Horizon — Sequence Smart Contract Wallet v2

Date	February 2023
Auditors	Martin Ortner, Heiko Fisch; Tejaswa Rastogi (support)

1 Executive Summary

This report presents the results of our engagement with **Horizon Blockchain Games** to review the smart contracts of their **Sequence Wallet v2**.

The review was conducted over the course of three weeks, from **February 20, 2023**, to **March 10, 2023**. A total of 2 x 2.5 person-weeks were spent.

- At the start of the engagement, the development team provided an excellent high-level overview of the smart contract system. We discussed the core concepts and mapped them to the actual source code. We then set out to work into the first week with the knowledge gathered so far, creating visual representations of the overall system and the signature decoding mechanisms.
- Into the second week, the development team provided a deep-dive into how signature verification works. Over the course of the last two weeks, we manually reviewed the code for generic and logic bugs and reviewed business logic, access controls, and parsing/validation of transactions.

The Sequence Wallet (documentation) was previously reviewed by ConsenSys Diligence in May 2020 (under the name Arcadeum Wallet). We would like to note that the main findings from the previous engagement – while not repeated in detail in this report – still apply to v2 of the wallet. Very briefly:

- It is possible to destroy a wallet instance via `selfdestruct` and then redeploy a fresh instance (with the same configuration) to the original address; previous transactions would then become replayable. (5.1)
- As the wallet's configuration uniquely determines the salt for the CREATE2 deployment, it is not possible to have two different wallet instances with identical configuration. (5.2)

The wallet smart contract signature decoding and verification algorithms could be more strict. There is a lot of freedom on how to encode signatures, and the wallet generally opts to continue on invalid data instead of failing early. This is a valid design decision because – assuming that all components are well-designed and -implemented – any signature or input deviating from the expected format should eventually lead to an invalid wallet configuration root. During the course of the review, we have investigated many such "degrees of freedom" that subjectively increase the attack surface. We were, however, unable to construct exploitable scenarios from them. Here are some examples that we suggest to be double-checked by the development team:

- Missing data length checks: `LibBytesPointer` and `LibBytes` directly read from calldata. However, none of the methods ensure that enough calldata bytes were provided when reading x bytes from the input stream. Reading out-of-bounds will not fail and continue returning zero-padded data instead. We assume this is a conscious design decision to optimize byte reads. While this 'freedom' subjectively increases the potential attack surface (out-of-bounds reads should usually fail instead of silently returning zero-padded data) we were unable to find an exploitable scenario in the time allotted for the engagement.
- Ambiguity between signature type `FLAG_SIGNATURE` and `FLAG_DYNAMIC_SIGNATURE`. An `ecrecover` signature check of type `FLAG_SIGNATURE` can also be encoded as `FLAG_DYNAMIC_SIGNATURE`. This gives the signature encoder freedom over how to encode the exact same signature check.
- Chained signatures and recursions: It is valid to have chained signatures that only contain a single next signature item, which is equivalent to not chaining it at all. Signature evaluation may recursively call `recoverBranch()` (`FLAG_NESTED`, `FLAG_BRANCH`). There are many ways to encode the same signature verification using the various signature type and signature components.

After the delivery of the initial version of this report, the client has taken our findings into consideration and implemented fixes for some of them (PR 168). We have reviewed these changes and updated the report accordingly.
Three findings were considered non-issues or intentional behavior. Some minor issues or code quality recommendations were acknowledged in principle, but, considering that the codebase has undergone multiple audits, the client chose to keep the changes at this point to a minimum to reduce the risk of introducing new mistakes. In these cases, the client's rationale for not making any changes has been added to the finding.

2 Scope

Initially, our review focused on the following code revision:

- 0xsequence/wallet-contracts@0928bf2bf471acd367b60bd629076b1b8628e4f2

The fixes implemented in response to the initial version of this report led to revision:

- 0xsequence/wallet-contracts@a1d94968d92f153196ed715ce9294be895ca8a9e

A detailed list of the files in scope can be found in the Appendix.

2.1 Objectives

Together with the client's team, the following priorities for the review were identified:

1. Correctness of the implementation, consistent with the intended functionality and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.

3 System Overview

3.1 Detailed Design

This review covered the Solidity smart contracts implementing the proxy architecture as well as the two basic versions of the wallet: (1) A "fixed" version with immutable signers configuration. This is the initial implementation contract for a deployed proxy. (2) An upgradable version that allows changing the configuration. (Note that the *implementation* can be upgraded in both cases.) Internally, the proxy's implementation contract is switched from (1) to (2) when the first configuration change happens. A wallet's configuration is essentially the root of a Merkle tree, which allows storing even complex permission scenarios in a single storage word.

The system has the following components:

- A minimal delegatecall proxy implementation (its creation bytecode is hardcoded in Wallet.sol).
- A factory contract that deploys instances of the proxy via CREATE2.
- Implementations which are composed from a set of modules. As mentioned above, there are two implementations, MainModule and MainModuleUpgradable.

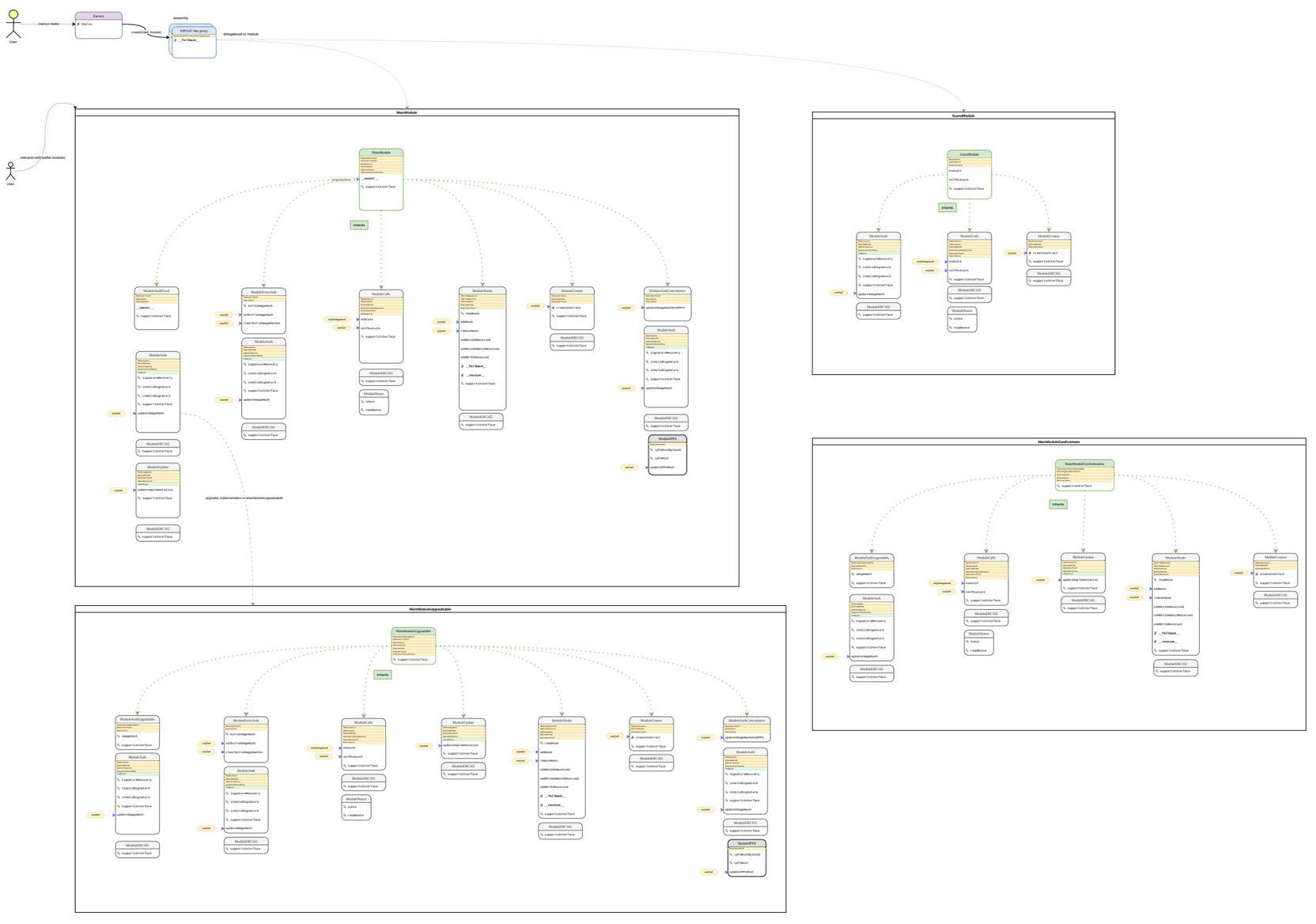
3.2 Key Observations

- When creating a new wallet instance, the CREATE2 salt is a hash derived from the multisig configuration, including the signing threshold, owner addresses, and weights. In the non-upgradable version of the wallet, input data can later be validated by emulating the behavior of CREATE2 and checking that the output matches the address of the contract account. This mechanism allows wallets to be deployed with an initial configuration but without using any additional storage or setup procedure. In the upgradable version, the config hash is stored in an additional state variable instead.
- In existing modules, every storage value is operated by the `ModuleStorage` module. By default, a wallet has the following storage values: nonces, implementation address, hooks, and image (i.e., the hashed configuration) – the latter only in the upgradable wallet. Every value has a pseudo-random location in the storage.
- The `ModuleHooks` and the `ModuleCalls` contracts (that are the parts of both current implementations) are using the `delegatecall` instruction. This instruction should be treated very carefully. So when adding hooks or making a `delegatecall`, users should make sure that target contracts are always using the `ModuleStorage` library and unique keys for every storage interaction.

3.3 Architecture Diagram

This section describes the top-level/deployable contracts, their inheritance structure and interfaces, actors, permissions, and important contract interactions of the system under review.

Contracts are depicted as boxes. Public reachable interface methods are outlined as rows in the box. The 🔑 icon indicates that a method is declared as non-state-changing (view/pure), while other methods may change state. A yellow dashed row at the top of the contract shows inherited contracts. A green dashed row at the top of the contract indicates that that contract is used in a `using for` declaration. Modifiers used as ACL are connected as yellow bubbles in front of methods.



Sequence Wallet - General Overview

3.4 Wallet Factory Proxy Code Analysis

The wallet factory deploys EIP-1167-like proxies that delegate the code execution context to one of the top-level wallet modules. The original EIP-1167 implementation statically encodes the proxy's delegatecall target while the variant used by Sequence Wallet stores the delegate target in the contract's storage at slot `storage[uint(address(this))]`. When interacted with, the contract retrieves the target address from `implementation = storage[uint(address(this))]` and delegatecalls the `implementation`.

When deploying a proxy, the factory appends the delegatecall target address to the end of the Wallet creation code. The following byte sequence represents the Wallet contract creation code:

```
0x603a600e3d39601a805130553df3363d3d373d3d3d363d30545af43d82803e903d91601857fd5bf3
```

This hex-string can be dissected into three parts:

- the deploy code (executed on creation),
- the contract code (executed when interacting with the contract), and
- the delegatecall target address appended to the end.

```
       deploy code                         contract code                                target (sstore'd on creation)
<603a600e3d39601a805130553df3><363d3d373d3d3d363d30545af43d82803e903d91601857fd5bf3><delegateTarget>
```

The bytecode disassembles as following:

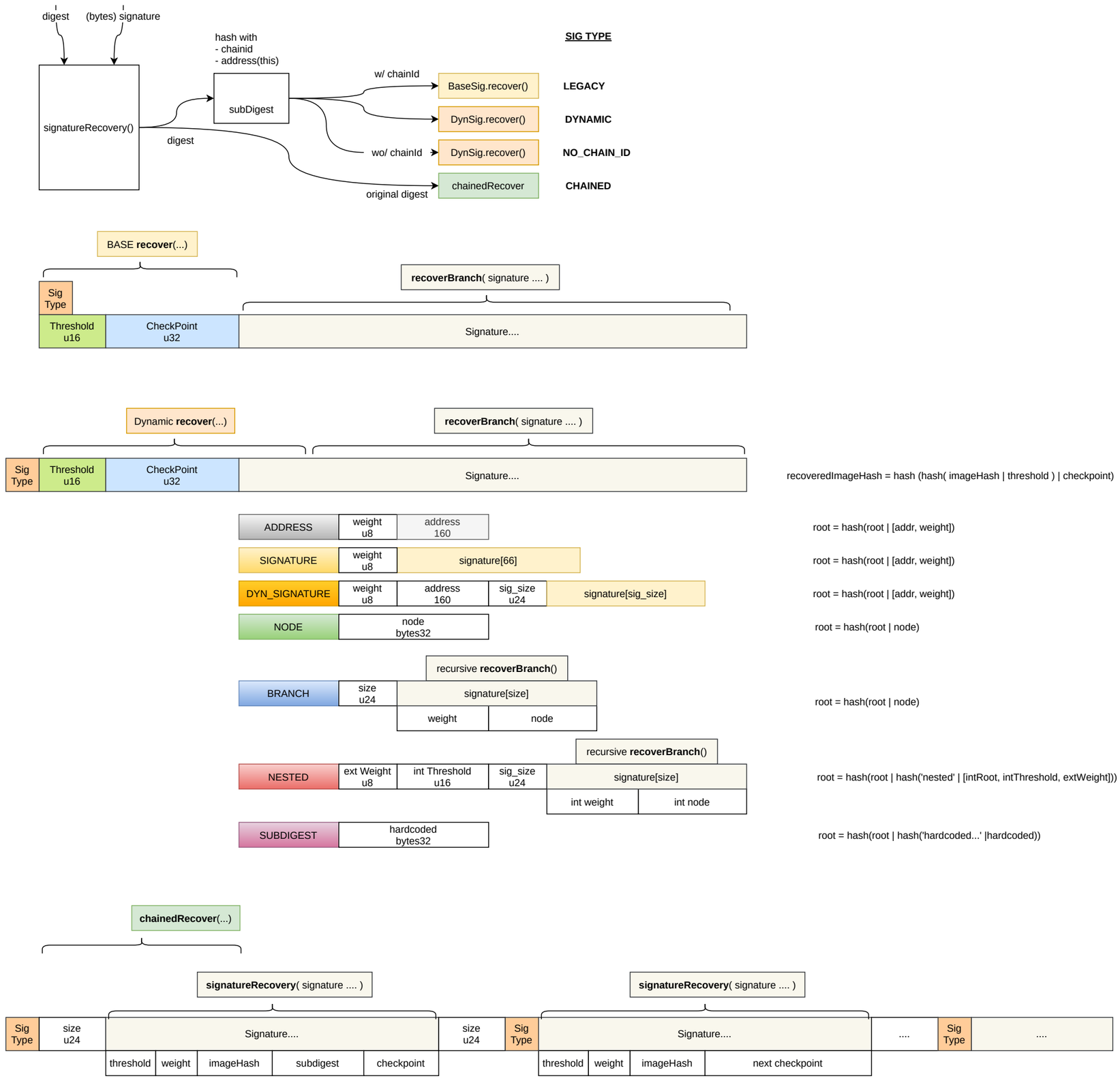
```
╔══════╤═══════════╤═══════╤═══════╤══════════╤══════════╤═══════════════╤═══════════╗
║ Step │ Loc       │ Len   │ Gas   │ Consumed │ Opcode   │ Instruction   │ Data      ║
╟──────┼───────────┼───────┼───────┼──────────┼──────────┼───────────────┼───────────╢
║    0 │ 0 (0x0)   │   2   │   3   │        3 │ 0x60     │ PUSH1         │ 0x3a  `:` ║
║    1 │ 2 (0x2)   │   2   │   3   │        6 │ 0x60     │ PUSH1         │ 0x0e      ║
║    2 │ 4 (0x4)   │   1   │   2   │        8 │ 0x3d     │ RETURNDATASIZE│           ║
║    3 │ 5 (0x5)   │   1   │   3   │       11 │ 0x39     │ CODECOPY      │           ║
║    4 │ 6 (0x6)   │   2   │   3   │       14 │ 0x60     │ PUSH1         │ 0x1a      ║
║    5 │ 8 (0x8)   │   1   │   3   │       17 │ 0x80     │ DUP1          │           ║
║    6 │ 9 (0x9)   │   1   │   3   │       20 │ 0x51     │ MLOAD         │           ║
║    7 │ 10 (0xa)  │   1   │   2   │       22 │ 0x30     │ ADDRESS       │           ║
║    8 │ 11 (0xb)  │   1   │   0   │       22 │ 0x55     │ SSTORE        │           ║
║    9 │ 12 (0xc)  │   1   │   2   │       24 │ 0x3d     │ RETURNDATASIZE│           ║
║   10 │ 13 (0xd)  │   1   │   0   │       24 │ 0xf3     │ RETURN        │           ║
║   11 │ 14 (0xe)  │   1   │   2   │       26 │ 0x36     │ CALLDATASIZE  │           ║
║   12 │ 15 (0xf)  │   1   │   2   │       28 │ 0x3d     │ RETURNDATASIZE│           ║
║   13 │ 16 (0x10) │   1   │   2   │       30 │ 0x3d     │ RETURNDATASIZE│           ║
║   14 │ 17 (0x11) │   1   │   3   │       33 │ 0x37     │ CALLDATACOPY  │           ║
║   15 │ 18 (0x12) │   1   │   2   │       35 │ 0x3d     │ RETURNDATASIZE│           ║
║   16 │ 19 (0x13) │   1   │   2   │       37 │ 0x3d     │ RETURNDATASIZE│           ║
║   17 │ 20 (0x14) │   1   │   2   │       39 │ 0x3d     │ RETURNDATASIZE│           ║
║   18 │ 21 (0x15) │   1   │   2   │       41 │ 0x36     │ CALLDATASIZE  │           ║
║   19 │ 22 (0x16) │   1   │   2   │       43 │ 0x3d     │ RETURNDATASIZE│           ║
║   20 │ 23 (0x17) │   1   │   2   │       45 │ 0x30     │ ADDRESS       │           ║
║   21 │ 24 (0x18) │   1   │ 200   │      245 │ 0x54     │ SLOAD         │           ║
║   22 │ 25 (0x19) │   1   │   2   │      247 │ 0x5a     │ GAS           │           ║
║   23 │ 26 (0x1a) │   1   │ 700   │      947 │ 0xf4     │ DELEGATECALL  │           ║
║   24 │ 27 (0x1b) │   1   │   2   │      949 │ 0x3d     │ RETURNDATASIZE│           ║
║   25 │ 28 (0x1c) │   1   │   3   │      952 │ 0x82     │ DUP3          │           ║
║   26 │ 29 (0x1d) │   1   │   3   │      955 │ 0x80     │ DUP1          │           ║
║   27 │ 30 (0x1e) │   1   │   3   │      958 │ 0x3e     │ RETURNDATACOPY│           ║
║   28 │ 31 (0x1f) │   1   │   3   │      961 │ 0x90     │ SWAP1         │           ║
║   29 │ 32 (0x20) │   1   │   2   │      963 │ 0x3d     │ RETURNDATASIZE│           ║
║   30 │ 33 (0x21) │   1   │   3   │      966 │ 0x91     │ SWAP2         │           ║
║   31 │ 34 (0x22) │   2   │   3   │      969 │ 0x60     │ PUSH1         │ 0x18      ║
║   32 │ 36 (0x24) │   1   │  10   │      979 │ 0x57     │ JUMPI         │           ║
║   33 │ 37 (0x25) │   1   │   0   │      979 │ 0xfd     │ REVERT        │           ║
║   34 │ 38 (0x26) │   1   │   1   │      980 │ 0x5b     │ JUMPDEST      │           ║
║   35 │ 39 (0x27) │   1   │   0   │      980 │ 0xf3     │ RETURN        │           ║
╚══════╧═══════════╧═══════╧═══════╧══════════╧══════════╧═══════════════╧═══════════╝
```

Here's an annotated version of the low-level Wallet proxy contract code:

```
// SPDX-License-Identifier: Apache-2.0
pragma solidity 0.8.18;

/**
    Minimal upgradeable proxy implementation, delegates all calls to the address
    defined by the storage slot matching the wallet address.

    Inspired by EIP-1167 Implementation (https://eips.ethereum.org/EIPS/eip-1167)

    deployed code:

        0x00    0x36         0x36      CALLDATASIZE      cds
        0x01    0x3d         0x3d      RETURNDATASIZE    0 cds
        0x02    0x3d         0x3d      RETURNDATASIZE    0 0 cds
        0x03    0x37         0x37      CALLDATACOPY
        0x04    0x3d         0x3d      RETURNDATASIZE    0
        0x05    0x3d         0x3d      RETURNDATASIZE    0 0
        0x06    0x3d         0x3d      RETURNDATASIZE    0 0 0
        0x07    0x36         0x36      CALLDATASIZE      cds 0 0 0
        0x08    0x3d         0x3d      RETURNDATASIZE    0 cds 0 0 0
        0x09    0x30         0x30      ADDRESS           addr 0 cds 0 0 0
        0x0A    0x54         0x54      SLOAD             imp 0 cds 0 0 0      //@audit - sload(address(this))
        0x0B    0x5a         0x5a      GAS               gas imp 0 cds 0 0 0
        0x0C    0xf4         0xf4      DELEGATECALL      suc 0                //@audit - sload(address(this)).delegatecall(...)
        0x0D    0x3d         0x3d      RETURNDATASIZE    rds suc 0
        0x0E    0x82         0x82      DUP3              0 rds suc 0
        0x0F    0x80         0x80      DUP1              0 0 rds suc 0
        0x10    0x3e         0x3e      RETURNDATACOPY    suc 0
        0x11    0x90         0x90      SWAP1             0 suc
        0x12    0x3d         0x3d      RETURNDATASIZE    rds 0 suc
        0x13    0x91         0x91      SWAP2             suc 0 rds
        0x14    0x60 0x18    0x6018    PUSH1             0x18 suc 0 rds
    /-- 0x16    0x57         0x57      JUMPI             0 rds
    |   0x17    0xfd         0xfd      REVERT
    \-> 0x18    0x5b         0x5b      JUMPDEST          0 rds
        0x19    0xf3         0xf3      RETURN

    flat deployed code: 0x363d3d373d3d3d363d30545af43d82803e903d91601857fd5bf3

    deploy function:

        0x00    0x60 0x3a    0x603a    PUSH1             0x3a
        0x02    0x60 0x0e    0x600e    PUSH1             0x0e 0x3a
        0x04    0x3d         0x3d      RETURNDATASIZE    0 0x0e 0x3a    //@audit - dataoffset=0, offset=0x0e, length=0x3a;
        0x05    0x39         0x39      CODECOPY                         //@audit - address(this).code[0x0e:0x0e+0x3a] ==> memory[0:0+0
        0x06    0x60 0x1a    0x601a    PUSH1             0x1a           //@audit -    copies "contract code" to memory
        0x08    0x80         0x80      DUP1              0x1a 0x1a
        0x09    0x51         0x51      MLOAD             imp 0x1a       //@audit - impl = memory[0x1a]   // code[0x1a+0x0e=0x28] is en
        0x0A    0x30         0x30      ADDRESS           addr imp 0x1a  //@audit - address(this)
        0x0B    0x55         0x55      SSTORE            0x1a           //@audit - sstore[address(this)] = impl
        0x0C    0x3d         0x3d      RETURNDATASIZE    0 0x1a         //@audit - 26
        0x0D    0xf3         0xf3      RETURN
        [...deployed code]

    flat deploy function: 0x603a600e3d39601a805130553df3363d3d373d3d3d363d30545af43d82803e903d91601857fd5bf3
*/
library Wallet {
  bytes internal constant creationCode = hex"603a600e3d39601a805130553df3363d3d373d3d3d363d30545af43d82803e903d91601857fd5bf3";
}
```

3.5 Signature Types

The four different signature types – LEGACY, DYNAMIC, NO_CHAIN_ID, and CHAINED – are depicted in the following diagram.



Sequence Wallet - Signature Types

4 Security Specification

4.1 Actors

The relevant actors are listed below with their respective abilities:

The *factory contract* creates new instances of the wallet on behalf of users. *Wallet owners* can send signed transactions to wallet instances that are executed if the preconfigured threshold is met. Owners can perform the following actions: execute transactions, change the implementation, add/remove function hooks that delegate execution to external code, and change the configuration (e.g., threshold, owner addresses and weights).

4.2 Trust Model

The Sequence Wallet contracts do not include any mechanisms which would require trust in a centralized administrator type of role. All actions, including code and configuration updates, are performed by the wallet owners via multisig transactions. Hence, the wallet contract system is entirely trustless from the view of the user.

Since users have complete control, they're also fully responsible and must be extremely careful what transactions they sign. While this is trivially and generally true, the following points deserve extra mention:

- Implementation upgrades switch the code that is executed in the context of the wallet instance. Changing the implementation to a flawed contract or wrong address can result in the loss of funds or "brick" the wallet.
- Hooks allow the addition of code to the core functionality of the wallet. Installing a flawed or even malicious hook can put all funds at risk.
- As mentioned above, a wallet instance's configuration is not stored on-chain explicitly; instead, a Merkle tree is built and only its root is stored in the wallet's storage. This is an elegant and efficient design, but it also comes with more risks than the traditional approach of storing the owners explicitly:
  (1) It makes the signature verification process more involved, which means there is a higher risk of bugs in the implementation. This could allow the execution of transactions that, in reality, haven't been authorized, and/or it could prevent the execution of transactions that *have* been authorized.
  (2) Changing the configuration means just storing a new Merkle root. Building the tree and computing its root is an off-chain process, though. If this code has bugs or gets compromised, funds could get stolen or frozen. In theory, users don't have to trust this off-chain code and can verify the Merkle root (or even compute it themselves) before they sign the corresponding transaction. In practice, this is hardly feasible, and many users will lack the time and/or skills to do that, so they will likely opt to trust this code. It should be noted that off-chain code, in particular the code that computes the Merkle root for a given configuration, has been not in scope for this engagement.
  (3) Finally, the "raw data" of the configuration must not be lost or forgotten. Unlike an explicitly stored list of owners that can always be recovered from on-chain data, this is not possible if we store just a hash of the configuration. If – perhaps years after setting a particular configuration – even a small part of it (such as one of the owner addresses) has been forgotten, all funds will be frozen because the witness for the Merkle tree can't be reconstructed. Hence, a good backup strategy for this data is essential.

5 Findings

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

5.1 ModuleHooks – Forced Fallback Function Execution on Hook Target and Selector Shadowing

Major ✓ Fixed

Resolution

In commit c31dade, the call forwarding logic has been wrapped in a `if (msg.data.length >= 4) { ... }`. This means that calls to the wallet will silently succeed without effect for calldata with length less than 4 bytes, while our recommendation was to revert in this case. However, the following has to be taken into account: The client explicitly wants the wallet to silently accept calldata it can't interpret, i.e., calls should succeed in this case, even though they won't have any effect. (See issue 5.5 for the client's detailed comment.) Hence, the fix for this finding is consistent with this intention.
We'd like to reiterate that the subtleties discussed in (2) could be a source of confusion and mistakes, so they should be documented.

Description

(1) `ModuleHooks` allows adding code to the wallet's core functionality. More specifically, function selectors can be registered together with a target contract address. If a call is not handled by the Solidity-generated function selector dispatcher, it will fall through to the wallet's `fallback` function, which implements an alternate high-level dispatcher. This dispatcher first calls `_readHook(bytes4)` on `msg.sig (bytes4)`, and if the method returns an address – i.e., a matching selector has been registered – it delegatecalls to the corresponding target address with `msg.data (bytes)`.

Now, `msg.sig` is right-padded to `byte4`, but `msg.data` is a variable length `bytes` array that may contain less than 4 bytes. If calldata length is less than 4, Solidity bypasses the built-in dispatcher and calls the contract's fallback function if it exists; otherwise, the call reverts. This behavior may lead to an inconsistency if a selector with trailing zero bytes has been registered (e.g., `minimumBid(uint256) ==> 0xfea32600`) but calldata length is less than 4: First, the wallet's fallback function will be called. As mentioned above, `msg.sig` is right-padded and will therefore match the registered selector, but then the delegatecall to the target is made with the original calldata that has length less than 4 bytes. Assuming the target contract has a fallback function, it will then be executed instead of the function with the registered selector.

contracts/modules/commons/ModuleHooks.sol:L107-L119

```
fallback() external payable {
  address target = _readHook(msg.sig);
  if (target != address(0)) {
    (bool success, bytes memory result) = target.delegatecall(msg.data);
    assembly {
      if iszero(success)  {
        revert(add(result, 0x20), mload(result))
      }

      return(add(result, 0x20), mload(result))
    }
  }
}
```

contracts/modules/commons/ModuleHooks.sol:L50-L57

```
/**
 * @notice Reads the implementation hook of a signature
 * @param _signature Signature function
 * @return The address of the implementation hook, address(0) if none
*/
function _readHook(bytes4 _signature) private view returns (address) {
  return address(uint160(uint256(ModuleStorage.readBytes32Map(HOOKS_KEY, _signature))));
}
```

Here's an exemplary disassembly of a basic contract's dispatcher as generated by solc v0.8.18. Unless at least 4 bytes of calldata have been supplied, function dispatching is skipped, and we jump to the fallback function (or revert if none is implemented). See the `@audit` annotations for the dispatcher checks.

```
|   21 | 32 (0x20)    |   2 |     3 |       60 | 0x60   | PUSH1          | 0x80                                                                  |
|   22 | 34 (0x22)    |   2 |     3 |       63 | 0x60   | PUSH1          | 0x40   `@`                                                            |
|   23 | 36 (0x24)    |   1 |     3 |       66 | 0x52   | MSTORE         |                                                                       |
|   24 | 37 (0x25)    |   1 |     2 |       68 | 0x34   | CALLVALUE      |                                                                       |
|   25 | 38 (0x26)    |   1 |     3 |       71 | 0x80   | DUP1           |                                                                       |
|   26 | 39 (0x27)    |   1 |     3 |       74 | 0x15   | ISZERO         |                                                                       |
|   27 | 40 (0x28)    |   3 |     3 |       77 | 0x61   | PUSH2          | 0x0010                                                                |
|   28 | 43 (0x2b)    |   1 |    10 |       87 | 0x57   | JUMPI          |                                                                       |
|   29 | 44 (0x2c)    |   2 |     3 |       90 | 0x60   | PUSH1          | 0x00                                                                  |
|   30 | 46 (0x2e)    |   1 |     3 |       93 | 0x80   | DUP1           |                                                                       |
|   31 | 47 (0x2f)    |   1 |     0 |       93 | 0xfd   | REVERT         |                                                                       |
|   32 | 48 (0x30)    |   1 |     1 |       94 | 0x5b   | JUMPDEST       |                                                                       |
|   33 | 49 (0x31)    |   1 |     2 |       96 | 0x50   | POP            |                                                                       |
|   34 | 50 (0x32)    |   2 |     3 |       99 | 0x60   | PUSH1          | 0x04                                                                  |
|   35 | 52 (0x34)    |   1 |     2 |      101 | 0x36   | CALLDATASIZE   |                                                                       |
|   36 | 53 (0x35)    |   1 |     3 |      104 | 0x10   | LT             | @audit: msg.length>0x04 -> dispatch. else, fallback                   |
|   37 | 54 (0x36)    |   3 |     3 |      107 | 0x61   | PUSH2          | 0x004f                                                                |
|   38 | 57 (0x39)    |   1 |    10 |      117 | 0x57   | JUMPI          |                                                                       |
|   39 | 58 (0x3a)    |   2 |     3 |      120 | 0x60   | PUSH1          | 0x00                                                                  |
|   40 | 60 (0x3c)    |   1 |     3 |      123 | 0x35   | CALLDATALOAD   |                                                                       |
|   41 | 61 (0x3d)    |   2 |     3 |      126 | 0x60   | PUSH1          | 0xe0                                                                  |
|   42 | 63 (0x3f)    |   1 |     3 |      129 | 0x1c   | SHR            |                                                                       |
|   43 | 64 (0x40)    |   1 |     3 |      132 | 0x80   | DUP1           | @audit: function dispatch                                             |
|   44 | 65 (0x41)    |   4 |     3 |      135 | 0x62   | PUSH3          | 0x5de6e0 → `getLotMaximumEthValue(bytes4)`
|   45 | 69 (0x45)    |   1 |     3 |      138 | 0x14   | EQ             |                                                                       |
|   46 | 70 (0x46)    |   3 |     3 |      141 | 0x61   | PUSH2          | 0x00e5                                                                |
|   47 | 73 (0x49)    |   1 |    10 |      151 | 0x57   | JUMPI          |                                                                       |
|   48 | 74 (0x4a)    |   1 |     3 |      154 | 0x80   | DUP1           |                                                                       |
|   49 | 75 (0x4b)    |   5 |     3 |      157 | 0x63   | PUSH4          | 0x0f647f7e                                                            |
|   50 | 80 (0x50)    |   1 |     3 |      160 | 0x14   | EQ             |                                                                       |
|   51 | 81 (0x51)    |   3 |     3 |      163 | 0x61   | PUSH2          | 0x00ef                                                                |
|   52 | 84 (0x54)    |   1 |    10 |      173 | 0x57   | JUMPI          |                                                                       |
|   53 | 85 (0x55)    |   1 |     3 |      176 | 0x80   | DUP1           |                                                                       |
|   54 | 86 (0x56)    |   5 |     3 |      179 | 0x63   | PUSH4          | 0x36a00093                                                            |
|   55 | 91 (0x5b)    |   1 |     3 |      182 | 0x14   | EQ             |                                                                       |
|   56 | 92 (0x5c)    |   3 |     3 |      185 | 0x61   | PUSH2          | 0x011f                                                                |
|   57 | 95 (0x5f)    |   1 |    10 |      195 | 0x57   | JUMPI          |                                                                       |
|   58 | 96 (0x60)    |   1 |     3 |      198 | 0x80   | DUP1           |                                                                       |
|   59 | 97 (0x61)    |   5 |     3 |      201 | 0x63   | PUSH4          | 0x552aaedb → `getAddress(bytes4)`                                      |
|   60 | 102 (0x66)   |   1 |     3 |      204 | 0x14   | EQ             |                                                                       |
|   61 | 103 (0x67)   |   3 |     3 |      207 | 0x61   | PUSH2          | 0x013b                                                                |
|   62 | 106 (0x6a)   |   1 |    10 |      217 | 0x57   | JUMPI          |                                                                       |
|   63 | 107 (0x6b)   |   3 |     3 |      220 | 0x61   | PUSH2          | 0x0050                                                                |
|   64 | 110 (0x6e)   |   1 |     8 |      228 | 0x56   | JUMP           |                                                                       |
|   65 | 111 (0x6f)   |   1 |     1 |      229 | 0x5b   | JUMPDEST       |                                                                       |
|   66 | 112 (0x70)   |   1 |     1 |      230 | 0x5b   | JUMPDEST       |
```

(2) It should be noted that the module allows setting hooks on function selectors that are also present in the wallet's core code (direct or inherited). Such hooks will never be executed as the Solidity implementation will have priority over the `fallback`-dispatched calls. We would also like to point out that this has the potential to lead to surprising behavior in the context of implementation upgrades: The new implementation might shadow hooks that were previously accessible or might activate hooks that were previously masked by the built-in dispatcher. Finally, it is important to remember that different function signatures may have the same selector, i.e., clashes may not be visible on the level of signatures. In other words, even if a hook target's source code only contains functions whose signature does not occur in the wallet code, a function in the hook target could still have the same selector as a function in the wallet core and therefore be shadowed by the "built-in function".

Recommendation

- (1) Add a calldata length check to the wallet's fallback function, and revert if calldata length is not at least 4.
- (2) Describe the subtleties of function selector shadowing/masking in the general documentation and the source code. Users must exercise caution and take the points above into consideration before signing transactions.

5.2 ModuleHooks – Check for Contract Existence Before Executing Hook Medium Won't Fix

Resolution

Client's comment:

> We consider this a non-issue, prioritizing slim and efficient code over adding checks for non-ops that introduce complexity and gas costs.

Description

`ModuleHooks` allows the wallet to set hooks on function signatures handled in the `fallback` function. However, the second-level function dispatcher does not ensure that the `delegatecall` target contract exists before the call is made. This might lead to the hook being executed on an address that does not contain code – resulting in the `delegatecall` always returning success and hiding that the target contract did not exist and no code was executed.

Examples

contracts/modules/commons/ModuleHooks.sol:L28-L37

```
/**
 * @notice Adds a new hook to handle a given function selector
 * @param _signature Signature function linked to the hook
 * @param _implementation Hook implementation contract
 * @dev Can't overwrite hooks that are part of the main module (those defined below)
 */
function addHook(bytes4 _signature, address _implementation) external override virtual onlySelf {
  if (_readHook(_signature) != address(0)) revert HookAlreadyExists(_signature);
  _writeHook(_signature, _implementation);
}
```

Recommendation

contracts/utils/LibAddress.sol:L12-L17

```
function isContract(address account) internal view returns (bool) {
  uint256 csize;
  // solhint-disable-next-line no-inline-assembly
  assembly { csize := extcodesize(account) }
  return csize != 0;
}
```

Before the `delegatecall` is made, use the `isContract` implementation from `LibAddress` to ensure the hook target address is a contract. Consider making the same check when the hook is added to catch mistakes early.

5.3 Factory – Check for Module Existence Before Deploying Wallet Medium Won't Fix

Resolution

Client's comment:

> We consider this a non-issue, as a proxy with a non-deployed `mainModule` may still be valid if the implementation is deployed later. Moreover, maintaining efficiency is crucial in this critical code section.

Description

Deploying a new wallet contract is done via the `Factory.deploy` method. This method takes a `salt` and a target `_mainModule`, where the `_mainModule` is the target address the minimal proxy delegatecalls to.

If the factory is used with a `_mainModule` pointing to an account that has no code, delegatecalls will succeed without error but not execute any code. This might stay unnoticed until a user actually checks the state of their wallet proxy.

Examples

contracts/Factory.sol:L17-L20

```
function deploy(address _mainModule, bytes32 _salt) public payable returns (address _contract) {
  bytes memory code = abi.encodePacked(Wallet.creationCode, uint256(uint160(_mainModule)));
  assembly { _contract := create2(callvalue(), add(code, 32), mload(code), _salt) }
}
```

Recommendation

In `Factory.deploy`, use the `isContract` implementation from `LibAddress` to ensure that the address at `_mainModule` is a contract.

contracts/utils/LibAddress.sol:L12-L17

```
function isContract(address account) internal view returns (bool) {
  uint256 csize;
  // solhint-disable-next-line no-inline-assembly
  assembly { csize := extcodesize(account) }
  return csize != 0;
}
```

5.4 MultiCallUtils – `callBlockNumber` Returns Gas Limit Instead of Block Number Medium ✓ Fixed

Resolution

Fixed in commit 37e6e65: `callBlockNumber` now returns the correct value.

Description

The contract `MultiCallUtils` contains a collection of `view` functions that provide access to block and transaction data. The function `callBlockNumber` erroneously returns the current gas limit instead of the block number:

contracts/modules/utils/MultiCallUtils.sol:L59-L65

```
function callGasLimit() external view returns (uint256) {
  return block.gaslimit;
}

function callBlockNumber() external view returns (uint256) {
  return block.gaslimit;
}
```

Recommendation

In `callBlockNumber`, replace `block.gaslimit` with `block.number`.

Moreover, several functions in this contract can be simplified by utilizing `<address>.code`, `<address>.codehash`, or `block.chainid` instead of inline assembly.

5.5 ModuleHooks – Fallback Function Doesn't Revert if No Matching Hook Found Minor Won't Fix

Resolution

Client's comment:

> This is the expected behavior, as our wallets are designed to resemble EOA wallets closely. EOA wallets don't fail transactions with arbitrary data, so we intentionally mimic that behavior to maintain consistency and user experience.

Description

The dispatcher Solidity inserts into the code of a contract roughly works as follows: If non-empty calldata is supplied and none of the functions defined in the contract is a match, then the fallback function is executed if there is one; otherwise, the call reverts. Very often, contracts don't have or need a fallback function, so the typical behavior for calldata that the contract can't interpret is reverting. And that makes sense: Sending data to a contract it doesn't know how to interpret is likely a mistake, so reverting seems reasonable. (It should be noted, though, that sending data to an account that doesn't have any code will succeed, which is a frequent source of mistakes and confusion.)

The hooks functionality in `ModuleHooks` allows to dynamically add and remove functions to the contract, so to speak. This is achieved through a fallback function; as outlined above, this fallback function takes over if none of the "built-in" functions is a match for the supplied calldata. Much like the dispatcher generated by the compiler, the fallback function then checks if there's a hook that matches the function signature in the calldata.

contracts/modules/commons/ModuleHooks.sol:L104-L119

```
/**
 * @notice Routes fallback calls through hooks
 */
fallback() external payable {
  address target = _readHook(msg.sig);
  if (target != address(0)) {
    (bool success, bytes memory result) = target.delegatecall(msg.data);
    assembly {
      if iszero(success)  {
        revert(add(result, 0x20), mload(result))
      }

      return(add(result, 0x20), mload(result))
    }
  }
}
```

As can be seen, if no matching hook is found, the fallback function doesn't revert; it silently succeeds instead.

Recommendation

We recommend having the fallback function revert if no matching hook is found to inform the caller that the calldata could not be interpreted as a function call on the core contract or any of the installed hooks. If the current behavior is intentional and you want to keep it, it should be clearly documented that calls to unhandled functions succeed silently.

5.6 SignatureValidator – `isValidSignature` Should Check Signature's Length Minor ✓ Fixed

Resolution

Fixed in commit 7f3acaa: `isValidSignature` now checks the signature's length and reverts if it is 0.

Description

The `isValidSignature` function in the `SignatureValidator` library reads the last byte of the signature to determine the signature type:

contracts/utils/SignatureValidator.sol:L112-L117

```
function isValidSignature(
  bytes32 _hash,
  address _signer,
  bytes calldata _signature
) internal view returns (bool valid) {
  uint256 signatureType = uint8(_signature[_signature.length - 1]);
```

If the `_signature` argument has length 0, then the underflowing subtraction will create a Panic. According to the Solidity documentation, this should not happen:

> Properly functioning code should never create a Panic, not even on invalid external input. If this happens, then there is a bug in your contract which you should fix.

Recommendation

In `isValidSignature`, first check if the signature length is 0 and revert if true.

5.7 ModuleHooks – Adding and Removing Hooks Should Emit an Event Minor ✓ Fixed

Resolution

Fixed in commit 60da59d: An event is now emitted in `_writeHook`. A no-op in `addHook` is still possible and emits an event, too.

Description

Essential state-changing functions should emit an event to have an audit trail and enable monitoring of smart contract usage.

Examples

contracts/modules/commons/ModuleHooks.sol:L34-L66

```
function addHook(bytes4 _signature, address _implementation) external override virtual onlySelf {
  if (_readHook(_signature) != address(0)) revert HookAlreadyExists(_signature);
  _writeHook(_signature, _implementation);
}

/**
 * @notice Removes a registered hook
 * @param _signature Signature function linked to the hook
 * @dev Can't remove hooks that are part of the main module (those defined below)
 *      without upgrading the wallet
 */
function removeHook(bytes4 _signature) external override virtual onlySelf {
  if (_readHook(_signature) == address(0)) revert HookDoesNotExist(_signature);
  _writeHook(_signature, address(0));
}

/**
 * @notice Reads the implementation hook of a signature
 * @param _signature Signature function
 * @return The address of the implementation hook, address(0) if none
*/
function _readHook(bytes4 _signature) private view returns (address) {
  return address(uint160(uint256(ModuleStorage.readBytes32Map(HOOKS_KEY, _signature))));
}

/**
 * @notice Writes the implementation hook of a signature
 * @param _signature Signature function
 * @param _implementation Hook implementation contract
*/
function _writeHook(bytes4 _signature, address _implementation) private {
  ModuleStorage.writeBytes32Map(HOOKS_KEY, _signature, bytes32(uint256(uint160(_implementation))));
}
```

Recommendation

Emit events for `addHook` and `removeHook` or within `_writeHook`. In particular for the former approach, also consider reverting in `addHook` if `_implementation` is `address(0)` to avoid emitting an event that indicates a hook was added if that's not actually the case. This might make sense even for the second variant to avoid emitting an event in case of a no-op.

5.8 Interfaces Unexpectedly Implement Code and Declare Internal Functions Minor Acknowledged

Resolution

Client's comment:

> We agree with the assessment regarding the unexpected implementation and internal function declarations in the interface files. However, given that the project has undergone multiple audits, we prefer to avoid making significant changes to the code structure. This decision helps us minimize the risk of introducing new unknown bugs in the process, ensuring the stability and reliability of the project.

Description

According to their naming pattern (`I<module_name>`) and location in the `interfaces` subdirectory, `IModuleAuth` and `IModuleUpdate` should be interface declarations. However, these source units are `abstract` contracts instead of `interface` types and unexpectedly *declare internal functions* or contain concrete *implementations* of functions (even though the return value is hardcoded).

Based on the contract name and filename (prefix `I`) as well as the filesystem location (`./interfaces/...`), one would assume that the source units contain external interface declarations only. Finding concrete implementations or internal abstract methods is highly unexpected.

Examples

- Implementation in an interface file

contracts/modules/commons/interfaces/IModuleAuth.sol:L34-L41

```
/**
 * @notice Validates the signature image
 * @return true if the signature image is valid
 */
function _isValidImage(bytes32) internal virtual view returns (bool) {
  return false;
}
```

- Declaration of internal function in an interface file

contracts/modules/commons/interfaces/IModuleAuth.sol:L48-L52

```
/**
 * @notice Updates the signers configuration of the wallet
 * @param _imageHash New required image hash of the signature
 */
function _updateImageHash(bytes32 _imageHash) internal virtual;
```

contracts/modules/commons/interfaces/IModuleUpdate.sol:L16-L21

```
/**
 * @notice Updates the implementation of the base wallet, used internally.
 * @param _implementation New main module implementation
 * @dev WARNING Updating the implementation can brick the wallet
 */
function _updateImplementation(address _implementation) internal virtual;
```

Recommendation

It is recommended to refactor the `abstract` 'interface' contracts, moving the internal declarations and concrete function implementations to the respective implementation source unit. The `interfaces` subfolder should only contain external interface declarations (and types, etc.) without executable code or internal function declarations.

5.9 ModuleAuthConvenience, ModuleExtraAuth – Missing Export of `interfaceId` via `supportsInterface` Minor ✓ Fixed

Resolution

Fixed in commit 3090402: `ModuleAuthConvenience` and `ModuleExtraAuth` now export their `interfaceId` via the `supportsInterface` method.

Description and Recommendation

All top-level submodules (see System Overview for a graphical representation) except `ModuleAuthConvenience` and `ModuleExtraAuth` export their `interfaceId` via the `supportsInterface` method. This appears to be an oversight. It is recommended to override the `supportsInterface` method in these two contracts to export the `interfaceId` in the same way as other modules (e.g., `ModuleCalls`).

5.10 Factory, ModuleCreator – Check for Successful Deployment Minor ✓ Fixed

Resolution

Fixed in 05e81ac and a1d9496: Both `Factory` and `ModuleCreator` now check whether the deployment was successful. No event was added to the `Factory` to indicate a successful wallet deployment.

Description

Deploying a new wallet contract is done via the `Factory.deploy` method. This method calls `create2` in an `assembly` block and returns the address.

contracts/Factory.sol:L17-L20

```
function deploy(address _mainModule, bytes32 _salt) public payable returns (address _contract) {
  bytes memory code = abi.encodePacked(Wallet.creationCode, uint256(uint160(_mainModule)));
  assembly { _contract := create2(callvalue(), add(code, 32), mload(code), _salt) }
}
```

If the `create2` call succeeds, `deploy` returns the address the wallet was deployed to. If, however, deployment fails, `create2` returns `0` – and the `deploy` function simply returns this value. This could happen, for example, because a `_salt` that has already been used is tried again.

Similarly, `ModuleCreator` – which allows the wallet to deploy contracts – does not check if the deployment was successful:

contracts/modules/commons/ModuleCreator.sol:L13-L21

```
/**
 * @notice Creates a contract forwarding eth value
 * @param _code Creation code of the contract
 * @return addr The address of the created contract
 */
function createContract(bytes memory _code) public override virtual payable onlySelf returns (address addr) {
  assembly { addr := create(callvalue(), add(_code, 32), mload(_code)) }
  emit CreatedContract(addr);
}
```

Recommendation

It is standard practice in smart contract programming to revert if an operation could not be executed successfully. We recommend adding a check that `_contract` is not equal to `address(0)` at the end of the Factory's `deploy` function to make sure

failing deployments will be noticed because the transaction reverts. Also consider emitting an event for a successful deployment; see issue 5.7.

Analogously, check for a successful deployment in `ModuleCreator.createContract`.

5.11 Unused Imports Minor ✓ Fixed

Resolution
The unused imports listed in the “Examples” section below were removed in commit f17d829. We have not checked if there are other unused imports left.

Description

Throughout the codebase, there are several occurrences of unused imports. These lines can safely be removed to increase readability and maintainability.

Examples

- GuestModule.sol

contracts/modules/GuestModule.sol:L5

```
import "../utils/SignatureValidator.sol";
```

contracts/modules/GuestModule.sol:L10

```
import "./commons/Implementation.sol";
```

contracts/modules/GuestModule.sol:L12

```
import "./commons/ModuleHooks.sol";
```

contracts/modules/GuestModule.sol:L14

```
import "./commons/ModuleUpdate.sol";
```

contracts/modules/GuestModule.sol:L17-L20

```
import "../interfaces/receivers/IERC1155Receiver.sol";
import "../interfaces/receivers/IERC721Receiver.sol";

import "../interfaces/IERC1271Wallet.sol";
```

- MainModule.sol

contracts/modules/MainModule.sol:L5

```
import "../utils/SignatureValidator.sol";
```

contracts/modules/MainModule.sol:L7

```
import "./commons/Implementation.sol";
```

contracts/modules/MainModule.sol:L15-L19

```
import "../interfaces/receivers/IERC1155Receiver.sol";
import "../interfaces/receivers/IERC721Receiver.sol";

import "../interfaces/IERC1271Wallet.sol";
```

- Other

contracts/modules/commons/ModuleAuth.sol:L5

```
import "../../utils/SignatureValidator.sol";
```

contracts/modules/commons/submodules/auth/SequenceChainedSig.sol:L5

```
import "./SequenceNoChainIdSig.sol";
```

There are likely more of these.

Recommendation

We recommend checking each import and removing the ones which are not necessary.

5.12 RequireUtils – Code Duplication _decodeNonce ✓ Fixed

Resolution

Fixed in commit 4bb2f8b: The duplicated functionality was removed from the `RequireUtils` contract, which now utilizes the `decodeNonce` function from the `SubModuleNonce` library.

Description and Recommendation

`RequireUtils._decodeNonce` reimplements `SubmoduleNonce.decodeNonce` while it could just call the original implementation in the library `SubmoduleNonce` instead. Consider deduplicating code to avoid maintaining multiple implementations of the same code. Directly call `SubModuleNonce.decodeNonce` instead of reimplementing.

contracts/modules/utils/RequireUtils.sol:L34-L44

```
 * @notice Decodes a raw nonce
 * @dev A raw nonce is encoded using the first 160 bits for the space
 *  and the last 96 bits for the nonce
 * @param _rawNonce Nonce to be decoded
 * @return _space The nonce space of the raw nonce
 * @return _nonce The nonce of the raw nonce
 */
function _decodeNonce(uint256 _rawNonce) private pure returns (uint256 _space, uint256 _nonce) {
  _nonce = uint256(bytes32(_rawNonce) & SubModuleNonce.NONCE_MASK);
  _space = _rawNonce >> SubModuleNonce.NONCE_BITS;
}
```

contracts/modules/commons/submodules/nonce/SubModuleNonce.sol:L20-L29

```
function decodeNonce(uint256 _rawNonce) internal pure returns (
  uint256 _space,
  uint256 _nonce
) {
  unchecked {
    // Decode nonce
    _space = _rawNonce >> NONCE_BITS;
    _nonce = uint256(bytes32(_rawNonce) & NONCE_MASK);
  }
}
```

The `unchecked` is not necessary and can be removed.

5.13 Modules That Are Not Deployed on Chain and Only Used for Gas Estimation Should Be Moved to a Dedicated Subfolder ✓ Fixed

Resolution

Fixed in commits 5174921 and 03048d2: `ModuleIgnoreAuthUpgradable` and `ModuleIgnoreNonceCalls` were moved to a dedicated subfolder.

Description

Modules like `ModuleIgnoreAuthUpgradable` and `ModuleIgnoreNonceCalls` are only used for off-chain gas estimation and never deployed on-chain. These modules always return `true` on image and nonce verification which can be fatal if used on-chain.

To further distinct their usage from standard submodules, it is recommended to move them to a dedicated subfolder (e.g. `contracts/modules/gas-estimation`).

```
.                              ModuleAuthFixed.sol            ModuleERC5719.sol              **ModuleIgnoreNonceCalls.sol     ModuleUpdate.sol
..                             ModuleAuthUpgradable.sol       ModuleExtraAuth.sol            ModuleNonce.sol                interfaces
Implementation.sol             ModuleCalls.sol                ModuleHooks.sol                ModuleOnlyDelegatecall.sol     submodules
ModuleAuth.sol                 ModuleCreator.sol              ModuleIPFS.sol                 ModuleSelfAuth.sol
ModuleAuthConvenience.sol      ModuleERC165.sol               **ModuleIgnoreAuthUpgradable.sol ModuleStorage.sol
```

The GasEstimator module inherits the following contracts:

contracts/modules/MainModuleGasEstimation.sol:L12-L24

```
/**
 * @notice Contains an alternative implementation of the MainModules that skips validation of
 *   signatures, this implementation SHOULD NOT be used directly on a wallet.
 *
 *   Intended to be used only for gas estimation, using eth_call and overrides.
 */
contract MainModuleGasEstimation is
  ModuleIgnoreAuthUpgradable,
  ModuleIgnoreNonceCalls,
  ModuleUpdate,
  ModuleHooks,
  ModuleCreator
{
```

The following two skip nonce and imagehash validation and should never be used on-chain:

contracts/modules/commons/ModuleIgnoreAuthUpgradable.sol:L8-L21

```
  @notice Implements ModuleAuthUpgradable but ignores the validity of the signature
    should only be used during gas estimation.
*/
abstract contract ModuleIgnoreAuthUpgradable is ModuleAuthUpgradable {
  /**
   * @notice Removes the signature validation from the module, by returning true for any _imageHash
   * @param _imageHash Hash image of signature
   * @return true always
   */
  function _isValidImage(bytes32 _imageHash) internal override(ModuleAuthUpgradable) virtual view returns (bool) {
    // Still validates the imageHash using the original mechanism for a more acurate estimation
    return super._isValidImage(_imageHash) || true;
  }
}
```

contracts/modules/commons/ModuleIgnoreNonceCalls.sol:L14-L27

```
abstract contract ModuleIgnoreNonceCalls is ModuleCalls {

  /**
   * @notice Verify if a nonce is valid
   * @param _rawNonce Nonce to validate (may contain an encoded space)
   */
  function _validateNonce(uint256 _rawNonce) internal override virtual {
    // Retrieve current nonce for this wallet
    (uint256 space, uint256 providedNonce) = SubModuleNonce.decodeNonce(_rawNonce);

    uint256 currentNonce = readNonce(space);
    if (currentNonce != providedNonce && false) {
      revert BadNonce(space, providedNonce, currentNonce);
    }
```

5.14 Unnecessary and Outdated Pragma Directive ✓ Fixed

Resolution

Fixed in commits 03048d2 and 5174921: All occurrences of `pragma experimental ABIEncoderV2;` were removed.

Description

Many source files use the pragma directive `pragma experimental ABIEncoderV2;`. ABI coder V2 is the default since Solidity v0.8.0 and is considered non-experimental as of Solidity v0.6.0. Hence, this directive is not necessary and even a bit misleading because the "experimental" status was removed long ago.

Examples

contracts/modules/MainModule.sol:L3

```
pragma experimental ABIEncoderV2;
```

contracts/modules/commons/ModuleCalls.sol:L3

```
pragma experimental ABIEncoderV2;
```

contracts/modules/utils/MultiCallUtils.sol:L3

```
pragma experimental ABIEncoderV2;
```

Recommendation

We recommend deleting this line in all source files that have it. If you want to be explicit for some reason, it should be replaced with `pragma abicoder v2;`.

5.15 SignatureValidator – Unnecessarily Wide Datatype `SIG_TYPE_*` Won't Fix

Resolution

Client's comment:

> We acknowledge the concerns raised about the unnecessarily wide datatype for the signature types in SignatureValidator. While using an enum (uint8) would be more suitable and efficient, we have decided not to make changes at this stage. The current implementation does not significantly impact the overall performance, and making changes might introduce new risks or issues.

Description

The signature type is appended to the signature byte sequence and is a `uint8`. The contract, however, uses `uint256` to represent the various signature types. This appears unnecessary as the signature type enumeration starts at 1 and ends at 3 and can never use the full `uint256` width.

contracts/utils/SignatureValidator.sol:L34-L36

```
uint256 private constant SIG_TYPE_EIP712 = 1;
uint256 private constant SIG_TYPE_ETH_SIGN = 2;
uint256 private constant SIG_TYPE_WALLET_BYTES32 = 3;
```

contracts/utils/SignatureValidator.sol:L48-L54

```
function recoverSigner(
  bytes32 _hash,
  bytes calldata _signature
) internal pure returns (address signer) {
  if (_signature.length != 66) revert InvalidSignatureLength(_signature);
  uint256 signatureType = _signature.readUint8(_signature.length - 1);
```

Consider using an `enum` (`uint8`) instead of `uint256`. This would also allow casting the type byte directly to the `enum` type, which is implicitly bounds-checked.

Additionally, the call to `_signature.readUint8` may cost more gas than just casting the byte directly like here:

contracts/utils/SignatureValidator.sol:L117

```
uint256 signatureType = uint8(_signature[_signature.length - 1]);
```

In any case, the same method should be used consistently.

Appendix 1 - Files in Scope

This audit covered the following files:

Original version `0928bf2bf471acd367b60bd629076b1b8628e4f2`:

File	SHA-1 hash
`contracts/Factory.sol`	`84cab2e7bef85288dcf9bc2c3ca2d32e9e363547`
`contracts/Wallet.sol`	`ceb56f84327cd8457a5ad72b42c94da70ec00d84`
`contracts/interfaces/IERC1271Wallet.sol`	`4b4b7eb87cef901e61efa29adc9fb3f73854656b`
`contracts/interfaces/receivers/IERC1155Receiver.sol`	`45cd65106bac542088e983afd8eda5d04585b388`
`contracts/interfaces/receivers/IERC223Receiver.sol`	`4bacd22391761cf429aa100c48dd5a48739e7d7a`
`contracts/interfaces/receivers/IERC721Receiver.sol`	`af6d0ac3934dbc21d5544f790ef041137640c2ae`
`contracts/modules/GuestModule.sol`	`5bc41420d81a8337a5bd791c9186995bd00cca16`
`contracts/modules/MainModule.sol`	`eb5b3907e4ba20819785ef61bb6c808efdda456c`
`contracts/modules/MainModuleGasEstimation.sol`	`b4ed0da45c202c22e27e2fff911bd0b232b0ffc3`
`contracts/modules/MainModuleUpgradable.sol`	`aeef28a12430cc595c8c0a87540fd75d4f6d1433`
`contracts/modules/commons/Implementation.sol`	`cc8cc16fc3f0a2ef879212d8fd6def0f18586d8d`
`contracts/modules/commons/ModuleAuth.sol`	`bde596d9e605472e97f04c86ef6300ab5ca9c3ae`
`contracts/modules/commons/ModuleAuthConvenience.sol`	`26bae570cd3a2a4cbfaa1a220c19481f6e8ea963`
`contracts/modules/commons/ModuleAuthFixed.sol`	`cca73b04618c0916a2079b42dd5035c72ad403f0`
`contracts/modules/commons/ModuleAuthUpgradable.sol`	`dc2d0ca7fd34afc10150937ce3a90b059d0c6dad`
`contracts/modules/commons/ModuleCalls.sol`	`d3384cd4c44e00812020a90c8026352ce4ad7c5f`
`contracts/modules/commons/ModuleCreator.sol`	`d718b0f37152accdfbd6516c7cd42bb667db7ac9`
`contracts/modules/commons/ModuleERC165.sol`	`679fc3d4d9e32a6500d8187e57ba118981e8ca01`
`contracts/modules/commons/ModuleERC5719.sol`	`d45f08dd25090d2443ffd1e0f2af10cf399e07a7`
`contracts/modules/commons/ModuleExtraAuth.sol`	`7fc0fe5530816dd9f3c17fc73b4ccc375a3d6392`
`contracts/modules/commons/ModuleHooks.sol`	`e4531457440b8b27803b487543bc79a809a83609`
`contracts/modules/commons/ModuleIPFS.sol`	`70610fd7110bfee3e92d398af1d3c3c7879d5b66`
`contracts/modules/commons/ModuleIgnoreAuthUpgradable.sol`	`6676820bdabccef6c96bdf7dda45c4ae7d02b194`
`contracts/modules/commons/ModuleIgnoreNonceCalls.sol`	`5e63235075d51a2363a422d10523399ed4f0ca65`
`contracts/modules/commons/ModuleNonce.sol`	`e51f182f4ff455cefd98a07ce7e302813ff60cac`
`contracts/modules/commons/ModuleOnlyDelegatecall.sol`	`931a04274331625a8e1a877268a1ff3509b52ea3`
`contracts/modules/commons/ModuleSelfAuth.sol`	`feb6c64ec913ed22e969edcf30b0e4bd3978b672`
`contracts/modules/commons/ModuleStorage.sol`	`ceadbaf46c2a8c06fbd0940bb76842932b131da4`
`contracts/modules/commons/ModuleUpdate.sol`	`dca454ec1eeddf0e7c539881bc70d2ee3a235d35`
`contracts/modules/commons/interfaces/IModuleAuth.sol`	`e0c549749df90df93fee01232705afe50184adc8`
`contracts/modules/commons/interfaces/IModuleAuthUpgradable.sol`	`a0a84b4efcf27fa5a2b7e1924c8fd0ab54899806`

File	SHA-1 hash
`contracts/modules/commons/interfaces/IModuleCalls.sol`	`bf6978f373c991c7df0e0acf6acf718768d5e634`
`contracts/modules/commons/interfaces/IModuleCreator.sol`	`b7133f651295f2b9c64d910440dc0bae59a15e16`
`contracts/modules/commons/interfaces/IModuleHooks.sol`	`49bfb951e74628710fb6e4dab1485a1dd391cea4`
`contracts/modules/commons/interfaces/IModuleUpdate.sol`	`c3e4051ce5b060fee90c50f1bd66d21721794d6c`
`contracts/modules/commons/submodules/auth/SequenceBaseSig.sol`	`827c130c56583f0de8638ee3cfa7bd25a6cfffb0`
`contracts/modules/commons/submodules/auth/SequenceChainedSig.sol`	`18d22a1d133926dc4e3baddeffd65977576201a6`
`contracts/modules/commons/submodules/auth/SequenceDynamicSig.sol`	`1e92bf01cdbf01d3bbc96b6673182e2c3786224e`
`contracts/modules/commons/submodules/auth/SequenceNoChainIdSig.sol`	`9c9a8db7ab0bd1779c67ed7afdb71ee36f9f93da`
`contracts/modules/commons/submodules/nonce/SubModuleNonce.sol`	`0e931f2ff6fac34a1657b87841667882e0513adc`
`contracts/modules/utils/GasEstimator.sol`	`cb5a719321cdf120284af321fc3d6e4d67d32115`
`contracts/modules/utils/MultiCallUtils.sol`	`8ea1ae6901e5fb06a171d7553be74d8c6600ad48`
`contracts/modules/utils/RequireUtils.sol`	`6faea7015c8b3fddad95a77fbec4727430b40a8d`
`contracts/modules/utils/SequenceUtils.sol`	`c36c43e11d7e1eda444ae133057bfeb2c23806dd`
`contracts/utils/LibAddress.sol`	`a4e2f5fc2d96af45259716e883b3720dba734490`
`contracts/utils/LibBytes.sol`	`9158548a92d531ea3d7c8e28a40091d263fa9662`
`contracts/utils/LibBytesPointer.sol`	`636d632ebc29ff39b88bbb69a632c8dce9447b35`
`contracts/utils/LibOptim.sol`	`bf95b13a6f01e306819b1ec9f39c6af05d0a667e`
`contracts/utils/LibString.sol`	`10416699728dbb9c08357d806ce3023d58939222`
`contracts/utils/SignatureValidator.sol`	`34b3d81b180bd29fed8b4ef58da77ddb3736d69c`

Revised version `a1d94968d92f153196ed715ce9294be895ca8a9e`:

File	SHA-1 hash
`contracts/Factory.sol`	`343924162a854a1e83d992560e70b0900884fde6`
`contracts/Wallet.sol`	`ceb56f84327cd8457a5ad72b42c94da70ec00d84`
`contracts/interfaces/IERC1271Wallet.sol`	`4b4b7eb87cef901e61efa29adc9fb3f73854656b`
`contracts/interfaces/receivers/IERC1155Receiver.sol`	`45cd65106bac542088e983afd8eda5d04585b388`
`contracts/interfaces/receivers/IERC223Receiver.sol`	`4bacd22391761cf429aa100c48dd5a48739e7d7a`
`contracts/interfaces/receivers/IERC721Receiver.sol`	`af6d0ac3934dbc21d5544f790ef041137640c2ae`
`contracts/modules/GuestModule.sol`	`950aaf0cc6ea052f13f62a8000eb68eed3a6c68d`
`contracts/modules/MainModule.sol`	`27f492fa6177a59c5a44a5abae8c96b02654bd91`
`contracts/modules/MainModuleGasEstimation.sol`	`9df9e82dbcd68ccc6bc7bef5f480f4555b2cf1fb`
`contracts/modules/MainModuleUpgradable.sol`	`57b2ce4abba07daa61f23f09ce29ee94085daf18`
`contracts/modules/commons/Implementation.sol`	`cc8cc16fc3f0a2ef879212d8fd6def0f18586d8d`
`contracts/modules/commons/ModuleAuth.sol`	`361638980b0fa04bd7a3101bd734ba6d61229914`
`contracts/modules/commons/ModuleAuthConvenience.sol`	`f59d10bd3c31e644ddd05b8f585d6b3f2e93d2f2`
`contracts/modules/commons/ModuleAuthFixed.sol`	`cca73b04618c0916a2079b42dd5035c72ad403f0`
`contracts/modules/commons/ModuleAuthUpgradable.sol`	`dc2d0ca7fd34afc10150937ce3a90b059d0c6dad`
`contracts/modules/commons/ModuleCalls.sol`	`4b86f53296cc80b752576074cbb1bbe095be8a4d`
`contracts/modules/commons/ModuleCreator.sol`	`3bc37e177a8cffbaaa0e1497f6027b9b9a84af33`
`contracts/modules/commons/ModuleERC165.sol`	`50244b3b2a33d78b72ae8830cd686565f19cccf6`
`contracts/modules/commons/ModuleERC5719.sol`	`d45f08dd25090d2443ffd1e0f2af10cf399e07a7`
`contracts/modules/commons/ModuleExtraAuth.sol`	`a9aa5a930eeaa93806ae4043d0092fcc40a8e698`
`contracts/modules/commons/ModuleHooks.sol`	`461a6570921b2a23941901360fbad1d54e74ca3b`
`contracts/modules/commons/ModuleIPFS.sol`	`70610fd7110bfee3e92d398af1d3c3c7879d5b66`
`contracts/modules/commons/ModuleNonce.sol`	`07aee71b81a902c791d020ddd7dc39491a3b0a41`
`contracts/modules/commons/ModuleOnlyDelegatecall.sol`	`80e2f11d38957aa85042e7641136676ed9fd84e1`
`contracts/modules/commons/ModuleSelfAuth.sol`	`feb6c64ec913ed22e969edcf30b0e4bd3978b672`
`contracts/modules/commons/ModuleStorage.sol`	`ceadbaf46c2a8c06fbd0940bb76842932b131da4`
`contracts/modules/commons/ModuleUpdate.sol`	`dca454ec1eeddf0e7c539881bc70d2ee3a235d35`
`contracts/modules/commons/gas-estimation/ModuleIgnoreAuthUpgradable.sol`	`44fedf0545e7d04a32172518666ef15817e785b4`
`contracts/modules/commons/gas-estimation/ModuleIgnoreNonceCalls.sol`	`9b179c081ee51355fb5c035a7ab418234eb64c3a`
`contracts/modules/commons/interfaces/IModuleAuth.sol`	`e0c549749df90df93fee01232705afe50184adc8`
`contracts/modules/commons/interfaces/IModuleAuthUpgradable.sol`	`a0a84b4efcf27fa5a2b7e1924c8fd0ab54899806`

File	SHA-1 hash
contracts/modules/commons/interfaces/IModuleCalls.sol	c158a054b9bd568f4991db9a1da266149de1c028
contracts/modules/commons/interfaces/IModuleCreator.sol	492fbe820693758489dd52c34c51a5f3a9a6fa0e
contracts/modules/commons/interfaces/IModuleHooks.sol	812a6b41481937340e2a612d23db2354ca912a55
contracts/modules/commons/interfaces/IModuleUpdate.sol	c3e4051ce5b060fee90c50f1bd66d21721794d6c
contracts/modules/commons/submodules/auth/SequenceBaseSig.sol	827c130c56583f0de8638ee3cfa7bd25a6cfffb0
contracts/modules/commons/submodules/auth/SequenceChainedSig.sol	44c763a160acca780836ec24703cb38d219fd059
contracts/modules/commons/submodules/auth/SequenceDynamicSig.sol	1e92bf01cdbf01d3bbc96b6673182e2c3786224e
contracts/modules/commons/submodules/auth/SequenceNoChainIdSig.sol	9c9a8db7ab0bd1779c67ed7afdb71ee36f9f93da
contracts/modules/commons/submodules/nonce/SubModuleNonce.sol	0e931f2ff6fac34a1657b87841667882e0513adc
contracts/modules/utils/GasEstimator.sol	cb5a719321cdf120284af321fc3d6e4d67d32115
contracts/modules/utils/MultiCallUtils.sol	50d006a83505af7257190353736f0e72ef5f6912
contracts/modules/utils/RequireUtils.sol	9705647df5a63b6c8412568b15fc9a25f027ee5f
contracts/modules/utils/SequenceUtils.sol	f1bbea3a00db308195b4c2e19b5b813f798b0bc8
contracts/utils/LibAddress.sol	a4e2f5fc2d96af45259716e883b3720dba734490
contracts/utils/LibBytes.sol	9158548a92d531ea3d7c8e28a40091d263fa9662
contracts/utils/LibBytesPointer.sol	636d632ebc29ff39b88bbb69a632c8dce9447b35
contracts/utils/LibOptim.sol	bf95b13a6f01e306819b1ec9f39c6af05d0a667e
contracts/utils/LibString.sol	10416699728dbb9c08357d806ce3023d58939222
contracts/utils/SignatureValidator.sol	d08161d36044a4f7341cc7d2f4c9dfe2840d72ec

Appendix 2 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.