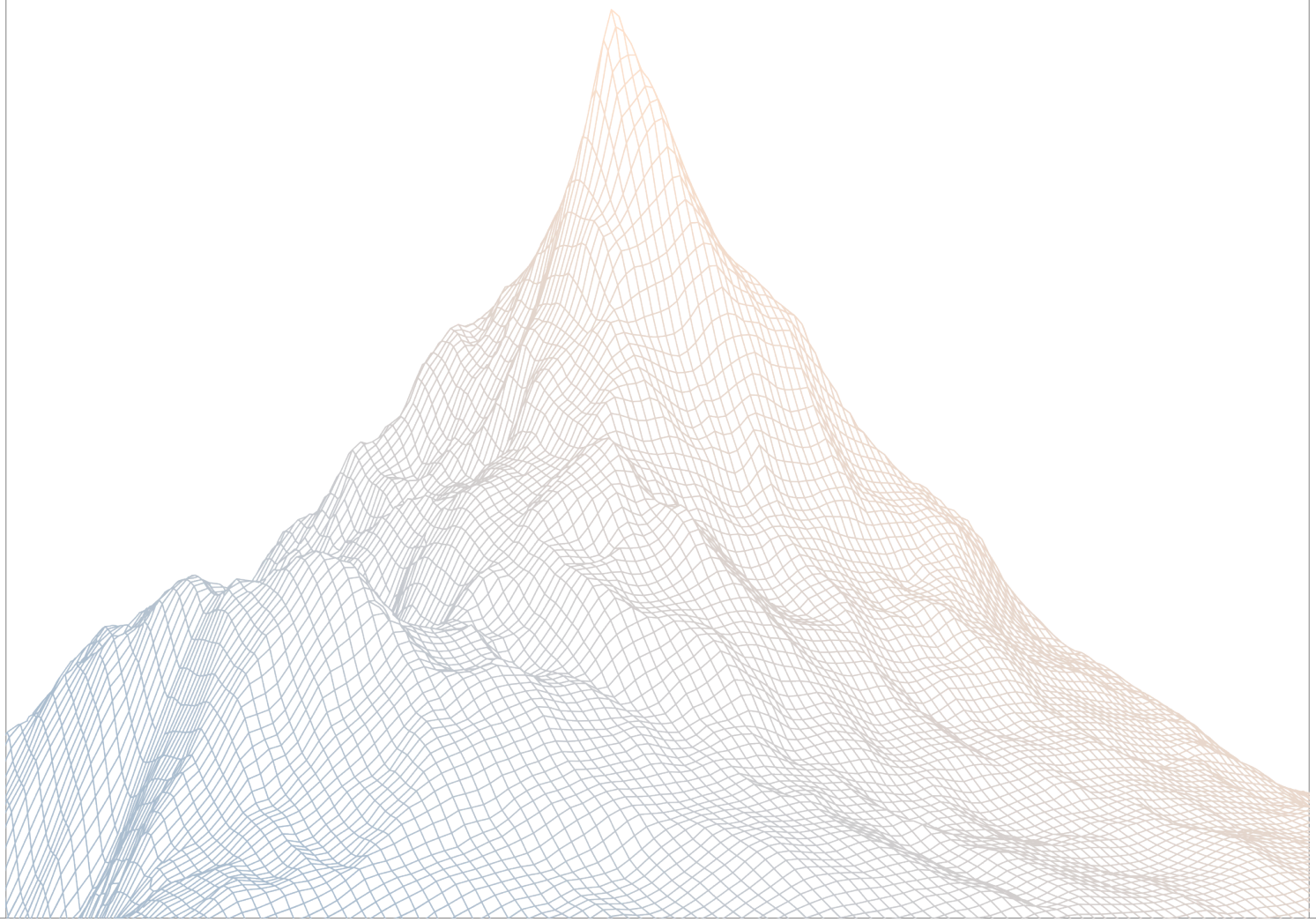
Zenith

Biconomy Composability

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: March 3rd to March 7th, 2025

AUDITED BY: cccz
chinmay

Contents

1 Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Biconomy

Biconomy's Composability Stack addresses challenges in decentralized finance (DeFi) by enabling developers to create dynamic, multi-step transactions directly from frontend code, eliminating the need for custom smart contracts or multiple user interventions.

2.2 Scope

The engagement involved a review of the following targets:

Target	mee-contracts
Repository	https://github.com/bcnmy/mee-contracts
Commit Hash	`6eb7914661f365374ab37c16a01ba51f6d3c3547`
Files	`contracts/composability/* (+ any files related to that change)`

2.3 Audit Timeline

March 3, 2025	Audit start
March 7, 2025	Audit end
March 19, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	0
Low Risk	5
Informational	1
Total Issues	**7**

3

Findings Summary

ID	Description	Status
H-1	The native token is not forwarded when calling execute-FromExecutor()	Resolved
L-1	setEntrypoint() can have the same effect as onUninstall() in ComposableExecutionModule.sol	Resolved
L-2	executeComposable() flow invokes hook checks twice	Resolved
L-3	aggregateValue should only be incremented if to is not the address(0)	Resolved
L-4	Refund of excess native tokens	Resolved
L-5	ComposableExecutionModule.onInstall() inconsistent with the specification	Resolved
I-1	Unused structures, errors and events in composability code	Resolved

4

Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] The native token is not forwarded when calling `executeFromExecutor()`

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Medium

Target

- ComposableExecutionModule.sol#L59-L64

Description:

When the smart wallet (Nexus, etc.) calls `ComposableExecutionModule.executeComposable()`, the native tokens are forwarded to ComposableExecutionModule.

```
} else if (calltype == CALLTYPE_SINGLE) {
    (success, result) = handler.call{ value: msg.value
    }(ExecLib.get2771CallData(callData));
} else {
```

And then `ComposableExecutionModule.executeComposable()` calls the `executeFromExecutor()` function of the smart wallet.

```
if (execution.to != address(0)) {
    returnData = IERC7579Account(msg.sender).executeFromExecutor({
        mode: ModeLib.encodeSimpleSingle(),
        executionCalldata: ExecutionLib.encodeSingle(execution.to,
    execution.value, composedCalldata)
    });
} else {
```

The problem here is that it does not forward any native tokens to the smart wallet, which may cause the subsequent execution to fail.

More seriously, the execution in the smart wallet may use the smart wallet's balance, then these forwarded native tokens will be locked in the ComposableExecutionModule.

Recommendations:

```
returnData = IERC7579Account(msg.sender).executeFromExecutor({
            returnData = IERC7579Account(msg.sender).executeFromExecutor{
                value:execution.value}({
                  mode: ModeLib.encodeSimpleSingle(),
                  executionCalldata:
ExecutionLib.encodeSingle(execution.to, execution.value,
composedCalldata)
              });
```

Biconomy: Resolved with PR-23

Zenith: Verified

4.2 Low Risk

A total of 5 low risk findings were identified.

[L-1] `setEntrypoint()` can have the same effect as `onUninstall()` in `ComposableExecutionModule.sol`

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- ComposableExecutionModule.sol#L73-L75

Description:

The smart account (Nexus) can call into `setEntrypoint()` on the `ComposableExecutionModule` to change its external `entrypoint` address.

This function allows it to set entrypoint as address(0). This will make the module state for the account equivalent to when the module has been uninstalled (ie. when the entrypoint state is deleted).

After such a call, `isInitialized()` checks on the module will start returning false even though the module is still installed on the account.

Recommendations:

```
    function setEntryPoint(address _entryPoint) external {
+       require(_entryPoint ≠ address(0), ZeroAddressNotAllowed());
        entryPoints[msg.sender] = _entryPoint;
    }
```

Biconomy: Resolved with PR-24

Zenith: Verified

[L-2] `executeComposable()` flow invokes hook checks twice

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- Nexus.sol#L171-L180
- ModuleManager.sol#L619-L652

Description:

The flow for a executeComposable() call is :

```
Entrypoint ⇒ SA.fallback() ⇒ fallback Handler ie.
ComposableExecutionModule.sol ⇒ SA.executeFromExecutor()
```

In this chain of calls, hook checks are invoked twice : in the following order =>

```
fallback() preCheck ⇒ call to fallback handler ⇒ preCheck in
SA.executeFromExecutor() ⇒ postCheck after the execution ⇒ flow comes back to
SA.fallback() ⇒ fallback postCheck
```

If the hook involves something like checking and incrementing a spending limit (or limiting the number of calls for a caller), then it may not work correctly. Example : SpendingLimitHook.

This may lead to unexpected outcomes for hook settings on the smart account.

Recommendations:

Make sure that the `executeComposable()` flow respects hooks' design expectations, and there are no unexpected results of double hook checks.

Biconomy: On Composability Module side: mitigation: PR-26. On Nexus side: PR-253

Zenith: Verified.

[L-3] `aggregateValue` should only be incremented if `to` is not the `address(0)`

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- ComposableExecutionModule.sol#L54-L67

Description:

`executeComposable()` will only call `executeFromExecutor()` if `to` is not `address(0)`, so it is better to increase `aggregateValue` when `to` is not `address(0)`.

```
ComposableExecution calldata execution = executions[i];
aggregateValue += execution.value;
require(msg.value >= aggregateValue, InsufficientMsgValue());
bytes memory composedCalldata
    = execution.inputParams.processInputs(execution.functionSig);
bytes[] memory returnData;
if (execution.to ≠ address(0)) {
    returnData = IERC7579Account(msg.sender).executeFromExecutor({
        mode: ModeLib.encodeSimpleSingle(),
        executionCalldata: ExecutionLib.encodeSingle(execution.to,
    execution.value, composedCalldata)
    });
} else {
    returnData = new bytes[](1);
    returnData[0] = "";
}
```

Recommendations:

```
            ComposableExecution calldata execution = executions[i];
        aggregateValue += execution.value;
        require(msg.value ≥ aggregateValue, InsufficientMsgValue());
```

```
        bytes memory composedCalldata
= execution.inputParams.processInputs(execution.functionSig);
        bytes[] memory returnData;
        if (execution.to ≠ address(0)) {
            aggregateValue += execution.value;
            require(msg.value ⩾ aggregateValue, InsufficientMsgValue());
            returnData
= IERC7579Account(msg.sender).executeFromExecutor({
                mode: ModeLib.encodeSimpleSingle(),
                executionCalldata:
ExecutionLib.encodeSingle(execution.to, execution.value,
composedCalldata)
            });
        } else {
            returnData = new bytes[](1);
            returnData[0] = "";
        }
```

```
    for (uint256 i; i < length; i++) {
        ComposableExecution calldata execution = executions[i];
        aggregateValue += execution.value;
        require(msg.value ⩾ aggregateValue, InsufficientMsgValue());
        bytes memory composedCalldata
= execution.inputParams.processInputs(execution.functionSig);
        bytes memory returnData;
        if (execution.to ≠ address(0)) {
            aggregateValue += execution.value;
            require(msg.value ⩾ aggregateValue, InsufficientMsgValue());
            returnData = _executeAction(execution.to, execution.value,
composedCalldata);
        } else {
            returnData = new bytes(0);
        }
        execution.outputParams.processOutputs(returnData,
address(this));
    }
```

Biconomy: Resolved with PR-24

Zenith: Verified.

[L-4] Refund of excess native tokens

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- ComposableExecutionModule.sol#L53-L69

Description:

`executeComposable()` requires `msg.value ⩾ aggregateValue`, so it is best to refund any excess native tokens.

```
require(msg.value >= aggregateValue, InsufficientMsgValue());
```

Recommendations:

```
        for (uint256 i; i < length; i++) {
            ComposableExecution calldata execution = executions[i];
            aggregateValue += execution.value;
            require(msg.value >= aggregateValue, InsufficientMsgValue());
            bytes memory composedCalldata
    = execution.inputParams.processInputs(execution.functionSig);
            bytes[] memory returnData;
            if (execution.to ≠ address(0)) {
                returnData
    = IERC7579Account(msg.sender).executeFromExecutor({
                    mode: ModeLib.encodeSimpleSingle(),
                    executionCalldata:
    ExecutionLib.encodeSingle(execution.to, execution.value,
    composedCalldata)
                });
            } else {
                returnData = new bytes[](1);
                returnData[0] = "";
            }
            execution.outputParams.processOutputs(returnData[0],
    msg.sender);
```

```
        }
    if(msg.value > aggregateValue){ msg.sender.transfer(msg.value -
        aggregateValue);}
```

Biconomy: Resolved with PR-24

Zenith: Verified

[L-5] `ComposableExecutionModule.onInstall()` inconsistent with the specification

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- ComposableExecutionModule.sol#L83-L87

Description:

In order to make `ComposableExecutionModule` be used as both EXECUTOR module and FALLBACK module, `ComposableExecutionModule.onInstall()` can be called multiple times to install the module.

```
function onInstall(bytes calldata data) external override {
    if (data.length >= 20) {
        entryPoints[msg.sender] = address(bytes20(data[0:20]));
    }
}
```

However, this violates the ERC7579 specification that `onInstall()` must revert when the module is enabled.

```
interface IERC7579Module {
     /**
     * @dev This function is called by the smart account during installation
    of the module
     * @param data arbitrary data that may be required on the module during
    `onInstall` initialization
     *
     * MUST revert on error (e.g. if module is already enabled)
     */
    function onInstall(bytes calldata data) external;
```

Recommendations:

Note: A single module that is of multiple types MAY decide to pass moduleTypeId inside data to onInstall and/or onUninstall methods, so those methods are able to properly handle installation/uninstallation for various types. Example:

According to the ERC7579 specification, for modules with multiple types, the type should be included in the passing data and different installations should be performed based on the type.

```
// Module.sol
function onInstall(bytes calldata data) external {
    // ...
    (uint256 moduleTypeId, bytes memory otherData) = abi.decode(data,
    (uint256, bytes));
    // ...
}
```

Like follows

```
mapping(uint256 => mapping(address => address)) private entryPoints;

function onInstall(bytes calldata data) external override {
    (uint256 moduleTypeId, bytes memory initData) = abi.decode(data,
    (uint256, bytes));
    require(moduleTypeId == TYPE_EXECUTOR || moduleTypeId == TYPE_FALLBACK);
    if(entryPoints[moduleTypeId][msg.sender] != 0 ) revert;
    entryPoints[moduleTypeId][msg.sender] = address(bytes20(initData[0:20]))
}
```

Biconomy: Resolved with PR-24

Zenith: Verified

4.3 Informational

A total of 1 informational findings were identified.

[I-1] Unused structures, errors and events in composability code

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

ComposableExecutionModule.sol#L28-L29 ComposableExecutionLib.sol

Description:

There are several instances of unused structures/ errors/ events in the codebase :

- structure `ParamValueType` in `ComposableExecutionLib.sol`
- Errors `ModuleAlreadyInitialized()` and `ExecutionFailed()` in `ComposableExecutionModule.sol`
- Errors `InvalidReturnDataHandling()`, `InvalidComposerInstructions()` and `StorageReadFailed()` in `ComposableExecutionLib.sol`
- Events `RemoteStorageSet()` and `CrossChainStorageSet()` in `Storage.sol`

Recommendations:

Remove these structures/ errors/ events as they are not used.

Biconomy: Resolved with PR-24 & PR-25

Zenith: Verified.