Rhinestone

Core Modules

by Ackee Blockchain

3.7.2024

Contents

1. Document Revisions

0.1	Draft report	24.5.2024
1.0	Final report	5.6.2024
1.1	Fix review	3.7.2024

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run free certification courses School of Solana, Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, RockawayX.

2.2. Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools and Wake is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
5. **Unit and fuzz testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzz tests.

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 (such as deployment scripts, compiler configuration, use of 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

		Likelihood			
		High	**Medium**	**Low**	**-**
Impact	**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 (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), 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 (see above) 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

Member’s Name	Position
Štěpán Šonský	Lead Auditor
Michal Převrátil	Auditor
Naoki Yoshida	Auditor
Jan Převrátil	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 to perform a security review of the Rhinestone protocol with a total time donation of 21 engineering days in a period between April 29 and May 24, 2024, with Štěpán Šonský as the lead auditor.

The audit was performed on the commit 013a123 [1] 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 Tools for Solidity 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 [2]. 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 (H1), removing a hook from a different list (H2), 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 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 Revision 1.0 for the system overview of the codebase.

Revision 1.1

Rhinestone engaged Ackee Blockchain to perform a fix review of Rhinestone on the commit `4531b2e` [3].

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.

[1] full commit hash: 013a123305556392632c3eae9f467dcdc4ccdf6e

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

[3] full commit hash: 4531b2e3fffeeff520bf37fbc4bb49eec726ed61

4. Summary of Findings

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

- a *Description*,
- an *Exploit scenario*,
- a *Recommendation* and if applicable
- a *Fix*.

There might often be multiple ways to solve or alleviate the issue, with varying requirements regarding the necessary changes to the codebase. In that case, we will try to enumerate them all, clarifying which solves the underlying issue better (albeit possibly only with architectural changes) than others.

	Severity	Reported	Status
H1: Missing threshold checks	High	1.0	Fixed
H2: Removing from a wrong array of sigs in `removeSigHook`	High	1.0	Fixed
H3: `OwnableExecutor` locked Ether	High	1.0	Fixed
H4: ERC-4337 restricted storage access	High	1.0	Fixed
H5: `Nominee` have limited access	High	1.0	Fixed
H6: Externally increasable borrower’s nonce	High	1.0	Fixed

	Severity	Reported	Status
H7: ERC-3156 flashloans implementation	High	1.0	Fixed
M1: Missing `sqrtPriceLimitX96` check	Medium	1.0	Fixed
M2: Removing different address	Medium	1.0	Fixed
M3: Missing module type condition	Medium	1.0	Fixed
L1: `HookMultiPlexer` with no hooks	Low	1.0	Fixed
L2: `flashLoan` front-run	Low	1.0	Fixed
L3: Unsafe ERC-20 calls	Low	1.0	Fixed
L4: Missing initialized check in SentinelList	Low	1.0	Fixed
L5: Missing deletion of execution element	Low	1.0	Fixed
L6: Excluding list element	Low	1.0	Fixed
W1: `MultiFactor` duplicate validators	Warning	1.0	Acknowledged
W2: Missing `clearTrustedForwarder` call	Warning	1.0	Fixed
W3: `SchedulingBase` executions count validation	Warning	1.0	Fixed
W4: Missing zero address check	Warning	1.0	Fixed

	Severity	Reported	Status
W5: Missing array length validation	Warning	1.0	Fixed
W6: Missing value check in ERC-20 transfers	Warning	1.0	Fixed
W7: TODOs in module `HookMultiPlexer`	Warning	1.0	Fixed
I1: `AutoSavings` percentage precision	Info	1.0	Fixed
I2: Unused code	Info	1.0	Fixed
I3: Unused variable	Info	1.0	Fixed
I4: Internal functions missing prefix	Info	1.0	Acknowledged
I5: Missing events	Info	1.0	Fixed
I6: Typos and incorrect documentation	Info	1.0	Fixed
I7: Redundant assignments in `SentinelList`	Info	1.0	Fixed
I8: Missing function restriction	Info	1.0	Fixed
I9: Proposal for refactoring `HookMultiPlexer`	Info	1.0	Fixed

Table 2. Table of Findings

5. Report revision 1.0

5.1. 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-3165 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-3165 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.

5.2. 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).

H1: Missing threshold checks

High severity issue

Impact:	High	Likelihood:	Medium
Target:	MultiFactor.sol, OwnableValidator.sol, SocialRecovery.sol	Type:	Data validation, 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

H2: 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

```
    function removeSigHook(address hook, bytes4 sig, HookType hookType) external {
        // cache the account
        address account = msg.sender;
        // check if the module is initialized and revert if it is not
        if (!isInitialized(account)) revert NotInitialized(account);

        // cache the storage config
        Config storage $config = $getConfig(account);

        if (hookType == HookType.SIG) {
            // get the length of the hooks for the same sig
            uint256 sigsHooksLength = $config.sigHooks[sig].length;
            // delete the hook
            $config.sigHooks[sig].popAddress(hook);

            // if there is only one hook for the sig, remove the sig
            if (sigsHooksLength == 1) {
                $config.targetSigs.popBytes4(sig);
            }
        } else if (hookType == HookType.TARGET_SIG) {
            // get the length of the hooks for the same sig
            uint256 targetSigsHooksLength = $config.targetSigHooks[sig].length;
            // delete the hook
            $config.targetSigHooks[sig].popAddress(hook);

            // if there is only one hook for the sig, remove the sig
            if (targetSigsHooksLength == 1) {
                $config.targetSigs.popBytes4(sig);
            }
        } else {
            revert UnsupportedHookType();
        }
    }
```

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 occurence 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 I9.

Go back to Findings Summary

H3: OwnableExecutor locked Ether

High severity issue

Impact:	High	Likelihood:	Medium
Target:	OwnableExecutor.sol	Type:	Loss of funds

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

```
    function executeOnOwnedAccount(
        address ownedAccount,
        bytes calldata callData
    )
        external
        payable
    {
        // check if the sender is an owner
        if (!accountOwners[ownedAccount].contains(msg.sender)) {
            revert UnauthorizedAccess();
        }

        // execute the transaction on the owned account
        IERC7579Account(ownedAccount).executeFromExecutor(ModeLib.encodeSimpleSingle(), callData);
    }

    /**
     * Executes a batch of transactions on the owned account
     *
     * @param ownedAccount address of the account to execute the transaction on
     * @param callData encoded data containing the transactions to execute
     */
    function executeBatchOnOwnedAccount(
        address ownedAccount,
        bytes calldata callData
    )
        external
        payable
    {
        // check if the sender is an owner
        if (!accountOwners[ownedAccount].contains(msg.sender)) {
            revert UnauthorizedAccess();
        }

        // execute the batch of transaction on the owned account
        IERC7579Account(ownedAccount).executeFromExecutor(ModeLib.encodeSimpleBatch(), callData);
    }
```

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:	EIP compliance

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

```
        SubValidatorConfig storage $validator = $subValidatorData({
            account: account,
            iteration: iteration,
            subValidator: validatorAddress,
            id: id
        });

        // check if the subValidator data is empty and return false if it is
        bytes memory validatorStorageData = $validator.data;
```

Listing 4. Excerpt from DataTypes.sol

```
struct SubValidatorConfig {
    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

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

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

Listing 6. Excerpt from DeadManSwitch

```
    function _preCheck(
        address account,
        address,
        uint256,
        bytes calldata
    )
        internal
        override
        returns (bytes memory hookData)
    {
        // if the module is not initialized, return and dont update the last access time
        if (!isInitialized(account)) return "";

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

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:	Data validation, 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

```
        address borrower,
        address, /*token*/
        uint256, /*amount*/
        uint256, /*fee*/
        bytes calldata data
    )
        external
        onlyAllowedCallbackSender
        returns (bytes32)
    {
        // decode the data
        (FlashLoanType flashLoanType, bytes memory signature, Execution[]
memory executions) =
            abi.decode(data, (FlashLoanType, bytes, Execution[]));
        // get the hash
        bytes32 hash = getTokengatedTxHash(flashLoanType, executions,
nonce[borrower]);
        // increment the nonce
        nonce[borrower]++;
        // 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

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

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

Listing 9. Excerpt from ColdStorageFlashloan

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

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:	Bad implementation

Description

ERC-3156 flashloans implementation does not follow the reference implementation and best practices to avoid security threats. According to ERC-3156 specification 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 ColdStorageHook

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

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 {
    // remove the address from the whitelist
    whitelist[msg.sender].pop(prevAddress, addressToRemove);
}
```

Fix 1.1

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

```
whitelist[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]()

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

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

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

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

```
    function addHook(address hook, HookType hookType) external {
        // cache the account
        address account = msg.sender;
        // check if the module is initialized and revert if it is not
        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-runnig

Description

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

Listing 14. Excerpt from [FlashloanLender]

```
    function flashLoan(
        IERC3156FlashBorrower receiver,
        address token,
        uint256 value,
        bytes calldata data
    )
```

Listing 15. Excerpt from [FlashloanCallback]

```
    function onFlashLoan(
        address borrower,
        address, /*token*/
        uint256, /*amount*/
        uint256, /*fee*/
        bytes calldata data
    )
```

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:	Non-standard tokens

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

```
        // get the execution hash
        bytes32 executionHash = _execDigest(target, value, callData);

        // check the timelocked execution
        _checkTimelockedExecution(account, executionHash);

        // emit the TimelockExecuted event
        emit TimelockExecuted(account, executionHash);

        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 `onUnistall` 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
LinkedList_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
LinkedList_InvalidEntry(start);
```

Fix 1.1

The finding was fixed by inverting the condition.

Go back to Findings Summary

W1: `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.

Fix 1.1

Acknowledged.

> This is a feature - validators should be able to be re-used.
>
> — Rhinestone

Go back to Findings Summary

W2: 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

W3: 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

W4: 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

W5: 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

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 ERC-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: 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]

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

Listing 18. Excerpt from [HookMultiplexer]

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

Recommendation

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

Fix 1.1

The implementation contains no more `TODO`s. This module was refactored as proposed in finding I9.

Go back to Findings Summary

I1: 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

```
    function calcDepositAmount(
        uint256 amountReceived,
        uint256 percentage
    )
        public
        pure
        returns (uint256)
    {
        // calculate the amount to be saved which is the
        // percentage of the amount received
        return (amountReceived * percentage) / 100;
    }
```

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

I2: 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

I3: 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 20. Excerpt from [ColdStorageHook]

```
    function checkHash(
        address account,
        bytes32 hash
    )
        external
        view
        returns (bytes32 executeAfter)
    {
        // get the executeAfter timestamp
        bool success;
        (success, executeAfter) = executions[account].tryGet(hash);
    }
```

Recommendation

Remove the `success` variable.

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

Fix 1.1

Fixed. The `success` variable was removed.

[Go back to Findings Summary]

I4: Internal functions missing prefix

Impact:	Info	Likelihood:	N/A
Target:	ERC7579HookDestruct.sol	Type:	Best practices

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.

Solution 1.1

Acknowledged.

> Would break existing modules from external developers.
>
> — Rhinestone

[Go back to Findings Summary]

I5: Missing events

Impact:	Info	Likelihood:	N/A
Target:	`**/*`	Type:	Events

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

I6: 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 `UnsopportedOperation`.
- 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

I7: 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

I8: 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 21. Excerpt from SocialRecovery

```
    function validateUserOp(
        PackedUserOperation calldata userOp,
        bytes32 userOpHash
    )
        external
        override
        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

I9: 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 H2: 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](#)

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Rhinestone: Core Modules, 3.7.2024.

Appendix B: Glossary of terms

The following terms might be used throughout the document:

Superclass/Ancestor of C

A contract that C inherits/derives from.

Subclass/Child of C

A contract that inherits/derives from C.

Syntactic contract

A Solidity contract. May have an inheritance chain, and may be deployed.

Deployed contract

An EVM account with non-zero code. If its source was written in Solidity, it was created through at least one syntactic contract. If that contract had superclasses (parents), it would be composed of multiple syntactic contracts.

Init/initialization function

A non-constructor function that serves as an initializer. Often used in upgradeable contracts.

External entrypoint

A `public` or `external` function.

Public/Publicly-accessible function/entrypoint

An `external` or `public` function that can be successfully executed by any network account.

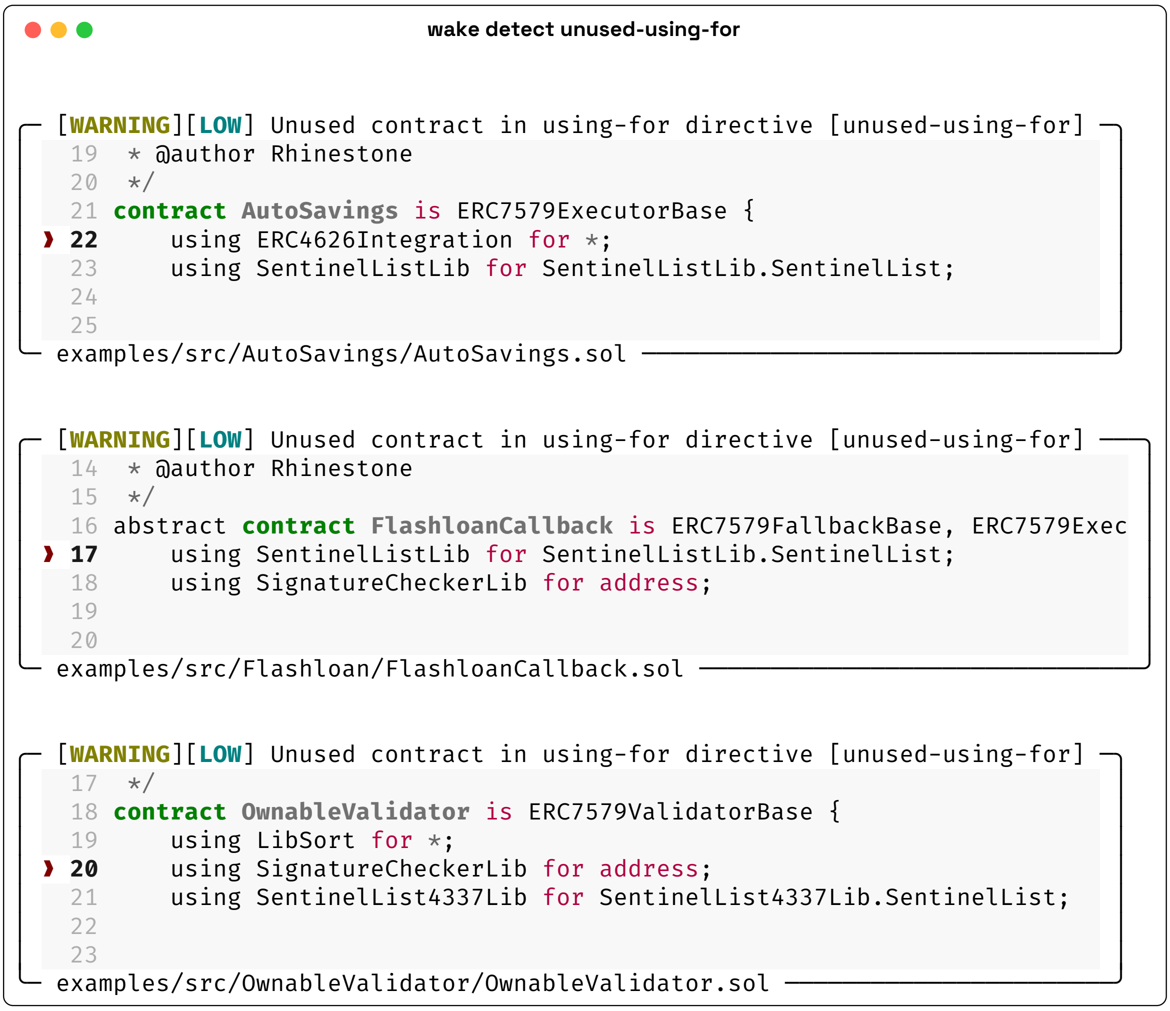
Mutating function

A non-`view` and non-`pure` function.

Appendix C: Wake outputs

This section lists the outputs from the Wake tool used during the audit.

C.1. Detectors



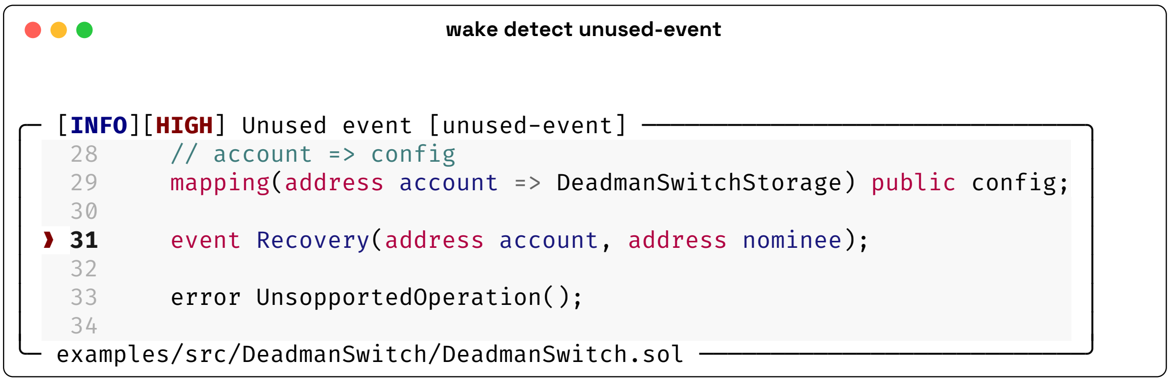
wake detect unused-using-for

```
╭─ [WARNING][LOW] Unused contract in using-for directive [unused-using-for] ─╮
│   19   * @author Rhinestone                                                │
│   20   */                                                                  │
│   21 contract AutoSavings is ERC7579ExecutorBase {                         │
│ ❱ 22     using ERC4626Integration for *;                                   │
│   23     using SentinelListLib for SentinelListLib.SentinelList;           │
│   24                                                                       │
│   25                                                                       │
╰─ examples/src/AutoSavings/AutoSavings.sol ─────────────────────────────────╯

╭─ [WARNING][LOW] Unused contract in using-for directive [unused-using-for] ───╮
│   14   * @author Rhinestone                                                  │
│   15   */                                                                    │
│   16 abstract contract FlashloanCallback is ERC7579FallbackBase, ERC7579Exec │
│ ❱ 17     using SentinelListLib for SentinelListLib.SentinelList;             │
│   18     using SignatureCheckerLib for address;                              │
│   19                                                                         │
│   20                                                                         │
╰─ examples/src/Flashloan/FlashloanCallback.sol ───────────────────────────────╯

╭─ [WARNING][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 SentinelList4337Lib for SentinelList4337Lib.SentinelList;   │
│   22                                                                       │
│   23                                                                       │
╰─ examples/src/OwnableValidator/OwnableValidator.sol ───────────────────────╯
```

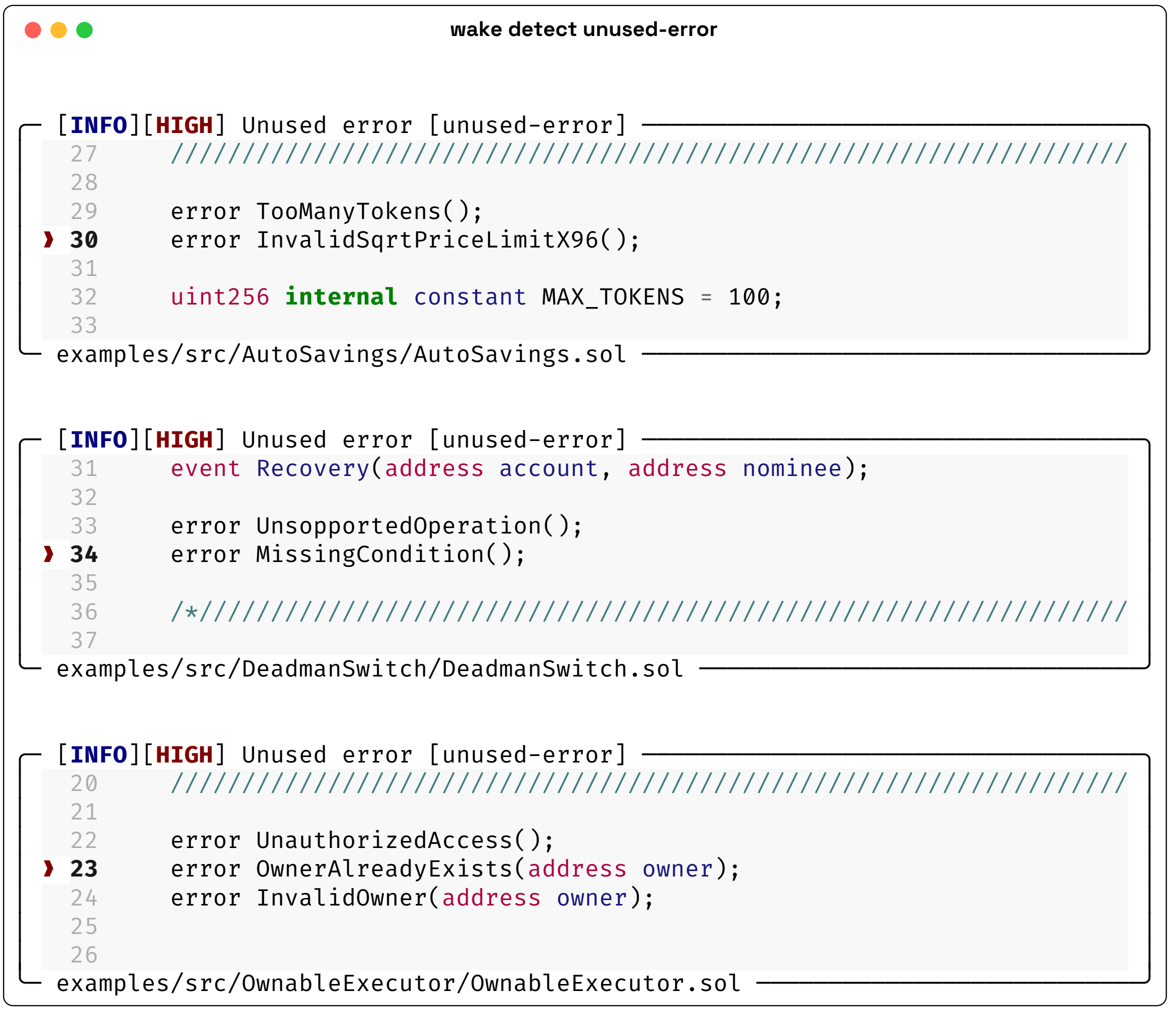
Figure 1. Unused using-for directives



wake detect unused-event

```
╭ [INFO][HIGH] Unused event [unused-event] ──────────────────────────╮
│  28      // account => config                                      │
│  29      mapping(address account => DeadmanSwitchStorage) public config;
│  30                                                                │
│❱ 31      event Recovery(address account, address nominee);        │
│  32                                                                │
│  33      error UnsopportedOperation();                             │
│  34                                                                │
╰ examples/src/DeadmanSwitch/DeadmanSwitch.sol ──────────────────────╯
```

Figure 2. Unused events



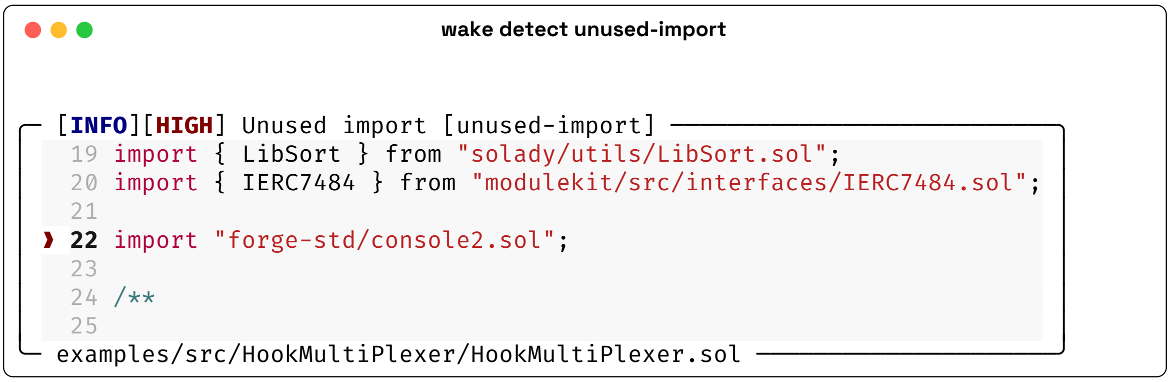
wake detect unused-error

```
╭ [INFO][HIGH] Unused error [unused-error] ──────────────────────────────╮
│  27      ///////////////////////////////////////////////////////////////////
│  28                                                                    │
│  29      error TooManyTokens();                                        │
│❱ 30      error InvalidSqrtPriceLimitX96();                             │
│  31                                                                    │
│  32      uint256 internal constant MAX_TOKENS = 100;                   │
│  33                                                                    │
╰ examples/src/AutoSavings/AutoSavings.sol ──────────────────────────────╯

╭ [INFO][HIGH] Unused error [unused-error] ──────────────────────────────╮
│  31      event Recovery(address account, address nominee);             │
│  32                                                                    │
│  33      error UnsopportedOperation();                                 │
│❱ 34      error MissingCondition();                                     │
│  35                                                                    │
│  36      /*/////////////////////////////////////////////////////////////////
│  37                                                                    │
╰ examples/src/DeadmanSwitch/DeadmanSwitch.sol ──────────────────────────╯

╭ [INFO][HIGH] Unused error [unused-error] ──────────────────────────────╮
│  20      ///////////////////////////////////////////////////////////////////
│  21                                                                    │
│  22      error UnauthorizedAccess();                                   │
│❱ 23      error OwnerAlreadyExists(address owner);                      │
│  24      error InvalidOwner(address owner);                            │
│  25                                                                    │
│  26                                                                    │
╰ examples/src/OwnableExecutor/OwnableExecutor.sol ──────────────────────╯
```

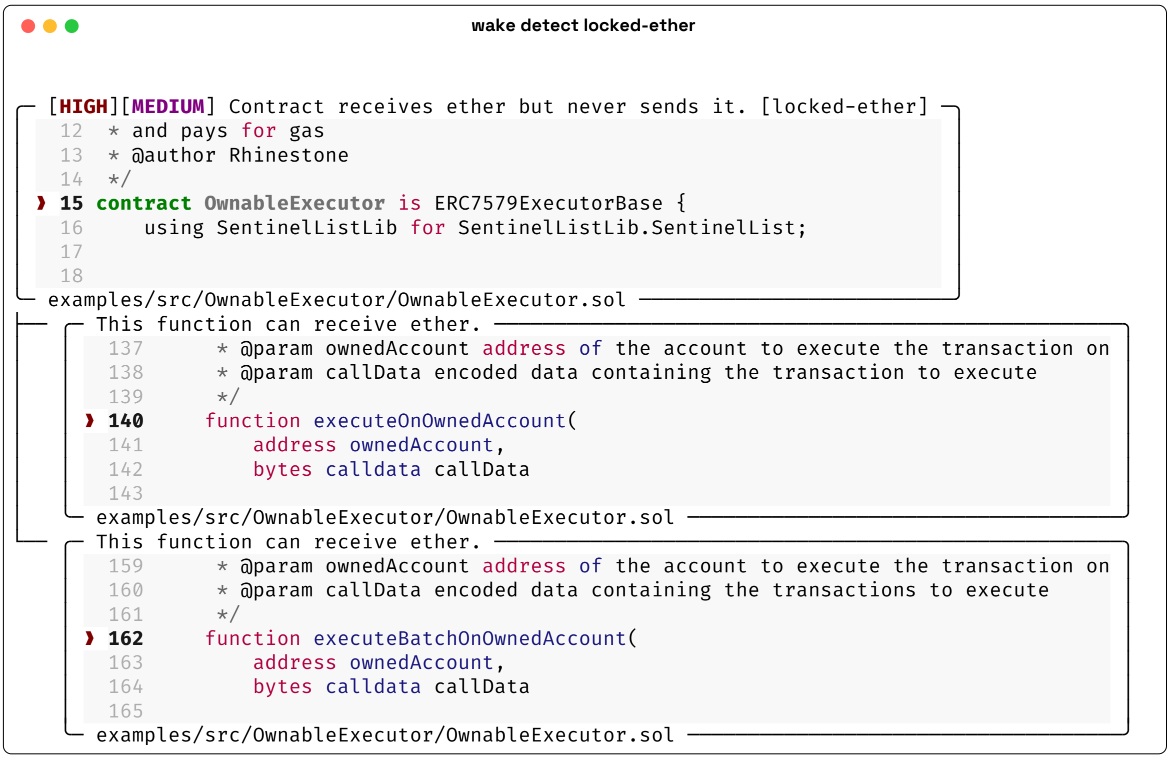
Figure 3. Unused errors



wake detect unused-import

```
╭─ [INFO][HIGH] Unused import [unused-import] ──────────────────────────╮
│    import { LibSort } from "solady/utils/LibSort.sol";                 │
│    import { IERC7484 } from "modulekit/src/interfaces/IERC7484.sol";   │
│                                                                        │
│ ❱  import "forge-std/console2.sol";                                    │
│                                                                        │
│    /**                                                                 │
│                                                                        │
╰─ examples/src/HookMultiPlexer/HookMultiPlexer.sol ─────────────────────╯
```

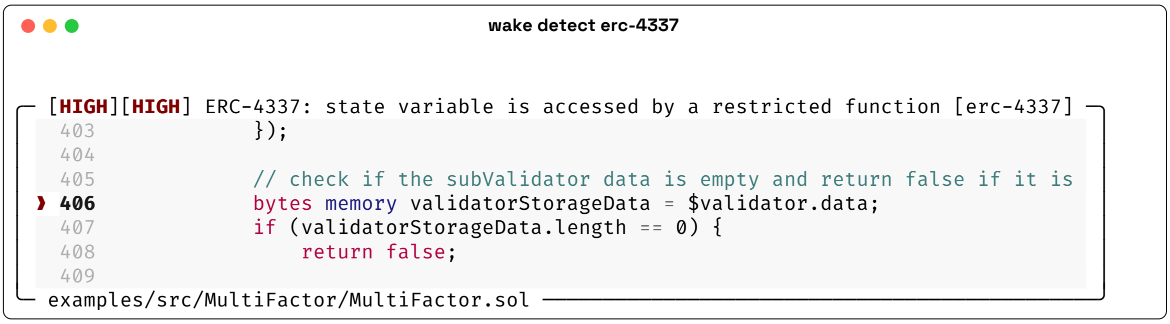
Figure 4. Unused imports



wake detect locked-ether

```
╭─ [HIGH][MEDIUM] Contract receives ether but never sends it. [locked-ether] ─╮
│    * and pays for gas                                                        │
│    * @author Rhinestone                                                      │
│    */                                                                        │
│ ❱  contract OwnableExecutor is ERC7579ExecutorBase {                         │
│        using SentinelListLib for SentinelListLib.SentinelList;              │
│                                                                              │
│                                                                              │
╰─ examples/src/OwnableExecutor/OwnableExecutor.sol ──────────────────────────╯
├── ╭─ This function can receive ether. ─────────────────────────────────────────────╮
│   │        * @param ownedAccount address of the account to execute the transaction on │
│   │        * @param callData encoded data containing the transaction to execute      │
│   │        */                                                                        │
│   │ ❱      function executeOnOwnedAccount(                                          │
│   │            address ownedAccount,                                                 │
│   │            bytes calldata callData                                               │
│   │                                                                                  │
│   ╰─ examples/src/OwnableExecutor/OwnableExecutor.sol ─────────────────────────────╯
└── ╭─ This function can receive ether. ─────────────────────────────────────────────╮
    │        * @param ownedAccount address of the account to execute the transaction on │
    │        * @param callData encoded data containing the transactions to execute     │
    │        */                                                                        │
    │ ❱      function executeBatchOnOwnedAccount(                                     │
    │            address ownedAccount,                                                 │
    │            bytes calldata callData                                               │
    │                                                                                  │
    ╰─ examples/src/OwnableExecutor/OwnableExecutor.sol ─────────────────────────────╯
```

Figure 5. Locked ether



wake detect erc-4337

```
╭ [HIGH][HIGH] ERC-4337: state variable is accessed by a restricted function [erc-4337] ╮
│   403              });                                                                  │
│   404                                                                                   │
│   405              // check if the subValidator data is empty and return false if it is │
│ ❱ 406              bytes memory validatorStorageData = $validator.data;                 │
│   407              if (validatorStorageData.length == 0) {                              │
│   408                  return false;                                                    │
│   409                                                                                   │
╰ examples/src/MultiFactor/MultiFactor.sol ───────────────────────────────────────────────╯
```

Figure 6. ERC-4337 storage access violation

C.2. Graphs

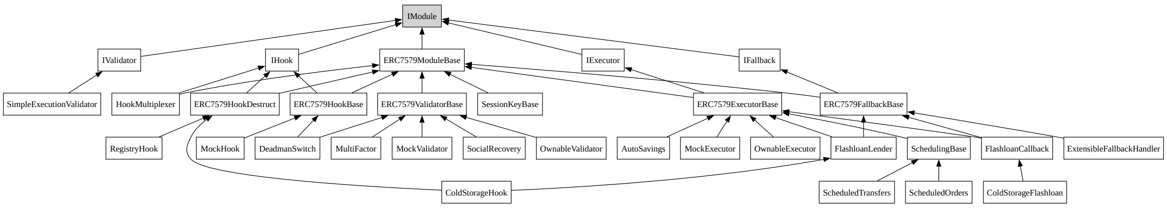


Figure 7. Inheritance graph

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://twitter.com/AckeeBlockchain