

## Core Modules

3.10.2024

# Contents

1. Document Revisions .....	3
2. Overview .....	4
2.1. Ackee Blockchain Security .....	4
2.2. Audit Methodology .....	5
2.3. Finding Classification .....	6
2.4. Review Team .....	8
2.5. Disclaimer .....	8
3. Executive Summary .....	9
Revision 1.0 .....	9
Revision 1.1 .....	11
Revision 2.0 .....	11
Revision 2.1 .....	12
4. Findings Summary .....	14
Report Revision 1.0 .....	18
Revision Team .....	18
System Overview .....	18
Trust Model .....	24
Findings .....	25
Report Revision 2.0 .....	79
Revision Team .....	79
Findings .....	79
Appendix A: How to cite .....	85
Appendix B: Wake Findings .....	86
B.1. Detectors .....	86
B.2. Graphs .....	89

# 1. Document Revisions

1.0-draft	Draft Report	24.05.2024
<a href="#">1.0</a>	Final report	05.06.2024
<a href="#">1.1</a>	Fix Review	03.07.2024
<a href="#">2.0</a>	Final Report	19.09.2024
<a href="#">2.1</a>	Fix Review	03.10.2024

## 2. Overview

This document presents our findings in reviewed contracts.

### 2.1. Ackee Blockchain Security

Ackee Blockchain Security is an in-house team of security researchers performing security audits focusing on manual code reviews with extensive fuzz testing for Ethereum and Solana. Ackee is trusted by top-tier organizations in web3, securing protocols including Lido, Safe, and Axelar.

We develop open-source security and developer tooling [Wake](#) for Ethereum and [Trident](#) for Solana, supported by grants from Coinbase and the Solana Foundation. Wake and Trident help auditors in the manual review process to discover hardly recognizable edge-case vulnerabilities.

Our team teaches about blockchain security at the Czech Technical University in Prague, led by our co-founder and CEO, Josef Gattermayer, Ph.D. As the official educational partners of the Solana Foundation, we run the [School of Solana](#) and the [Solana Auditors Bootcamp](#).

Ackee's mission is to build a stronger blockchain community by sharing our knowledge.

Ackee Blockchain a.s.

Rohanske nabrezi 717/4

186 00 Prague, Czech Republic

<https://ackee.xyz>

[hello@ackee.xyz](mailto:hello@ackee.xyz)

## 2.2. Audit Methodology

### 1. Verification of technical specification

The audit scope is confirmed with the client, and auditors are onboarded to the project. Provided documentation is reviewed and compared to the audited system.

### 2. Tool-based analysis

A deep check with Solidity static analysis tool [Wake](#) in companion with [Solidity for VS Code](#) extension is performed, flagging potential vulnerabilities for further analysis early in the process.

### 3. Manual code review

Auditors manually check the code line by line, identifying vulnerabilities and code quality issues. The main focus is on recognizing potential edge cases and project-specific risks.

### 4. Local deployment and hacking

Contracts are deployed in a local [Wake](#) environment, where targeted attempts to exploit vulnerabilities are made. The contracts' resilience against various attack vectors is evaluated.

### 5. Unit and fuzz testing

Unit tests are run to verify expected system behavior. Additional unit or fuzz tests may be written using [Wake](#) framework if any coverage gaps are identified. The goal is to verify the system's stability under real-world conditions and ensure robustness against both expected and unexpected inputs.

## 2.3. Finding Classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in *configuration* (system settings or parameters, such as deployment scripts, compiler configurations, using multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

*Low* to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

### Severity

		<i>Likelihood</i>			
		High	Medium	Low	N/A
<i>Impact</i>	High	Critical	High	Medium	-
	Medium	High	Medium	Low	-
	Low	Medium	Low	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

## Impact

- ¥ High - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- ¥ Medium - Code that activates the issue will result in consequences of serious substance.
- ¥ Low - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- ¥ Warning - The issue cannot be exploited given the current code and/or *configuration*, but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- ¥ Info - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or *configuration* was to change.

## Likelihood

- ¥ High - The issue is exploitable by virtually anyone under virtually any circumstance.
- ¥ Medium - Exploiting the issue currently requires non-trivial preconditions.
- ¥ Low - Exploiting the issue requires strict preconditions.

## 2.4. Review Team

The following table lists all contributors to this report. For authors of the specific revision, see the `Revision team` section in the respective `Report revision` chapter.

Members Name	Position
It's not a joke	Lead Auditor
Michal Pševrtil	Auditor
Jan Pševrtil	Auditor
Naoki Yoshida	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

## 2.5. Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.



# 3. Executive Summary

## Revision 1.0

Rhinestone engaged Ackee Blockchain Security to perform a security review of the Core Modules with a total time donation of 21 engineering days in a period between April 29 and May 24, 2024, with ! t" p±n ! onsk# as the lead auditor.

The audit was performed on the commit [013a12<sup>\[1\]</sup>](#) and the scope was the following:

- ¥ [ModuleKit Examples](#), excluding external dependencies,
- ¥ [SentinelList](#) library (f3f84d6),
- ¥ [CheckNSignatures](#) library (53617ec).

We began our review using static analysis tools, including [Wake](#) in companion with [Solidity for VS Code](#) VS Code extension. We then took a deep dive into the logic of the contracts. For testing and fuzzing, we have involved [Wake](#) testing framework. Implemented fuzz tests are available on GitHub<sup>[2]</sup>.

During the review, we paid special attention to:

- ¥ checking the logic of examples according to specifications,
- ¥ checking the assets cannot be locked or lost,
- ¥ validating [ERC-3156](#) flashloans implementation,
- ¥ checking [ERC-4337](#) restrictions are followed,
- ¥ detecting possible reentrancies in the code,
- ¥ ensuring the arithmetic of the system is correct,
- ¥ ensuring access controls are not too weak or too strict,

¥ looking for common issues such as data validation.

Our review resulted in 32 findings, ranging from Info to High severity. The most severe high issues point to various problems in the codebase such as missing threshold checks ([H2](#)), removing a hook from a different list ([H1](#)), locked Ether ([H3](#)), [ERC-4337](#) restricted storage access ([H4](#)), updating `waitPeriod` for the nominee ([H5](#)), externally increasable borrower's nonce ([H6](#)) and many violations in [ERC-3156](#) flashloans implementation ([H7](#)). Since the codebase contains major problems, we do not recommend deploying and using the contracts until all the severe issues are resolved. The code is mostly well documented, but the code quality is not as polished as the reference examples should be.

Ackee Blockchain Security recommends Rhinestone:

- ¥ add the threshold protection when removing validators/owners,
- ¥ avoid locking assets in the contract,
- ¥ prevent interacting with restricted storage slots according to [ERC-4337](#) rules,
- ¥ fix `lastAccess` timestamp resetting for a nominee in `DeadmanSwitch` contract,
- ¥ fix bypassing whitelist and nonce increase in `ColdStorageFlashloan` contract,
- ¥ strictly follow the [ERC-3156](#) specification,
- ¥ add a check for slippage protection in `ScheduledOrders` contract,
- ¥ fix the `SentinelList.pop` function parameters order in `ColdStorageFlashLoan.removeAddress`,
- ¥ fix module types condition in `ColdStorageHook` function,
- ¥ address all other reported issues,

- ¥ perform a complete internal code review to ensure better code quality,
- ¥ complete the missing documentation.

See [Report Revision 1.0](#) for the system overview of the codebase.

## Revision 1.1

Rhinestone engaged Ackee Blockchain Security to perform a fix review of the Core Modules on the given commit [4531b2](#)<sup>[3]</sup>.

The code was moved into a new [core-modules](#) repository. The audit scope only included the fixes for findings from the previous review. No additional changes to the codebase were reviewed.

## Revision 2.0

Rhinestone engaged Ackee Blockchain Security to perform a security review of the Core Modules updates with a total time donation of 2 engineering days in a period between September 16 and September 19, 2024, with ! t" p±n ! onsk# as the lead auditor.

The audit was performed on the [diff](#) since the previous revision, commit [18b057](#)<sup>[4]</sup> and the scope was the following:

- ¥ AutoSavings/AutoSavings.sol
- ¥ ScheduledOrders/ScheduledOrders.sol
- ¥ ScheduledTransfers/ScheduledTransfers.sol
- ¥ utils/uniswap/UniswapIntegration.sol

The [diff](#) contains the following changes:

- ¥ the new contract [UniswapIntegration.sol](#),
- ¥ handling of the [sqrtX96](#), which was previously stored in the contract and

can be invalid during the execution time,

- ¥ setting the swap router in the `UniswapIntegration` contract and its child contracts (`AutoSavings` and `ScheduledOrders`).

We began our review using static analysis tools, including [Wake](#). We then performed a manual code review of the changes in the contracts. For testing and fuzzing, we have involved [Wake](#) testing framework. The fuzz test from the previous revision was updated and passed. During the review, we paid special attention to:

- ¥ `sqrtpPriceLimitX96` changes,
- ¥ swap router changes
- ¥ updated Uniswap integration.

Our review resulted in 4 findings, ranging from Warning to Medium severity. The most severe one, [M4](#), points to possible price ratio precision errors.

Ackee Blockchain Security recommends Rhinestone:

- ¥ address reported issues.

See [Report Revision 2.0](#) for the system overview and trust model.

## Revision 2.1

Rhinestone engaged Ackee Blockchain Security to perform a fix review of the Core Modules on the given commit `ccf1a4`<sup>[5]</sup>.

The audit scope only included the fixes for findings from the previous review. No additional changes to the codebase were reviewed. Issues [M4: Missing precision in price ratio decimals](#) and [W8: Missing deinitialization for swap router when uninstalling](#) were fixed, [W9: Uniswap is not deployed on some chains](#) and [W10: Contract naming](#) were acknowledged.

[1] full commit hash: 013a123305556392632c3eae9f467dc4ccdf6e

[2] fuzz tests: <https://github.com/Ackee-Blockchain/tests-rhinestone-modulekit-examples>

[3] full commit hash: 4531b2e3fffeeff520bf37fbc4bb49eec726ed61

[4] full commit hash: 18b0576411f8ae361a15b2ee69c4835a7e04b362

[5] full commit hash: ccf1a425ffeed3ef4a5ffabd962c8ab9db9c24f7

## 4. Findings Summary

The following section summarizes findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- ¥ *Description*,
- ¥ *Exploit scenario* (if severity is low or higher),
- ¥ *Recommendation* and
- ¥ *Fix* (if applicable).

Summary of findings:

Critical	High	Medium	Low	Warning	Info	Total
0	7	4	6	10	9	36

Table 2. Findings Count by Severity

Findings in detail:

Finding title	Severity	Reported	Status
<a href="#">H1: Removing from a wrong array of sigs in <code>removeSigHook</code></a>	High	<a href="#">1.0</a>	Fixed
<a href="#">H2: Missing threshold checks</a>	High	<a href="#">1.0</a>	Fixed
<a href="#">H3: <code>OwnableExecutor</code> locked <code>Ether</code></a>	High	<a href="#">1.0</a>	Fixed
<a href="#">H4: ERC-4337 restricted storage access</a>	High	<a href="#">1.0</a>	Fixed
<a href="#">H5: <code>Nominee</code> have limited access</a>	High	<a href="#">1.0</a>	Fixed

Finding title	Severity	Reported	Status
<a href="#">H6: Externally increasable borrower's nonce</a>	High	<a href="#">1.0</a>	Fixed
<a href="#">H7: ERC-3156 flashloans implementation</a>	High	<a href="#">1.0</a>	Fixed
<a href="#">M1: Missing <code>sqrtPriceLimitX96</code> check</a>	Medium	<a href="#">1.0</a>	Fixed
<a href="#">M2: Removing different address</a>	Medium	<a href="#">1.0</a>	Fixed
<a href="#">M3: Missing module type condition</a>	Medium	<a href="#">1.0</a>	Fixed
<a href="#">L1: <code>HookMultiPlexer</code> with no hooks</a>	Low	<a href="#">1.0</a>	Fixed
<a href="#">L2: <code>flashLoan</code> front-run</a>	Low	<a href="#">1.0</a>	Fixed
<a href="#">L3: Unsafe ERC-20 calls</a>	Low	<a href="#">1.0</a>	Fixed
<a href="#">L4: Missing initialized check in <code>SentinelList</code></a>	Low	<a href="#">1.0</a>	Fixed
<a href="#">L5: Missing deletion of execution element</a>	Low	<a href="#">1.0</a>	Fixed
<a href="#">L6: Excluding list element</a>	Low	<a href="#">1.0</a>	Fixed
<a href="#">W1: TODOs in module <code>HookMultiPlexer</code></a>	Warning	<a href="#">1.0</a>	Fixed
<a href="#">W2: <code>MultiFactor</code> duplicate validators</a>	Warning	<a href="#">1.0</a>	Acknowledged
<a href="#">W3: Missing <code>clearTrustedForwarder</code> call</a>	Warning	<a href="#">1.0</a>	Fixed

Finding title	Severity	Reported	Status
<a href="#">W4: <del>SchedulingBase</del> executions count validation</a>	Warning	<a href="#">1.0</a>	Fixed
<a href="#">W5: Missing zero address check</a>	Warning	<a href="#">1.0</a>	Fixed
<a href="#">W6: Missing value check in ERC-20 transfers</a>	Warning	<a href="#">1.0</a>	Fixed
<a href="#">W7: Missing array length validation</a>	Warning	<a href="#">1.0</a>	Fixed
<a href="#">I1: Redundant assignments in <del>SentinelList</del></a>	Info	<a href="#">1.0</a>	Fixed
<a href="#">I2: Proposal for refactoring <del>HookMultiPlexer</del></a>	Info	<a href="#">1.0</a>	Fixed
<a href="#">I3: <del>AutoSavings</del> percentage precision</a>	Info	<a href="#">1.0</a>	Fixed
<a href="#">I4: Unused code</a>	Info	<a href="#">1.0</a>	Fixed
<a href="#">I5: Typos and incorrect documentation</a>	Info	<a href="#">1.0</a>	Fixed
<a href="#">I6: Missing function restriction</a>	Info	<a href="#">1.0</a>	Fixed
<a href="#">I7: Unused variable</a>	Info	<a href="#">1.0</a>	Fixed
<a href="#">I8: Internal functions missing prefix</a>	Info	<a href="#">1.0</a>	Acknowledged
<a href="#">I9: Missing events</a>	Info	<a href="#">1.0</a>	Fixed
<a href="#">M4: Missing precision in price ratio decimals</a>	Medium	<a href="#">2.0</a>	Fixed



Finding title	Severity	Reported	Status
<a href="#">W8: Missing deinitialization for swap router when uninstalling</a>	Warning	<a href="#">2.0</a>	Fixed
<a href="#">W9: Uniswap is not deployed on some chains</a>	Warning	<a href="#">2.0</a>	Acknowledged
<a href="#">W10: Contract naming</a>	Warning	<a href="#">2.0</a>	Acknowledged

*Table 3. Table of Findings*

# Report Revision 1.0

## Revision Team

Members Name	Position
It's a Pansy	Lead Auditor
Michal Pevrtil	Auditor
Jan Pevrtil	Auditor
Naoki Yoshida	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

## System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

## Contracts

Contracts we find important for better understanding are described in the following section.

Modules generally provide additional functionality to smart accounts. They can be installed or uninstalled from a smart account. There are 4 types of modules defined in `ERC7579ModuleBase` abstract contract:

- ¥ Validator (`TYPE_VALIDATOR`) - validators are modules invoked during the `UserOperation` validation phase. They verify `UserOperation` signatures to determine execution eligibility. As the primary enforcers of smart account access control, validators are critical for system security.
- ¥ Executor (`TYPE_EXECUTOR`) - executors are modules invoked during the

`UserOperation` execution phase. They expand the account's native capabilities by extending its execution logic.

¥ Fallback (`TYPE_FALLBACK`) - fallbacks are modules invoked by the account's fallback function to extend its functionality.

¥ Hook (`TYPE_HOOK`) - hooks are modules triggered before or after execution to enforce specific behavior.

## AutoSavings.sol

The `AutoSavings` module allows users to automatically save a percentage of their received tokens into a designated [ERC-4626](#) vault. When a user receives tokens, the contract calculates a percentage of those tokens and automatically deposits them into a specified vault. If the received tokens are not the same as the vault's underlying asset, the contract can swap them through Uniswap V3 to ensure the correct tokens are deposited.

## ColdStorageHook.sol

The `ColdStorageHook` module allows users to lock down a sub-account, restricting the transfer of assets until a specified time has elapsed. Prevents immediate transfers of [ERC-20](#) and [ERC-721](#) tokens from a sub-account. Transfers can only be executed after a set waiting period. Contains a specific owner who can initiate time-locked transfers or modify the waiting period. The owner can request time-locked executions for transactions, including token transfers and module configuration changes. The module integrates flash loan capabilities, allowing the owner to borrow assets for the transaction execution time.

## DeadManSwitch.sol

The `DeadManSwitch` module allows users to designate a nominee who can recover their account if they become inactive for a certain period. Users specify a trusted nominee who can take control of their account in case of

inactivity. A configurable timeout period is set, after which the nominee can trigger the recovery process. The nominee must provide a valid signature to prove their authorization. If the timeout expires and the nominee provides a valid signature, they can take control of the user's smart account.

## HookMultiPlexer.sol

The `HookMultiPlexer` module enables smart accounts to integrate with various hooks, offering flexibility and customization in transaction processing. Allows smart accounts to add and manage multiple hooks simultaneously. Users can add, remove, and customize hooks based on their specific needs. The module leverages the [ERC-7484](#) registry to verify the authenticity of hooks. The multiplexer supports the following hook types:

- ¥ Global hooks - Triggered for all transactions.
- ¥ Value Hooks - Triggered only when the transaction has a value (sending ETH).
- ¥ Delegatecall Hooks: Triggered for delegatecall transactions.
- ¥ Signature Hooks: Triggered for specific function signatures.
- ¥ Target Signature Hooks: Triggered when specific functions are called on external contracts.

## HookMultiplexerLib.sol

Helper library for `HookMultiPlexer`. It contains functions for executing `preCheck` and `postCheck` functions on `subHook` and `subHooks` array. Also, it contains functions for arrays management (joining arrays, checking the array is sorted and elements are unique, pushing unique elements to array, popping, and searching), and function `decodeOnInstall` for decoding the data passed to the `HookMultiPlexer.onInstall` function.

## MultiFactor.sol

Validator module that enhances the security of smart accounts by requiring multiple validations for each transaction. Allows for integration with multiple sub-validators, each of which can enforce different authentication methods. Requires a certain number (threshold) of sub-validators to approve a transaction before it is considered valid. Users can add, remove, or update sub-validators and their associated data. Utilizes the [ERC-7484](#) registry to verify that sub-validators are attested and trustworthy.

## MultiFactorLib.sol

Helper library for the `MultiFactor` module. It contains functions for decoding an array of validators and packing/unpacking sub-validators with IDs.

## OwnableExecutor.sol

The `OwnableExecutor` module allows smart accounts to specify one or more owners who can execute transactions on their behalf while covering the gas costs. Users can assign multiple addresses as owners, enabling shared control of the smart account. Owners can execute single and batch transactions on the account they own. Uses the `SentinelList` library for the list of owners.

## OwnableValidator.sol

Validator module that enables multi-sig control over a smart account, requiring a certain number of designated owners to approve transactions. Users can specify multiple Ethereum addresses as owners of the smart account. Validator requires a minimum number (threshold) of owners to sign a transaction for it to be considered valid. Uses the `CheckNSignatures` library to recover and verify multiple signatures, and supports [ERC-1271](#) signature validation for contracts acting as owners. Uses the `SentinelList` library for the list of owners.

## RegistryHook.sol

A module that interacts with an external [ERC-7484](#) registry. It helps enforce security and trust by verifying the authenticity of modules and executors. Module checks if a module being installed on the smart account is attested to by the registry. And checks if an executor used for a transaction is attested to by the registry.

## ScheduledOrders.sol

The `ScheduledOrders` module enables users to schedule token swaps on Uniswap V3 for a future execution time. It allows users to set up token swaps with specific parameters (tokens, amount, price limit) and schedule their execution for a later time. Using `SchedulingBase` as a base contract.

## ScheduledTransfers.sol

The `ScheduledTransfers` module allows users to schedule token transfers (native tokens and [ERC-20](#) tokens) to be executed in the future. It enables users to set up transfers with a specific recipient, token, amount, and schedule their execution for a later time. Using `SchedulingBase` as a base contract.

## SocialRecovery.sol

Validator module allowing for account recovery through a social recovery mechanism. User designates a set of trusted addresses as guardians and threshold of guardian signatures required for executing `UserOperations`. When the threshold of guardian signatures is met, `UserOperation` can be executed. The recovery process is restricted to `CALLTYPE_SINGLE` operations and only on installed validator modules, preventing misuse of the recovery mechanism for unauthorized actions. Uses the `CheckNSignatures` library to recover and verify multiple signatures.

## SentinelList.sol

The `SentinelList` libraries implement a linked list data structure using the mapping. The library contains all necessary operations for managing the list, such as pushing, popping, checking for existence, iterating, and getting paginated content. The [ERC-4337](#) variant of the `SentinelList` library (`SentinelList4337`) is designed to follow the [ERC-4337](#) storage restrictions.

## CheckNSignatures.sol

Library for recovering multiple signers (both EOA and contracts) using provided `dataHash`, `signatures` and the number of `requiredSignatures`. Also, contains the `signatureSplit` function for splitting the signature into `v`, `r`, and `s` parts.

# Actors

This part describes actors of the system, their roles, and permissions.

## Owner / Smart Account

The owner / smart account can install, uninstall and configure modules. In some of the modules, the owner can delegate specific permissions to other actors (nominees, guardians).

## Attester

Attesters are entities that give attestation to modules according to attestation schema which contains basic security assumptions.

## Nominee, guardian

Nominees and guardians are external entities with delegated permissions to perform certain actions on behalf of the account owner when the threshold is reached.

## Lender

The lender role is a part of [ERC-3156](#) flashloans implementation in `FlashloanLender` and `ColdStorageHook`. The lender provides the funds for the flashloan transaction to the allowed borrower (owner).

## Borrower

The borrower role is another part of [ERC-3156](#) flashloans implementation in `FlashloanCallback` and `ColdStorageFlashloan`, who can borrow funds from the lender and return them in the same transaction. The `ColdStorageFlashloan` contract contains a whitelist of trusted lenders.

## Trust Model

Generally, the users have to trust module implementations (module developers). This point of trust is supported by the attestations mechanism, where the user can choose only modules that are attested by trusted attesters who perform module audits. This mechanism decreases the risk of using malicious modules. However, the security of using a module (or combination of modules) cannot be fully guaranteed. Especially the combination of potential bugs and trust assumptions in different installed modules can introduce various unpredictable security threats. The best practice to minimize potential attack vectors is to install as few modules as possible.

Some of the modules (namely: `DeadmanSwitch`, `OwnableExecutor`, `OwnableValidator` and `SocialRecovery`) delegate specific permissions to 3rd party accounts using multi-sig mechanism. That creates another trust assumption, where the user has to trust the 3rd party accounts in terms of misusing their privileges.

The [ERC-3156](#) flashloans implementation `FlashloanLender`, `ColdStorageHook`, `FlashloanCallback` and `ColdStorageFlashloan` assumes that both parties are



trusted entities (cold storage as a lender and the cold storage owner as a borrower).

## Findings

The following section presents the list of findings discovered in this revision.

# H1: Removing from a wrong array of sigs in `removeSigHook`

*High severity issue*

Impact:	High	Likelihood:	Medium
Target:	HookMultiPlexer.sol	Type:	Logic error

## Description

Code duplications in the function `removeSigHook` resulted in a copy-paste error. The following code listing shows the whole `removeSigHook` function.

*Listing 1. Excerpt from [HookMultiPlexer](#)*

```
344 function removeSigHook(address hook, bytes4 sig, HookType hookType) external
345 {
346     // cache the account
347     address account = msg.sender;
348     // check if the module is initialized and revert if it is not
349     if (!isInitialized(account)) revert NotInitialized(account);
350
351     // cache the storage config
352     Config storage $config = $getConfig(account);
353
354     if (hookType == HookType.SIG) {
355         // get the length of the hooks for the same sig
356         uint256 sigHooksLength = $config.sigHooks[sig].length;
357         // delete the hook
358         $config.sigHooks[sig].popAddress(hook);
359
360         // if there is only one hook for the sig, remove the sig
361         if (sigHooksLength == 1) {
362             $config.targetSigs.popBytes4(sig);
363         }
364     } else if (hookType == HookType.TARGET_SIG) {
365         // get the length of the hooks for the same sig
366         uint256 targetSigHooksLength = $config.targetSigHooks[sig].length;
367         // delete the hook
368         $config.targetSigHooks[sig].popAddress(hook);
369
370         // if there is only one hook for the sig, remove the sig
```

```

370         if (targetSigsHooksLength == 1) {
371             $config.targetSigs.popBytes4(sig);
372         }
373     } else {
374         revert UnsupportedHookType();
375     }
376 }

```

When a user wants to remove his only hook for the given `sig` of `hookType` equal to `SIG`, then the `sig` is removed from the config array `targetSigs` (instead of `sigs`).

This results in two situations. Firstly, if there is such a value present in `targetSigs` array, then it is removed. Thanks to this, the function `getHooks` does not return the right value. This error can even disable `targetSigHooks` entirely if a call type is of type `CALLTYPE_BATCH` and all `targetSigHooks` are added under the same `sig`, which was removed.

Secondly, the value of the `sig` is still present in `sigs` array, which causes the function `isInitialized` to return true, even if all hooks are removed by calling the corresponding removal functions. However, this can be worked around by uninstalling the module completely.

## Exploit scenario

Hooks of type `TARGET_SIG` can be disabled in the following scenario:

1. One `sigHook A` and one `targetSigHook B` are added under the same `sig S` value using calls:
 

```

! addSigHook(hook=A, sig=S, type=SIG)

! addSigHook(hook=B, sig=S, type=TARGET_SIG)

```
2. This stores the `sig S` to both arrays `sigs` and `targetSigs` as well as the hooks themselves to corresponding structures.

3. The `sigHook A` is removed using the `sig S` value in function call:

```
! removeSigHook(hook=A, sig=S, type=SIG).
```

4. BUT due to the code bug, this removes an entry from `targetSigs` (therefore its length will be 0) instead of `sigs`.

5. This disables the `targetSigHook B` in the function `_getFromBatch`, because there is a check `targetSigs.length != 0;`

6. The result is, that `targetSigHook B` will be disabled (will not be called at all) in all calls of `calltype = CALLTYPE_BATCH`, because in this case the hooks are invoked using the `_getFromBatch` function.

Based on the responsibility of `targetSigHook B` this could lead to major consequences.

## Recommendation

Change the first occurrence of `$config.targetSigs.popBytes4(sig);` to

```
$config.sigs.popBytes4(sig);
```

## Fix 1.1

The issue was removed during the refactoring of this module as proposed in finding [I2](#).

[Go back to Findings Summary](#)

## H2: Missing threshold checks

*High severity issue*

Impact:	High	Likelihood:	Medium
Target:	MultiFactor.sol, OwnableValidator.sol, SocialRecovery.sol	Type:	Denial of service

### Description

The project contains multiple [ERC-7579](#) validators. Each of the following validators has a function to remove a single signer from the validator configuration:

```
¥ MultiFactor.removeValidator,  
¥ OwnableValidator.removeOwner,  
¥ SocialRecovery.removeGuardian.
```

None of the functions checks the currently configured threshold and the signers count. Because of this, it is possible to remove a signer so that the threshold will be greater than the signers count.

### Exploit scenario

Owners of a smart account with the `OwnableValidator` validator with the 3/3 scheme want to rotate one owner for another. They remove one of the owners. A new owner cannot be added because the threshold is set to 3, but there are only 2 owners left.

Because `OwnableValidator` is the only validator configured for the smart account, the account becomes inaccessible.

## Recommendation

Always check the threshold and the signers count before removing a signer and revert the transaction if the threshold is equal to the current signers count.

### Fix 1.1

Fixed by adding the threshold checks to all aforementioned modules along with the logic needed to track the current signers count.

[Go back to Findings Summary](#)

### H3: **OwnableExecutor** locked Ether

*High severity issue*

Impact:	High	Likelihood:	Medium
Target:	OwnableExecutor.sol	Type:	Logic error

#### Description

The contract **OwnableExecutor** defines two functions to execute a single operation and a batch of operations on a smart account by an external approved entity. Both functions call **executeFromExecutor** on an [ERC-7579](#) smart account.

*Listing 2. Excerpt from [OwnableExecutor](#)*

```
140 function executeOnOwnedAccount(  
141     address ownedAccount,  
142     bytes calldata calldata  
143 )  
144     external  
145     payable  
146 {  
147     // check if the sender is an owner  
148     if (!accountOwners[ownedAccount].contains(msg.sender)) {  
149         revert UnauthorizedAccess();  
150     }  
151  
152     // execute the transaction on the owned account  
153  
154     ERC7579Account(ownedAccount).executeFromExecutor(ModelLib.encodeSimpleSingle  
155     (), calldata);  
156 }  
157 /**  
158  * Executes a batch of transactions on the owned account  
159  *  
160  * @param ownedAccount address of the account to execute the transaction on  
161  * @param calldata encoded data containing the transactions to execute  
162  */  
162 function executeBatchOnOwnedAccount(  

```

```

163     address ownedAccount,
164     bytes calldata calldata
165 )
166     external
167     payable
168 {
169     // check if the sender is an owner
170     if (!accountOwners[ownedAccount].contains(msg.sender)) {
171         revert UnauthorizedAccess();
172     }
173
174     // execute the batch of transaction on the owned account
175     ERC7579Account(ownedAccount).executeFromExecutor(ModelLib.encodeSimpleBatch(
176     ), calldata);
177 }

```

All the functions `executeOnOwnedAccount`, `executeBatchOnOwnedAccount`, and `executeFromExecutor` are payable. However, the Ether sent to `OwnableExecutor` is not forwarded to the smart account.

This finding was discovered using an automated static analysis detector in the Wake framework (see [Appendix C](#)).

## Exploit scenario

A user wants to transfer additional Ether to a smart account and execute an operation through the `OwnableExecutor` module. The smart account already holds some Ether. Due to the issue in `OwnableExecutor` contract, the Ether sent with `executeOnOwnedAccount` or `executeBatchOnOwnedAccount` functions remains locked in the module. The module is not upgradeable, and so the Ether is lost.

## Recommendation

Forward all the Ether sent in `executeOnOwnedAccount` and `executeBatchOnOwnedAccount` functions to the smart account.



Fix 1.1

Fixed by passing `msg.value` to external calls in both affected functions.

[Go back to Findings Summary](#)

## H4: ERC-4337 restricted storage access

*High severity issue*

Impact:	Medium	Likelihood:	High
Target:	MultiFactor.sol	Type:	Standards violation

### Description

[ERC-7562](#) defines a set of validation rules for execution of [ERC-4337](#) validation phase. The rules include restrictions on storage access. Particularly, storage access in other contracts than the smart account itself is only allowed to slots `A` and `keccak256(A || x) + offset`, where `A` represents the address of the smart account, `x` is any `bytes32` value, `offset` is a number in between 0 and 128, and `||` represents concatenation.

The `validateUserOp` function in the `MultiFactor` module is subject to these restrictions. For each smart account and each validator assigned to the smart account, there is a `SubValidatorConfig` entry.

*Listing 3. Excerpt from [MultiFactor](#)*

```
398 SubValidatorConfig storage $validator = $SubValidatorData({
399     account: account,
400     iteration: iteration,
401     subValidator: validatorAddress,
402     id: id
403 });
404
405 // check if the subValidator data is empty and return false if it is
406 bytes memory validatorStorageData = $validator.data;
```

*Listing 4. Excerpt from [DataTypes](#)*

```
15 struct SubValidatorConfig {
16     bytes data;
```

Retrieval of `$validator` follows the limitations. However, the copy of `$validator.data` to memory triggers a sequence of storage slot reads that are not allowed by the [ERC-4337](#) rules.

The storage slot dedicated `SubValidatorConfig.data` is allowed to be accessed. The slot holds the data length and may contain the data itself if the length is small enough. If the data are longer, a new storage slot is computed as `keccak256(P)`, where `P` is the number of the slot holding the length of the data. The new slot and subsequent slots are used to store the data. However, the [ERC-4337](#) rules do not allow reading from these slots.

This finding was discovered using an automated static analysis detector in the Wake framework (see [Appendix C](#)).

## Exploit scenario

A smart account user installs the `MultiFactor` module as the only validator for a smart account.

The user wants to perform other user operations on the smart account, but it is impossible because the `MultiFactor` module must be used to verify the user operations. User operation bundlers do not accept such user operations because the [ERC-4337](#) rules are not satisfied, and bypassing the rules might lead to denial of service attacks.

The smart account user is unable to perform any user operations on the smart account.

## Recommendation

Store the data in a storage slot of form `keccak256(A || x) + offset` and restrict the data length to `32 * 128 = 4096` bytes (for `offset` in between 1 and

128), with the length stored in the first slot with `offset = 0`. If the limit is too strict, consider splitting the data into smaller chunks and storing them with different `x` values used.

#### Fix 1.1

The data is now stored in `bytes32[10]` array, which prevents the restricted storage access but limits the data length to 320 bytes.

[Go back to Findings Summary](#)

## H5: **Nominee** have limited access

*High severity issue*

Impact:	Medium	Likelihood:	High
Target:	DeadManSwitch.sol	Type:	Denial of service

### Description

The validator generates validation data that execution validity relies on the **lastAccess** timestamp and user-defined **waitPeriod** time and verifies the signature from the **nominee** address.

*Listing 5. Excerpt from [DeadmanSwitch](#)*

```
161 return _packValidationData({
162     sigFailed: !sigValid,
163     validAfter: _config.lastAccess + _config.timeout,
164     validUntil: type(uint48).max
165 });
```

Before each execution, the hook updates the **lastAccess** timestamp in the **\_preCheck** function.

*Listing 6. Excerpt from [DeadmanSwitch](#)*

```
103 function _preCheck(
104     address account,
105     address,
106     uint256,
107     bytes callData
108 )
109     internal
110     override
111     returns (bytes memory hookData)
112 {
113     // if the module is not initialized, return and dont update the last
114     // access time
115     if (!isInitialized(account)) return "";
```

```
115
116     // update the last access time
117     DeadmanSwitchStorage storage _config = config[account];
118     _config.lastAccess = uint48(block.timestamp);
119 }
```

But in the current implementation, even the if execution is from the `nominee` address, the `lastAccess` timestamp is updated.

## Exploit scenario

1. After the owner of the smart account does not perform access for the `waitPeriod` time, the `nominee` address tries to access the smart account.
2. But after one successful transaction, the `nominee` address needs to wait for `waitPeriod` time again.

## Recommendation

Ensure that the first nominee's operation does not block future operations and the `lastAccess` timestamp is not updated. Or propose another solution that ensures the full recovery of the smart account in one transaction.

### Fix 1.1

The issue was fixed by adding a line resetting the timeout to the `DeadmanSwitch.validateUserOp` function:

```
uint48 validAfter = _config.lastAccess + _config.timeout;

config[userOp.sender].timeout = 0;
```

[Go back to Findings Summary](#)

## H6: Externally increasable borrower's nonce

*High severity issue*

Impact:	Medium	Likelihood:	High
Target:	ColdStorageFlashloan.sol, FlashLoanLender.sol	Type:	Denial of service

### Description

An arbitrary borrower's nonce can be increased by a malicious actor. The `FlashloanCallback.onFlashLoan` function is protected by the `onlyAllowedCallbackSender` modifier however, this modifier can be bypassed to increase the borrower's nonce.

*Listing 7. Excerpt from [FlashloanCallback](#)*

```
106     address borrower,
107     address, /*token*/
108     uint256, /*amount*/
109     uint256, /*fee*/
110     bytes calldata data
111 )
112     external
113     onlyAllowedCallbackSender
114     returns (bytes32)
115 {
116     // decode the data
117     (FlashLoanType flashLoanType, bytes memory signature, Execution[] memory
118     executions) =
119     abi.decode(data, (FlashLoanType, bytes, Execution[]));
120     // get the hash
121     bytes32 hash = getTokengatedTxHash(flashLoanType, executions,
122     nonce[borrower]);
123     // increment the nonce
124     nonce[borrower]++;
125     // format the hash
```

The modifier `onlyAllowedCallbackSender` calls the virtual function

`FlashloanCallback._isAllowedCallbackSender` and checks the result. This function is overridden in the `ColdStorageFlashloan` contract. It checks the whitelist of `msg.sender` however, an arbitrary whitelist can be created using unprotected external functions `onInstall` and `addAddress`.

*Listing 8. Excerpt from [ColdStorageFlashloan](#)*

```
109 function _isAllowedCallbackSender() internal view virtual override returns
    (bool) {
110     address caller = _msgSender();
111     return whitelist[msg.sender].contains(caller);
112 }
```

The `_msgSender` function is just checking the last 20 bytes of calldata which can be arbitrary.

*Listing 9. Excerpt from [ERC7579FallbackBase](#)*

```
18 function _msgSender() internal pure returns (address sender) {
19     // The assembly code is more direct than the Solidity version using
    `abi.decode`.
20     /* solhint-disable no-inline-assembly */
21     /// @solidity memory-safe-assembly
22     assembly {
23         sender := shr(96, calldataload(sub(calldatasize(), 20)))
24     }
25     /* solhint-enable no-inline-assembly */
26 }
```

Therefore the `onFlashLoan` function is executable from malicious contracts and can increment nonce of arbitrary `borrower` account passed to `onFlashLoan` function.

## Exploit scenario

1. The attacker creates a contract that implements `IERC1271` and bypasses the `isValidSignature` by returning `bytes4(0x1626ba7e)`.



2. The contract calls `ColdStorageFlashloan.addAddress` to add the target borrower's address to the whitelist.
3. The contract creates malicious calldata for `onFlashLoan` call and call the function.
4. The borrower's nonce is incremented, which invalidates the borrower's flashloan transaction.
5. Also, the `_execute` function performs an external call to `msg.sender` (malicious contract) which can be potentially misused for various actions.

## Recommendation

Attach the nonce to the lender-borrower pair to avoid nonce incrementation by malicious actors.

```
mapping(address lender => mapping(address borrower => uint256 nonces)) public  
nonce;
```

## Fix 1.1

The finding was fixed by using two-dimensional mapping for nonce.

```
mapping(address account => mapping(address borrower => uint256 nonces)) public  
nonce;
```

The nonce now depends on the account and borrower pair.

[Go back to Findings Summary](#)

## H7: ERC-3156 flashloans implementation

*High severity issue*

Impact:	High	Likelihood:	Medium
Target:	FlashloanLender.sol, FlashloanCallback.sol	Type:	Code quality

### Description

[ERC-3156](#) flashloans implementation does not follow the reference implementation and best practices to avoid security threats. According to [ERC-3156](#) we identified the following violations:

#### Lender

- ¥ The function `FlashloanLender.flashLoan` performs only the transfer from the lender to the borrower, but missing the transfer from the borrower back to the lender (amount + fee) and relies on the borrower to perform this operation. [ERC-3156](#) specs define this approach as a must.
- ¥ The function `FlashloanLender.flashLoan` is missing the reentrancy protection.
- ¥ The `flashFee` function must revert for unsupported tokens. (Never reverts in `ColdStorageHook`).
- ¥ The `maxFlashLoan` must return the maximum possible loan for the `token` or 0 for the unsupported token. (Returns always 0 in `ColdStorageHook`).

#### Receiver

- ¥ The `FlashloanCallback` does not implement `IERC3156FlashBorrower` interface, although `onFlashLoan` function is present.

## Exploit scenario

Violating the [ERC-3156](#) specification and best practices opens many back doors for balance manipulations, draining funds using reentrancy or weak access controls (in combination with the [H6: Externally increasable borrower's nonce](#) which allows bypassing the borrower's whitelist). During the limited time, we did not find any specific exploit scenario however, the violations above are critical and can lead to severe security threats.

## Recommendation

Strictly follow all "MUST" assumptions in the [ERC-3156](#) specification to avoid security threats. Also, add reentrancy protection to the `flashLoan` function (even the both parties are trusted), since the [ERC-3156](#) by design cannot follow the Check Effects Interaction pattern.

### Fix 1.1

Fixed. The function `_transferTokenBack` is implemented and used in `FlashloanLender.flashLoan` function, which is now protected using the `nonReentrant` modifier.

The function `ColdStorageHook.maxFlashLoan` now returns `token.balanceOf(msg.sender)`. The function `ColdStorageHook.flashFee` always reverts with the `UnsupportedToken` error. The function `ColdStorageHook.flashFeeToken` returns `address(0)`.

[Go back to Findings Summary](#)

## M1: Missing `sqrtPriceLimitX96` check

*Medium severity issue*

Impact:	High	Likelihood:	Low
Target:	ScheduledOrders.sol	Type:	Data validation

### Description

The module `ScheduledOrders` serves as an [ERC-7579](#) executor for swapping tokens through Uniswap V3.

However, the contract does not validate the `sqrtPriceLimitX96` parameter value. Setting the parameter to zero skips slippage protection in Uniswap.

### Exploit scenario

A user installs the `ScheduledOrders` module and sets `sqrtPriceLimitX96` parameter to zero. The function `executeOrder` executing the swap is then called automatically by an off-chain tool. Due to the missing slippage protection, the automated call may perform a highly unfavorable swap for the user.

### Recommendation

Check if `sqrtPriceLimitX96` parameter equals zero and revert in such case.

### Fix 1.1

The issue was fixed by adding the revert condition.

```
if (sqrtPriceLimitX96 == 0) revert InvalidSqrtPriceLimitX96();
```

[Go back to Findings Summary](#)

## M2: Removing different address

*Medium severity issue*

Impact:	Low	Likelihood:	High
Target:	ColdStorageFlashloan.sol	Type:	Logic error

### Description

The order of arguments is swapped against the `SentinelList.pop` function.

Snippet from the `ColdStorageFlashLoan.removeAddress` function:

*Listing 10. Excerpt from [ColdStorageFlashLoan](#)*

```
85 function removeAddress(address addressToRemove, address prevAddress) external
86 {
87     // remove the address from the whitelist
88     whitelist[msg.sender].pop(addressToRemove, prevAddress);
89 }
```

Snippet from the `SentinelList.pop` function.

```
function pop(SentinelList storage self, address prevEntry, address popEntry)
```

### Exploit scenario

The arguments are swapped, therefore transactions fail in general. If the transaction succeeds, the unexpected address is removed and the unexpected address remains in the list.

### Recommendation

Change the order of the `removeAddress` function arguments.

```
function removeAddress(address addressToRemove, address prevAddress) external {
```

```
Ê // remove the address from the whi tel i st
Ê whi tel i st[msg. sender].pop(prevAddress, addressToRemove);
}
```

## Fix 1.1

The order of arguments in the `removeAddress` function was fixed.

```
whi tel i st[msg. sender].pop({ prevEntry: prevAddress, popEntry: addressToRemove
});
```

[Go back to Findings Summary](#)

## M3: Missing module type condition

*Medium severity issue*

Impact:	Low	Likelihood:	High
Target:	ColdStorageHook.sol	Type:	Configuration

### Description

The `ColdStorageHook` module is used as an executor but the `isModuleType` function does not return `true` for `TYPE_EXECUTOR`.

*Listing 11. Excerpt from [ColdStorageHook](#)*

```
589 function isModuleType(uint256 typeId) external pure virtual returns (bool) {
590     if (typeId == TYPE_HOOK || typeId == TYPE_FALLBACK) {
591         return true;
592     }
593 }
```

### Exploit scenario

The user wants to use the `ColdStorageHook` module as an executor but it is not possible in the current setup.

### Recommendation

Add the `TYPE_EXECUTOR` constant into the condition in the `isModuleType` function.

```
function isModuleType(uint256 typeId) external pure virtual returns (bool) {
    if (typeId == TYPE_EXECUTOR || typeId == TYPE_HOOK || typeId ==
    TYPE_FALLBACK) {
        return true;
    }
}
```

## Fix 1.1

The constant `TYPE_EXECUTOR` was added to the condition in the `isModuleType` function.

[Go back to Findings Summary](#)



## L1: **HookMultiPlexer** with no hooks

*Low severity issue*

Impact:	Low	Likelihood:	Medium
Target:	HookMultiPlexer.sol	Type:	Logic error

### Description

The function `isInitialized` in **HookMultiPlexer** module checks if the module is initialized based on array lengths.

*Listing 12. Excerpt from [HookMultiPlexer](#)*

```
195 function isInitialized(address smartAccount) public view returns (bool) {
196     // cache the storage config
197     Config storage $config = $getConfig(smartAccount);
198     // if any hooks are set, the module is initialized
199     return $config.globalHooks.length != 0 ||
    È $config.delegatecallHooks.length != 0
200     || $config.valueHooks.length != 0 || $config.sigs.length != 0
201     || $config.targetSigs.length != 0;
202 }
```

Installation of the module with no hooks or removal of the last hook leaves the module uninitialized, and `addHook` function would revert.

*Listing 13. Excerpt from [HookMultiPlexer](#)*

```
252 function addHook(address hook, HookType hookType) external {
253     // cache the account
254     address account = msg.sender;
255     // check if the module is initialized and revert if it is not
256     if (!isInitialized(account)) revert NotInitialized(account);
```

### Exploit scenario

1. The **HookMultiPlexer** module is installed with no hooks on a smart account

or the last hook is removed from the module by calling the `removeHook` function.

2. The `addHook` function is called to install a new hook.
3. The function reverts because the module is not considered initialized.
4. The user is forced to reinstall the module with at least one hook.

## Recommendation

Use an extra boolean variable to track the initialization state of the module.

```
bool private initialized;  
  
...  
  
function isInitialized(address smartAccount) public view returns (bool) {  
    return initialized;  
}
```

## Fix 1.1

Fixed by adding an extra boolean variable `initialized` to the storage tracking the initialization state of the module.

[Go back to Findings Summary](#)

## L2: **flashLoan** front-run

*Low severity issue*

Impact:	Low	Likelihood:	Low
Target:	FlashloanCallback.sol, FlashloanLender.sol	Type:	Front-running

### Description

Although unlikely, the function **flashLoan** may be front-run with different **token** and **value** parameters.

*Listing 14. Excerpt from [FlashloanLender](#)*

```
95 function flashLoan(  
96     IERC3156FlashBorrower receiver,  
97     address token,  
98     uint256 value,  
99     bytes calldata data  
100 )
```

*Listing 15. Excerpt from [FlashloanCallback](#)*

```
105 function onFlashLoan(  
106     address borrower,  
107     address, /*token*/  
108     uint256, /*amount*/  
109     uint256, /*fee*/  
110     bytes calldata data  
111 )
```

The signature being validated is stored in **data**. The signed data do not include the **token** and **value** parameters. Given this, anyone can front-run the **flashLoan** function execution with different **token** and **value** parameters under the condition that the contract that performs the execution already has enough tokens used in the execution.

## Exploit scenario

An attacker is observing the transactions pool and tries to front-run `flashLoan` executions with different `token` and `value` parameters. If the front-run succeeds, the execution is performed in an unexpected way and the legitimate transaction does not succeed because of a nonce used in the signature.

## Recommendation

Make the signature depend on the `token` and `value` parameters.

## Fix 1.1

Fixed by making the signature depend on the `token` and `value` parameters.

[Go back to Findings Summary](#)

## L3: Unsafe ERC-20 calls

*Low severity issue*

Impact:	Medium	Likelihood:	Low
Target:	AutoSavings.sol, FlashloanLender.sol, ScheduledTransfers.sol, Uniswap.sol	Type:	Standards violation

### Description

The project contains multiple modules interacting with [ERC-20](#) tokens, but none of them uses `SafeERC20` or its alternative. As a consequence, the executed transactions may not revert (even though they should) or may revert (even though they should not).

### Exploit scenario

Specifically, the following situations may occur:

1. The `AutoSavings` contract calls `approve` function through `UniswapV3Integration` contract or directly from the contract itself. The `approve` function may revert if the allowance is not reset to zero first.
2. The `FlashloanLender` contract calls `transfer` function, which may return `false` and not revert.
3. The `ScheduledTransfers` contract calls `transfer` function, which may return `false` and not revert, counting the execution as successful.

### Recommendation

Use `SafeERC20` library or its alternative when interacting with [ERC-20](#) tokens.

## Fix 1.1

All of the described scenarios were fixed by resetting the allowance to zero before calling the `approve` function and checking the optional return value of the `transfer` and `transferFrom` functions.

[Go back to Findings Summary](#)

## L4: Missing initialized check in SentinelList

*Low severity issue*

Impact:	Medium	Likelihood:	Low
Target:	SentinelList.sol, SentinelListBytes32.sol, SentinelList4337.sol	Type:	Logic error

### Description

When using the `push` function, it does not check whether the list has been initialized. Therefore, it is possible to use `push` function, but this data will be lost because the list was not properly initialized beforehand.

```
function push(SentinelList storage self, address newEntry) internal {
    if (newEntry == ZERO_ADDRESS || newEntry == SENTINEL) {
        revert LinkedList_InvalidEntry(newEntry);
    }
    if (self.entries[newEntry] != ZERO_ADDRESS) revert
    LinkedList_EntryAlreadyInList(newEntry);
    self.entries[newEntry] = self.entries[SENTINEL];
    self.entries[SENTINEL] = newEntry;
}
```

### Exploit scenario

1. The user using a module that has multiple module types, and the user wants to reinstall the module.
2. The user calls `onUninstall` function.
3. The user calls `addAddress` function which just does push to the list, before `onInstall` call.
4. The transaction succeeds and the list state is initialized but this element does not exist in the list.

## Recommendation

Check `entries[SENTINEL]` value is `ZERO_ADDRESS` or not.

### Fix 1.1

The finding was fixed by adding the `safePush` function. The `safePush` function checks whether the list has been initialized and if not, it initializes the list and inserts the element. It is recommended to use the `push` function only if it is confirmed the list is initialized.

[Go back to Findings Summary](#)



## L5: Missing deletion of execution element

*Low severity issue*

Impact:	Low	Likelihood:	Medium
Target:	ColdStorageHook.sol	Type:	Logic error

### Description

Previously requested executions with timestamps exceeding the `executeAfter` function remain callable even after one execution and even after reinstalling the module.

*Listing 16. Excerpt from [ColdStorageHook](#)*

```
448 // get the execution hash
449 bytes32 executionHash = _execDigest(target, value, callData);
450
451 // check the timelocked execution
452 _checkTimelockedExecution(account, executionHash);
453
454 // emit the TimelockExecuted event
455 emit TimelockExecuted(account, executionHash);
456
457 return "";
```

It emits the `TimelockExecuted` event but the execution is not removed from the `executions` mapping.

### Exploit scenario

The user can call the same `target` with the same `amount` and same `callData` repeatedly after the timestamp exceeds `executeAfter`.

If the user reinstalls the `ColdStorageHook` module, he can execute the execution without requesting, because the subAccount's entry in `executions` mapping is not cleared in `onUninstall` function.

## Recommendation

Remove the execution from `executions` mapping in the `onExecuteFromExecutor` function.

Delete all executions for `subAccount` from `executions` mapping in the `onUninstall` function.

## Fix 1.1

The finding was fixed by removing the execution hash for each execution and all execution hashes are removed when uninstalling the module.

[Go back to Findings Summary](#)

## L6: Excluding list element

*Low severity issue*

Impact:	Low	Likelihood:	Medium
Target:	SentinelListBytes32.sol	Type:	Logic error

### Description

In the `getEntriesPaginated` function, the starting element should be contained in the list and should not revert when `start` is contained in the list.

```
if (start != SENTINEL && contains(self, start)) revert  
  LinkedLi st_InvalidEntry(start);
```

### Exploit scenario

If the start is not `SENTINEL` but rather an element contained in the list, it does not return an array of elements.

### Recommendation

Fix the code to verify the existence of the `start` element.

```
if (start != SENTINEL && !contains(self, start)) revert  
  LinkedLi st_InvalidEntry(start);
```

### Fix 1.1

The finding was fixed by inverting the condition.

[Go back to Findings Summary](#)

## W1: TODOs in module **HookMultiPlexer**

Impact:	Warning	Likelihood:	N/A
Target:	HookMultiPlexer.sol	Type:	Code quality

### Description

The **HookMultiPlexer** module contains two TODOs. These indicate areas requiring further attention and development and can be a hint for hackers in rare cases.

The following code snippets reveal their location.

*Listing 17. Excerpt from [HookMultiPlexer](#)*

```
407 // TODO: write tests for this. I think this breaks if globalHooks is empty
408 // get the global and account sig hooks
409 address[] memory hooks = $config.globalHooks;
```

*Listing 18. Excerpt from [HookMultiPlexer](#)*

```
479 // todo: optimise
480 assembly ("memory-safe") {
481     let dataPointer := add(hookData.offset, calldataload(hookData.offset))
482     hooksAndContexts.offset := add(dataPointer, 0x20)
483     hooksAndContexts.length := calldataload(dataPointer)
484 }
```

### Recommendation

It is recommended to address these TODOs to ensure code completeness and maintainability.

#### Fix 1.1

The implementation contains no more TODOs. This module was refactored as proposed in finding [12](#).

[Go back to Findings Summary](#)

## W2: **MultiFactor** duplicate validators

Impact:	Warning	Likelihood:	N/A
Target:	MultiFactor.sol	Type:	Data validation

### Description

The function `onInstall` in the **MultiFactor** module accepts an array of initial validators. The function does not check if there are any duplicate pairs

```
(address validatorAddress, ValidatorId id).
```

### Recommendation

Consider checking if a given pair of `validatorAddress` and `id` parameters already was initialized and revert in this case.

### Acknowledgment 1.1

*This is a feature - validators should be able to be re-used.*

Ñ Rhinestone Team

[Go back to Findings Summary](#)

## W3: Missing `clearTrustedForwarder` call

Impact:	Warning	Likelihood:	N/A
Target:	RegistryHook.sol	Type:	Logic error

### Description

Unlike other hooks, the `RegistryHook` module is missing the `clearTrustedForwarder` call in the `onUninstall` function.

### Recommendation

Call the `clearTrustedForwarder` function in the `RegistryHook.onUninstall` function.

### Fix 1.1

Fixed. The `clearTrustedForwarder` call was added to the `RegistryHook.onUninstall` function.

[Go back to Findings Summary](#)

## W4: **SchedulingBase** executions count validation

Impact:	Warning	Likelihood:	N/A
Target:	SchedulingBase.sol	Type:	Data validation

### Description

The **SchedulingBase** contract is used by two executors to schedule executions on a smart account. However, the contract does not check that **numberOfExecutions** parameter is greater than zero.

### Recommendation

Check the **numberOfExecutions** parameter and revert if it equals zero.

### Fix 1.1

Fixed by adding an extra check that the **numberOfExecutions** parameter is not equal to zero when creating a new execution.

[Go back to Findings Summary](#)



## W5: Missing zero address check

Impact:	Warning	Likelihood:	N/A
Target:	OwnableExecutor.sol	Type:	Data validation

### Description

In the `OwnableExecutor` contract the `onInstall` function missing the `owner` zero-address validation. Other `accountOwners` related functions check zero-address and revert with `InvalidOwner`

### Recommendation

Add the zero-address check for the owner address into the `onInstall` function.

### Fix 1.1

The finding was fixed by adding the owner zero-address check.

[Go back to Findings Summary](#)

## W6: Missing value check in ERC-20 transfers

Impact:	Warning	Likelihood:	N/A
Target:	ColdStorageHook.sol	Type:	Logic error

### Description

Users can request execution that sending [ERC-20](#) or [EIP-712](#) with the native token value. Usually, the `transfer` or `transferFrom` functions are not `payable` and the transaction would revert. However, in case transfer functions are `payable`, the native token would be transferred to the token contract.

### Recommendation

Check the native token value is zero when it was requested for `transfer` or `transferFrom` execution.

### Fix 1.1

Fixed by checking the `value` is zero when calldata length is not zero at `requestTimelockedExecution` function.

[Go back to Findings Summary](#)

## W7: Missing array length validation

Impact:	Warning	Likelihood:	N/A
Target:	AutoSavings.sol	Type:	Data validation

### Description

The `AutoSavings.onInstall` function is missing array length mismatch validation.

### Recommendation

Add an array length mismatch validation.

```
if (_tokens.length != _configs.length) revert TokenConfigLengthMismatch();
```

### Fix 1.1

Fixed. The issue was fixed by creating the `ConfigWithToken` struct with the token address and changing the init data to `ConfigWithToken[]`.

[Go back to Findings Summary](#)

## I1: Redundant assignments in `SentinelList`

Impact:	Info	Likelihood:	N/A
Target:	SentinelList.sol, SentinelListBytes32.sol, SentinelList4337.sol	Type:	Code quality

### Description

Libraries `SentinelList` and its variants contain redundant assignments.

In the function `popAll`, in the following code snippet, the last line is excessive.

```
function popAll (SentinelList storage self) internal {  
    address next = self.entries[SENTINEL];  
    while (next != ZERO_ADDRESS) {  
        address current = next;  
        next = self.entries[next];  
        self.entries[current] = ZERO_ADDRESS;  
    }  
    self.entries[SENTINEL] = ZERO_ADDRESS;  
}
```

The assignment `self.entries[SENTINEL] = ZERO_ADDRESS` is redundant, because `self.entries[SENTINEL]` is either:

1. already equal to `ZERO_ADDRESS` if the sentinel list is uninitialized, then the while loop is skipped.
2. not equal to `ZERO_ADDRESS`, so it goes into the while loop, where:
  - ! it's immediately set to `ZERO_ADDRESS` if the sentinel list is empty because `SENTINEL` points to `SENTINEL`.
  - ! or it's set to `ZERO_ADDRESS` in the last transit through the while loop because the last entry of the sentinel list points to `SENTINEL`.

## Recommendation

Remove the unnecessary assignments to make the libraries cleaner.

### Fix 1.1

The redundant assignments were removed. Also, documentation comments were added for all methods in the library `SentinelList` and its variants.

[Go back to Findings Summary](#)

## I2: Proposal for refactoring `HookMultiPlexer`

Impact:	Info	Likelihood:	N/A
Target:	HookMultiPlexer.sol	Type:	Code quality

### Description

In the module `HookMultiPlexer` are many if and else statements regarding the hook types, only to work with the right variable. For example in functions `addHook`, `addSigHook`, `removeHook` or `removeSigHook`.

Nested mapping similar to:

```
// sig => hook type => hooks
mapping(bytes4 => mapping(HookType => address[])) hooks;
```

would solve this handling and would make the code much more concise, readable and maintainable.

There are also a lot of code duplications along the module. The longest one has 26 lines (comments including) in the `onInstall` function - lines 98 to 123 and lines 125 to 150. Other code duplications are in functions `onUninstall`, `getHooks`, `addSigHook` or in function `removeSigHook` which even resulted in issue [H1: Removing from a wrong array of sigs in `removeSigHook`](#).

### Recommendation

Consider refactoring the `HookMultiPlexer` module.

#### Fix 1.1

The whole class was refactorized using mapping inspired by the proposal. That led to a decrease in the line count of implementation by about half.

[Go back to Findings Summary](#)

### I3: **AutoSavings** percentage precision

Impact:	Info	Likelihood:	N/A
Target:	AutoSavings.sol	Type:	Arithmetics

#### Description

The **AutoSavings** module allows for saving a given percentage of received tokens. The following function is used to calculate the amount of tokens to save.

*Listing 19. Excerpt from [AutoSavings](#)*

```
194 function calcDepositAmount(  
195     uint256 amountReceived,  
196     uint256 percentage  
197 )  
198     public  
199     pure  
200     returns (uint256)  
201 {  
202     // calculate the amount to be saved which is the  
203     // percentage of the amount received  
204     return (amountReceived * percentage) / 100;  
205 }
```

The current resolution is 1%, i.e. the minimal percentage to save is 1%.

#### Recommendation

Consider increasing the precision with at least two decimal places.

#### Fix 1.1

Fixed. The percentage resolution was increased to 2 decimal places and the PRBMath library is now used for fixed-point math.

[Go back to Findings Summary](#)

## I4: Unused code

Impact:	Info	Likelihood:	N/A
Target:	**/*	Type:	Code quality

### Description

The project contains multiple occurrences of unused code. See [Appendix C](#) for the full list.

Unused functions were not reported due to the nature of the project being a base kit for other smart account modules.

### Recommendation

Remove the unused code to improve the readability and maintainability of the codebase.

### Fix 1.1

All of the unused code occurrences were fixed.

[Go back to Findings Summary](#)



## 15: Typos and incorrect documentation

Impact:	Info	Likelihood:	N/A
Target:	**/*	Type:	Code quality

### Description

There are several typos and documentation issues across the project.

- ¥ Multiple projects define an error named `UnsoportedOperation`.
- ¥ The file named `HookMultiPlexer.sol` contains the contract named `HookMultiplexer`.
- ¥ `SocialRecovery.isValidSignatureWithSender` uses copy-pasted documentation string from `DeadmanSwitch`.
- ¥ `ColdStorageHook.requestTimelockedModuleConfig` function documentation is copy-pasted from function `requestTimelockedExecution` and does not describe the actual `requestTimelockedModuleConfig` behavior.
- ¥ `ColdStorageHook.onExecuteFromExecutor` documentation mentions that the function reverts but it's not.
- ¥ The `SentinelList.sol` file contains the `SentinelListLib` library.
- ¥ The `SentinelList4337.sol` file contains the `SentinelList4337Lib` library.
- ¥ The `SentinelListBytes32Lib.sol` file contains `LinkedBytes32Lib` library.
- ¥ The `SentinelList` is missing NatSpec documentation.
- ¥ The file `CheckNSignatures.sol` contains the `CheckSignatures` contract.
- ¥ The `CheckNSignatures` is missing NatSpec documentation.

### Recommendation

Fix the typos and documentation to improve code quality.

### Fix 1.1

Most of the recommendations were applied, the rest was acknowledged.

[Go back to Findings Summary](#)

## I6: Missing function restriction

Impact:	Info	Likelihood:	N/A
Target:	SocialRecovery.sol	Type:	Code quality

### Description

The `validateUserOp` function is not a `virtual` function and the function can be restricted to `view`.

*Listing 20. Excerpt from [Social Recovery](#)*

```
226 function validateUserOp(  
227     PackedUserOperation calldata userOp,  
228     bytes32 userOpHash  
229 )  
230     external  
231     override  
232     returns (ValidationData)
```

### Recommendation

Consider restricting the function to `view`.

### Fix 1.1

The finding was fixed by restricting the function to `view`.

[Go back to Findings Summary](#)

## I7: Unused variable

Impact:	Info	Likelihood:	N/A
Target:	ColdStorageHook.sol	Type:	Code quality

### Description

The `success` variable in the `ColdStorageHook.checkHash` function is not used.

*Listing 21. Excerpt from [ColdStorageHook](#)*

```
146 function checkHash(  
147     address account,  
148     bytes32 hash  
149 )  
150     external  
151     view  
152     returns (bytes32 executeAfter)  
153 {  
154     // get the executeAfter timestamp  
155     bool success;  
156     (success, executeAfter) = executions[account].tryGet(hash);  
157 }
```

### Recommendation

Remove the `success` variable.

```
(, executeAfter) = executions[account].tryGet(hash);
```

### Fix 1.1

Fixed. The `success` variable was removed.

[Go back to Findings Summary](#)

## I8: Internal functions missing prefix

Impact:	Info	Likelihood:	N/A
Target:	ERC7579HookDestruct.sol	Type:	Standards violation

### Description

Internal functions in the `ERC7579HookDestruct` contract are not prefixed with an underscore. Namely `onExecute`, `onExecuteBatch`, `onExecuteFromExecutor`, `onExecuteBatchFromExecutor`, `onInstallModule`, `onUninstallModule`, `onUnknownFunction`, and `onPostCheck`.

### Recommendation

Add an underscore prefix to internal function names according to Solidity best practices.

### Acknowledgment 1.1

*Would break existing modules from external developers.*

Ń Rhinestone Team

[Go back to Findings Summary](#)

## I9: Missing events

Impact:	Info	Likelihood:	N/A
Target:	**/*	Type:	Logging

### Description

Most of the modules are missing events emits in state changing functions.

¥ All - `onInstall`, `onUninstall`

¥ AutoSavings - `setConfig`, `deleteConfig`

¥ ColdStorageFlashloan - `addAddress`, `removeAddress`

¥ ColdStorageHook - `setWaitPeriod`

¥ HookMultiplexer - `addHook`, `addSigHook`, `removeHook`, `removeSigHook`

¥ MultiFactor - `setThreshold`

¥ OwnableExecutor, OwnableValidator - `addOwner`, `removeOwner`

¥ SocialRecovery - `setThreshold`, `addGuardian`, `removeGuardian`

### Recommendation

It is a good practice to emit events after every important state change.

#### Fix 1.1

Fixed. Event emits for important state changes are added.

[Go back to Findings Summary](#)

# Report Revision 2.0

## Revision Team

Members Name	Position
It's a Pansky	Lead Auditor
Jan Pevrtil	Auditor
Naoki Yoshida	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

## Findings

The following section presents the list of findings discovered in this revision.

## M4: Missing precision in price ratio decimals

*Medium severity issue*

Impact:	Low	Likelihood:	High
Target:	UniswapIntegration.sol	Type:	Arithmetics

### Description

The price ratio should have decimals to prevent the ratio from losing precision. So, the price ratio should have a decimal for the calculation. Otherwise, the value of the ratio could be rounded down and it will be 0.

*Listing 22. Excerpt from [UniswapIntegration](#)*

```
144 uint256 decodedSqrtPrice = sqrtPriceX96 / (2 ** 96);
145 priceRatio = decodedSqrtPrice * decodedSqrtPrice;
146 return priceRatio;
```

### Exploit scenario

The `priceRatioToPrice` function will return the price based on the ratio. When the ratio is less than 1, the price will be 0. Which causes the error in the future calculation.

Also if the ratio is 1.5 then, the return value will be 1 because of the rounding down.

### Recommendation

The usage of the calculated value would depend on the module, so it would be better to have a calculation for each module.

In case it is necessary to the ratio of tokens, do not use the `priceRatio` value for the calculation of trading. Instead, use the `sqrtPriceX96` for trading calculations to prevent loss of precision.



Or, use precision for the ratio value, for example, using  $10^{18}$  precision for the price ratio to return the ratio. To calculate accurately without overflow, it is recommended to use the uniswap library `FullMath`, which does not support the 0.8.0 version, so migration is required.

The unused view functions `sqrPriceX96toPriceRatio`, `priceRatioToSqrPriceX96`, and `priceRatioToPrice` are moved to the `test` directory.

[Go back to Findings Summary](#)

## W8: Missing deinitialization for swap router when uninstalling

Impact:	Warning	Likelihood:	N/A
Target:	ScheduledOrders.sol	Type:	Logic error

### Description

Missing the `_deinitSwapRouter()` function call in the `onUninstall` function which reset the swap router setting.

### Recommendation

Add the `_deinitSwapRouter()` function call in the `onUninstall` function.

The `_deinitSwapRouter()` function call is added in the `onUninstall` function.

[Go back to Findings Summary](#)

## W9: Uniswap is not deployed on some chains

Impact:	Warning	Likelihood:	N/A
Target:	UniswapIntegration.sol	Type:	Configuration

### Description

Uniswap for the swapping tokens is not deployed on some chains. The module requires the user to input the swap router address.

### Recommendation

Be aware that Uniswap for the swapping tokens is not deployed for those chains. Make sure not to deploy the module for those chains or implement functionality that completes the operation without using swapping tokens.

The issue was acknowledged by the client.

[Go back to Findings Summary](#)

## W10: Contract naming

Impact:	Warning	Likelihood:	N/A
Target:	UniswapIntegration.sol	Type:	Code quality

### Description

The file `UniswapIntegration.sol` contains contract `InitializableUniswapV3Integration`. Inconsistent contract naming decreases the code clarity and can lead to overlooked errors.

*Listing 23. Excerpt from [UniswapIntegration](#)*

```
15 abstract contract InitializableUniswapV3Integration {
```

### Recommendation

Unify the contract and file naming.

The issue was acknowledged by the client.

[Go back to Findings Summary](#)

# Appendix A: How to cite

Please cite this document as:

[Ackee Blockchain Security](#), Rhinestone: Core Modules, 3.10.2024.

# Appendix B: Wake Findings

This section lists the outputs from the [Wake](#) framework used for testing and static analysis during the audit.

## B.1. Detectors

```
wake detect unused-using-for

! "[WARNI NG][LOW] Unused contract in using-for directive [unused-using-for]" #
$ 19 * @author Rhi nestone $
$ 20 */ $
$ 21 contract AutoSavings is ERC7579ExecutorBase { $
$ %22 using ERC4626Integration for *; $
$ 23 using SentinelLib for SentinelLib.SentinelLib; $
$ 24 $ $
$ 25 $ $
&" examples/src/AutoSavings/AutoSavings.sol ".....

! "[WARNI NG][LOW] Unused contract in using-for directive [unused-using-for]" #
$ 14 * @author Rhi nestone $
$ 15 */ $
$ 16 abstract contract FIASHLanCallBack is ERC7579FallbackBase, ERC7579Exec $
$ %17 using SentinelLib for SentinelLib.SentinelLib; $
$ 18 using SignatureCheckerLib for address; $
$ 19 $ $
$ 20 $ $
&" examples/src/FIASHLan/FIASHLanCallBack.sol ".....

! "[WARNI NG][LOW] Unused contract in using-for directive [unused-using-for]" #
$ 17 */ $
$ 18 contract OwnableValidator is ERC7579ValidatorBase { $
$ 19 using LibSort for *; $
$ %20 using SignatureCheckerLib for address; $
$ 21 using SentinelLib4337Lib for SentinelLib4337Lib.SentinelLib; $
$ 22 $ $
$ 23 $ $
&" examples/src/OwnableValidator/OwnableValidator.sol ".....
```

Figure 1. Unused using-for directives

```

wakefdetectfunused-event

! "Ê[INFO][HIGH]ÊUnusedÊEventÊ[unused-event]Ê"#####"#$
$ 28Ê //ÊaccountÊ=>Êconfig $
$ 29Ê mapping(address account => DeadmanSwi tchStorage) publ i c confi g; $
$ 30Ê $
$ %Ê31Ê event Recovery(address account, address nomi nee); $
$ 32Ê $
$ 33Ê error UnsoportedOperati on(); $
$ 34Ê $
&" exampl es/src/DeadmanSwi tch/DeadmanSwi tch. sol Ê"#####"

```

Figure 2. Unused events

```

wakefdetectfunused-error

! "Ê[INFO][HIGH]ÊUnusedÊErrorÊ[unused-error]Ê"#####"#$
$ 27Ê ////////////////////////////////////// $
$ 28Ê $
$ 29Ê error TooManyTokens(); $
$ %Ê30Ê error Inval i dSqrtPri ceLi mi tX96(); $
$ 31Ê $
$ 32Ê ui nt256 i nternal Êconstant MAX_TOKENS = 100; $
$ 33Ê $
&" exampl es/src/AutoSavi ngs/AutoSavi ngs. sol Ê"#####"

! "Ê[INFO][HIGH]ÊUnusedÊErrorÊ[unused-error]Ê"#####"#$
$ 31Ê event Recovery(address account, address nomi nee); $
$ 32Ê $
$ 33Ê error UnsoportedOperati on(); $
$ %Ê34Ê error Mi ssi ngCondi ti on(); $
$ 35Ê $
$ 36Ê /*//////////////////////////////////// $
$ 37Ê $
&" exampl es/src/DeadmanSwi tch/DeadmanSwi tch. sol Ê"#####"

! "Ê[INFO][HIGH]ÊUnusedÊErrorÊ[unused-error]Ê"#####"#$
$ 20Ê ////////////////////////////////////// $
$ 21Ê $
$ 22Ê error UnauthorizedAccess(); $
$ %Ê23Ê error OwnerAl readyExi sts(address owner); $
$ 24Ê error Inval i dOwner(address owner); $
$ 25Ê $
$ 26Ê $
&" exampl es/src/Ownabl eExecutor/Ownabl eExecutor. sol Ê"#####"

```

Figure 3. Unused errors

```

wakefdetectunused-import

! "É[INFO][HIGH]ÉUnusedÉimportÉ[unused-import]É"#####
$ 19Éimport { LibSort } from "solady/uti ls/Li bSort. sol "; $
$ 20Éimport { IERC7484 } from "modul eki t/src/i nterfaces/I ERC7484. sol "; $
$ 21É $
$ %É22Éimport "forge-std/consol e2. sol "; $
$ 23É $
$ 24É/** $
$ 25É $
&" exampl es/src/HookMul ti Pl exer/HookMul ti Pl exer. sol É"#####

```

Figure 4. Unused imports

```

wakefdetectlocked-ether

! "É[HIGH][MEDIUM]ÉContractÉrecei vesÉetherÉbutÉneverÉsendsÉi t. É[locked-ether]É"#
$ 12É * and pays for gas $
$ 13É * @author Rhi nestone $
$ 14É */ $
$ %É15Écontract Ownabl eExecu tor i s ERC7579Execu torBase { $
$ 16É usi ng Senti nel Li stLi b for Senti nel Li stLi b. Senti nel Li st; $
$ 17É $
$ 18É $
&" exampl es/src/Ownabl eExecu tor/Ownabl eExecu tor. sol É"#####
("É! "ÉThi sÉfuncti onÉcanÉrecei veÉether. É"#####
$ÉÉÉ$ 137É * @param ownedAccount address of the account to execute the transaction on $
$ÉÉÉ$ 138É * @param calldata encoded data containing the transaction to execute $
$ÉÉÉ$ 139É */ $
$ÉÉÉ$ %É140É functi on executeOnOwnedAccount( $
$ÉÉÉ$ 141É address ownedAccount, $
$ÉÉÉ$ 142É bytes calldata calldata $
$ÉÉÉ$ 143É $
$ÉÉÉ&" exampl es/src/Ownabl eExecu tor/Ownabl eExecu tor. sol É"#####
)""É! "ÉThi sÉfuncti onÉcanÉrecei veÉether. É"#####
$ 159É * @param ownedAccount address of the account to execute the transaction on $
$ 160É * @param calldata encoded data containing the transactions to execute $
$ 161É */ $
$ %É162É functi on executeBatchOnOwnedAccount( $
$ 163É address ownedAccount, $
$ 164É bytes calldata calldata $
$ 165É $
&" exampl es/src/Ownabl eExecu tor/Ownabl eExecu tor. sol É"#####

```

Figure 5. Locked ether

```

wakefdetecterc-4337

! "É[HIGH][HIGH]ÉERC-4337: ÉstateÉvari abl eÉi sÉaccessedÉbyÉaÉrestri ctedÉfuncti onÉ[erc-4337]É"#
$ 403É }); $
$ 404É $ $
$ 405É //ÉcheckÉi fÉtheÉsubVal i datorÉdataÉi sÉemptyÉandÉreturnÉfal seÉi fÉi tÉi s $
$ %É406É bytes memory val i datorStorageData = $val i dator. data; $
$ 407É i f (val i datorStorageData. length == 0) { $
$ 408É return false; $
$ 409É $
&" exampl es/src/Mul ti Factor/Mul ti Factor. sol É"#####

```

Figure 6. ERC-4337 storage access violation



# B.2. Graphs

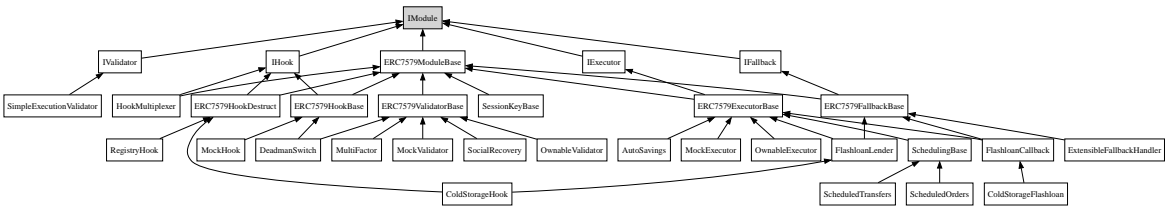


Figure 7. Inheritance graph

