Rhinestone

Safe7579

by Ackee Blockchain

5.7.2024

Contents

ackee | blockchain security

1. Document Revisions

1.0	Final report	14.6.2024
1.1	Fix review	20.6.2024
2.0	Final report	5.7.2024

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
Lukáš Rajnoha	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 16 engineering days in a period between June 3 and June 14, 2024, with Štěpán Šonský as the lead auditor.

The audit was performed on the commit 90dd363 [1] and the scope was the following:

- core/
  - AccessControl.sol
  - ExecutionHelper.sol
  - Initializer.sol
  - ModuleManager.sol
  - RegistryAdapter.sol
  - SetupDCUtil.sol
- lib/
  - ExecutionLib.sol
  - ModeLib.sol
- utils/
  - DCUtil.sol
  - Safe7579UserOperationBuilder.sol
- DataTypes.sol
- Safe7579.sol

- Safe7579Launchpad.sol

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. During the review, we paid special attention to:

- checking Safe deployment using the Launchpad,
- checking module management logic and multi-type module installation,
- checking fallback handler implementation,
- checking possible DoS scenarios,
- checking front-running possibilities,
- ensuring delegatecalls are used correctly,
- detecting possible reentrancies in the code,
- checking compliance with used ERCs,
- ensuring access controls are not too relaxed or too strict,
- looking for common issues such as data validation.

Our review resulted in 28 findings, ranging from Informational to Critical severity. The most severe one allows the attacker to front-run the Safe deployment using the Launchpad and take over control of smart wallets created using it (see C1). Other high-severity issues refer to the `Safe7579.initializeAccount` function front-running (H1) and the wrong context used in the `withRegistry` modifier in the `Safe7579.executeFromExecutor` function (H2). Medium issues are mostly overlooked trivial mistakes. The overall code quality is average, the codebase contains TODOs, the unused code and also the project is not fully covered by NatSpec documentation.

Ackee Blockchain recommends Rhinestone:

- fix newly deployed Safe takeover possibility,
- protect the `Safe7579.initializeAccount` function from front-running,
- fix the context in the `withRegistry` modifier in `Safe7579.executeFromExecutor` function,
- fix SIG hooks overriding,
- fix issues with the `success` return values in batch execution,
- call module `onInstall` function during `_initModule` process,
- resolve all TODOs and remove unused code,
- cover the codebase with NatSpec documentation,
- address all other reported issues,
- also, we recommend performing continuous internal peer code reviews.

See Revision 1.0 for the system overview of the codebase.

Revision 1.1

The review was done on the given commit: `180f0ac`. 20 issues were fixed, 3 issues acknowledged, and 1 issue (W7) was not fixed. We recommend to fix the remaining issue. The fix review was focused only on issues remediations, other code changes (if any) were not audited.

See Revision 1.1 for the review of the updated codebase and additional information we consider essential for the current scope.

Revision 2.0

Rhinestone engaged Ackee Blockchain to perform an incremental security review of an updated version of the Safe7579 module with a total time donation of 3 engineering days in a period between July 2 and July 5, 2024, with Michal Převrátil as the lead auditor.

The scope of the audit was the changes to all files in the `src/` directory in the #7 pull request, with `d961421` [2] being the latest commit. Namely, the following files were reviewed:

- src/ISafe7579.sol
- src/Safe7579.sol
- src/Safe7579Launchpad.sol
- src/core/Initializer.sol
- src/core/ModuleManager.sol
- src/core/SafeOp.sol
- src/core/SupportViewer.sol
- src/interfaces/IERC7579Account.sol
- src/interfaces/ISafe.sol
- src/interfaces/ISafeOp.sol

See Revision 2.0 for a detailed description of the changes.

Our review began with static analysis using the Wake tool, yielding the I5 finding. We then performed a manual review and fuzzing, focusing on the changes in the `src/` directory. We paid special attention to:

- making sure no new vulnerabilities were introduced,
- code quality remained at high standards,
- the code behaves within the expectations of Safe users.

The review resulted in 2 warnings and 2 informational findings.

Ackee Blockchain recommends Rhinestone:

- strive to achieve maximum compatibility and consistency with the Safe ecosystem as well as the ERC standards,
- perform peer code reviews to ensure the code quality remains high.

[1] full commit hash: 90dd3637f26352158a2aa4562d75d47dada9fcbe

[2] full commit hash: d961421641b826f6c970e13b0af18111ec10ebad

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
C1: ERC-4337 counterfactual address can be stolen	Critical	1.0	Fixed
H1: `initializeAccount` vulnerable to front-running	High	1.0	Fixed
H2: Executors cannot be used	High	1.0	Fixed
M1: Missing event and `onInstall` call in `_initModules`	Medium	1.0	Fixed
M2: `BatchedExecUtil._tryExecute` inverted `success`	Medium	1.0	Fixed

	Severity	Reported	Status
M3: `BatchedExecUtil.tryExecute` single return value	Medium	1.0	Fixed
M4: `ModuleManager._installHook` SIG hook overwriting	Medium	1.0	Fixed
M5: Locked Ether	Medium	1.0	Partially fixed
L1: Fallback handler `CallType` validation	Low	1.0	Fixed
L2: Domain-specific message encoding missing with `signedMessages`	Low	1.0	Fixed
L3: ERC-4337 factory standard violation	Low	1.0	Acknowledged
L4: `_multiTypeInstall` module type validation	Low	1.0	Fixed
W1: `postCheck` function is different from the EIP-7579 interface	Warning	1.0	Fixed
W2: `uninstallModule` reverts on multi-type module	Warning	1.0	Fixed
W3: Hooks can prevent module uninstallation	Warning	1.0	Fixed
W4: Missing data validations	Warning	1.0	Fixed
W5: Underscore prefixed public function	Warning	1.0	Fixed

	Severity	Reported	Status
W6: Hardcoded Enum.Operation values	Warning	1.0	Fixed
W7: Incomplete unused Safe7579UserOperationBuilder	Warning	1.0	Not fixed
W8: Missing `TryExecutionFailed` emits	Warning	1.0	Fixed
I1: Duplicated code	Info	1.0	Acknowledged
I2: Unused code	Info	1.0	Fixed
I3: Typos and incorrect documentation	Info	1.0	Fixed
I4: Code structure	Info	1.0	Acknowledged
W9: Safe does not implement validator interface	Warning	2.0	Reported
W10: Inconsistent signature checking	Warning	2.0	Reported
I5: Unused using-for	Info	2.0	Reported
I6: Typo	Info	2.0	Reported

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.

Safe7579.sol

The `Safe7579` contract is an adapter for Safe smart accounts to ensure full ERC-7579 compliance. It is registered as both a module and a fallback handler on the Safe. The `validateUserOp` function is provided for transaction verification by an ERC-4337 `Entrypoint`. To support ERC-1271, the `isValidSignature` function uses Safe's signed messages and `checkSignatures` features or ERC-7579 validation modules. Handling ERC-7579 operations and their specific execution modes is done by the functions `execute` (intended to be called by the `Entrypoint`) and `executeFromExecutor` (intended to be called by executor modules). The functions `installModule`, `uninstallModule`, and `isModuleInstalled` are responsible for module management. Additional helper functions such as `supportsExecutionMode`, `supportsModule`, and `accountId` are available.

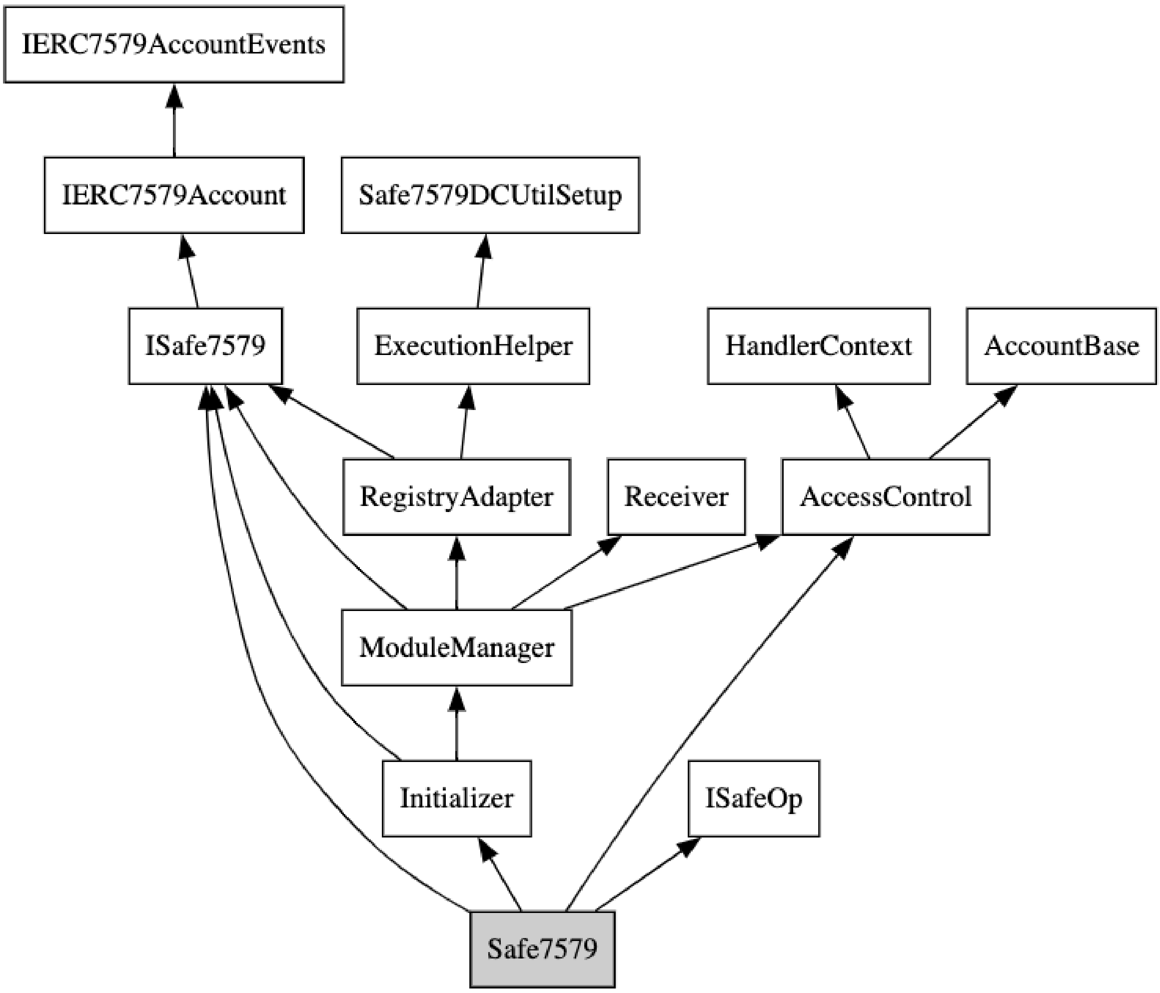

Figure 1. Safe7579 inheritance graph

Safe7579Launchpad.sol

The `Safe7579Launchpad` contract deploys a new Safe7579-enabled Safe smart account. After the `SafeProxy` is created, its singleton points to this `Safe7579Launchpad` during the validation phase until it is changed to the `SafeSingleton` in the execution phase. The contract inherits its storage structure from `SafeStorage` to match the `SafeSingleton`. It includes functions for initializing a new Safe7579 account, namely `initSafe7579` (configures the `Safe7579` with all module types except validators), `preValidationSetup` (stores the `initHash` used during the validation phase), and `setupSafe` (upgrades the `SafeProxy` to an actual `SafeSingleton`). The `validateUserOp` function is included for transaction verification by an [ERC-4337](ERC-4337) `Entrypoint` and initializing validators. Other helper functions, such as `predictSafeAddress` and `hash`, are also provided.

Initializer.sol

The `Initalizer` contains functions for initializing Safe7579 for Safe smart accounts, mainly `launchpadValidators` and `initializeAccount`.

AccessControl.sol

`AccessControl` includes modifiers restricting access, namely `onlyEntryPointOrSelf` and `onlyEntryPoint`. Additionally, it includes helper functions such as `entryPoint`, which returns the address of the `Entrypoint` contract supported by the `Safe7579` contract, and `_msgSender`, which allows fetching the original caller address (used in combination with Safe's `FallbackHandler`).

ModuleManager.sol

The `ModuleManager` contract handles the lifecycle of modules. It provides functions for installing and removing all module types. Functions and modifiers for module execution are also provided, such as running hooks before and after execution and calling associated fallback handlers. Additional helper functions are included to check if a module is installed and retrieve module data. The contract also adds the storage necessary for module management.

RegistryAdapter.sol

The `RegistryAdapter` contract is an ERC-7484 compliant adapter for registries. It includes functions, modifiers, and storage for registry management and handles checks to ERC-7484 registries.

ExecutionHelper.sol

The `ExecutionHelper` contract provides an abstraction layer for interactions on the `Safe7579` contract with other modules and external contracts. The `Safe7579` contract is installed as a module on the Safe smart account. Since all interactions with modules must originate from `SafeProxy`, the `execTransactionFromModule` function is utilized instead of direct calls by the `Safe7579` contract. The `ExecutionHelper` provides an abstraction layer above these interactions, offering functions `_exec` and `_delegatecall`, along with their `try` (does not revert on failure) and `return` (returns data from the call) variants. The `ExecutionHelper` also supports batched executions by forwarding such calls to the `Safe7579DCUtil`, which handles the batch execution logic.

DCUtil.sol

This file contains the `ModuleInstallUtil`, `BatchedExecUtil`, and `Safe7579DCUtil` contracts. The `ModuleInstallUtil` contains functions used for module initialization/deinitialization, invoking the necessary function on the module and emitting an event. The `BatchedExecUtil` contains functions for batched transaction execution. The `Safe7579DCUtil` inherits both contracts and adds a function for making static calls. `Safe7579DCUtil` is used by the `ExecutionHelper` contract.

SetupDCUtil.sol

The `SetupDCUtil` contract is inherited by `ExecutionHelper` and is used to create a new `Safe7579DCUtil` contract during its creation. The `ExecutionHelper` contract then utilizes the `Safe7579DCUtil` contract for batched executions and during module initialization or deinitialization.

ExecutionLib.sol

The `ExecutionLib` library contains functions for encoding and decoding

`Execution` data. `Execution` is a struct used to encode executions for the `Safe7579` contract.

ModeLib.sol

ERC-7579 enabled accounts to use custom execution modes. The execution mode is encoded in the `ModeCode` (32 bytes), which consists of `CallType`, `ExecType`, `ModeSelector`, and `ModePayload`. The `ModeLib` library contains functions for encoding and decoding `ModeCode` data, along with additional helper functions.

DataTypes.sol

`DataTypes` contains structs used by the project, namely `FallbackHandler`, `HookType`, `ModuleInit` and `RegistryInit`.

Actors

ERC-4337 entry point

ERC-4337 entry point is a permissionless contract used by userOp bundlers to execute user operations. The entry point interacts with ERC-4337 factories, calls the `validateUserOp` function on smart wallets and executes user operations on the smart wallets.

ERC-4337 factory

ERC-4337 factory is a helper contract responsible for deploying new smart wallets in the initial phase of userOp using CREATE2 or a similar mechanism.

Safe7579Launchpad

Safe7579Launchpad is a contract serving as an initial implementation of a newly deployed smart wallet. During the userOp execution, the contract sets up the smart wallet with the Safe7579 module and changes the smart wallet's singleton to the Safe singleton.

Safe7579 Safe module

Safe7579 is a permissionless contract working as a Safe module and fallback handler, operating only on the `msg.sender` basis.

ERC-7579 modules

ERC-7579 modules are extensions to the Safe smart account that can be installed by the smart wallet owners. These modules may perform various operations on the smart wallet, such as executing transactions, validating user operations, or managing other changes on the smart wallet.

ERC-7484 registry

ERC-7484 registry is a contract that stores information about ERC-7579 modules and their capabilities.

ERC-7484 attesters

ERC-7484 attesters are subjects providing attestations about the capabilities of ERC-7579 modules.

5.2. Trust Model

Smart wallet owners have to trust ERC-7579 modules they install on their smart wallets. The trust is supported by using an ERC-7484 registry with a set of attesters the smart wallet owners trust.

Smart wallet owners can trust (if implemented correctly) the entry point and factory to deploy a new smart wallet at a given address with their full control, allowing them to send Ether and grant access controls to the pre-computed address in advance.

Other actors should have no control over the smart wallet, as the smart wallet is controlled by the owners and the ERC-7579 modules the owners install.

C1: ERC-4337 counterfactual address can be stolen

Critical severity issue

Impact:	High	Likelihood:	High
Target:	Safe7579Launchpad.sol	Type:	Front-running, Double initialization

Description

ERC-4337 allows a smart wallet to be created as a part of the first userOp execution on the wallet. For this purpose, the `Safe7579Launchpad` contract is used, as it serves as the smart wallet initial implementation contract. The full execution flow is as follows:

1. A user builds a userOp creating a new Safe account with ERC-7579 enabled features and broadcasts it to bundlers.
2. A bundler collects the userOp and executes it later on the ERC-4337 entry point.
3. The entry point calls `SenderCreator` (a helper contract) to deploy a new smart wallet.
4. `SenderCreator` calls `SafeProxyFactory` which creates a new Safe proxy and sets the singleton (contract implementation address) to the address of `Safe7579Launchpad`.
5. `SafeProxyFactory` also calls `Safe7579Launchpad.preValidationSetup` on the just created Safe proxy, storing the initialization data hash and performing an optional delegatecall.
6. The entry point execution continues as with any other userOp, calling

`validateUserOp` on the Safe proxy.

7. This involves checking the initialization data hash stored in the userOp against the one stored in the Safe proxy, setting up [ERC-7579](ERC-7579) validators and using one of them to validate the userOp.
8. Bootstrapping of the smart wallet finishes with the call to `Safe7579Launchpad.setupSafe`, changing the singleton address to the Safe singleton (implementation) contract and configuring Safe owners and threshold from the initialization data.

However, due to a missing check for double initialization in `Safe7579Launchpad.preValidationSetup`, it is possible to bypass the standard execution flow and take full control of the address of the smart wallet created in step 4.

Listing 1. Excerpt from Safe7579Launchpad

```
    function preValidationSetup(
        bytes32 initHash,
        address to,
        bytes calldata preInit
    )
        external
        onlyProxy
    {
        // sstore inithash
        _setInitHash(initHash);

        // if a delegatecall target is provided, SafeProxy will execute a delegatecall
        if (to != address(0)) {
            (bool success,) = to.delegatecall(preInit);
            if (!success) revert PreValidationSetupFailed();
        }
    }
```

Exploit scenario

1. An attacker observes userOp pools, waiting for a userOp creating a new Safe account using `Safe7579Launchpad`.
2. The attacker uses the same data payload to be passed to `SenderCreator`, but calls the `SenderCreator` contract directly, bypassing the ERC-4337 entry point.
3. A new Safe proxy is created with the same storage data and at the same address as in step 4 of the standard execution flow.
4. The attacker calls `Safe7579Launchpad.preValidationSetup` on the newly created Safe proxy, using any value for `initHash` but setting `to` and `preInit` so that the function performs delegatecall to a malicious contract that can perform arbitrary actions, including changing the singleton to the Safe singleton and configuring Safe owners and threshold.

The address of the smart wallet may be pre-funded with Ether to pay for the userOp execution, and so this issue may lead to a loss of funds. Also, the legitimate user may assume the smart wallet to be created under their control and set access controls in other contracts to the smart wallet address prior to the smart wallet deployment. The attacker may then take control of the smart wallet and access the other contracts.

See Appendix C for a proof of concept script in the Wake development and testing framework.

Recommendation

Add the following check to `Safe7579Launchpad.preValidationSetup` to prevent double initialization:

```
require(_initHash() == bytes32(0), "Safe7579Launchpad: already initialized");
```

Fix 1.1

Fixed by implementing the recommendation:

```
if (getInitHash() != bytes32(0)) revert Safe7579LaunchpadAlreadyInitialized();
```

[Go back to Findings Summary]

H1: `initializeAccount` vulnerable to front-running

High severity issue

Impact:	High	Likelihood:	Medium
Target:	Safe7579.sol, Safe7579Launchpad.sol	Type:	Front-running

Description

The execution flow enabling ERC-7579 features on existing Safe accounts involves:

- enabling the `Safe7579` contract as a Safe module,
- setting the `Safe7579` contract as the fallback handler on the Safe,
- calling `Safe7579.initializeAccount` through the configured fallback handler.

The first two steps may be performed in the reversed order. However, to prevent the Safe account from being stolen, it is important to set the fallback handler and call the `initializeAccount` function in the same transaction.

Exploit scenario

Safe owners decide to set up `Safe7579` on an existing account. Not being aware of security implications, they perform the three steps, each in a different transaction.

A malicious actor monitors transaction pools and front-runs the `initializeAccount` call with their own data, taking full control of the Safe account.

Recommendation

Create a helper launchpad function that performs all three steps in a single transaction. Document the importance of performing all three steps in a single transaction and guide users to use the launchpad function.

Fix 1.1

The finding was fixed by adding the `onlyEntryPointOrSelf` modifier to the `initializeAccount` function as well as the `launchpadValidators` function, which may also be abused for front-running.

Go back to Findings Summary

H2: Executors cannot be used

High severity issue

Impact:	High	Likelihood:	Medium
Target:	Safe7579.sol	Type:	Bad implementation

Description

The function `Safe7579.executeFromExecutor` allows ERC-7579 executor modules to execute operations on behalf of Safe smart accounts.

Listing 2. Excerpt from Safe7579

```
    function executeFromExecutor(
        ModeCode mode,
        bytes calldata executionCalldata
    )
        external
        payable
        override
        onlyExecutorModule
        withHook(IERC7579Account.executeFromExecutor.selector)
        withRegistry(msg.sender, MODULE_TYPE_EXECUTOR)
        returns (bytes[] memory returnDatas)
```

The `withRegistry` modifier should check that the sender module is attested as an executor module by trusted attesters with a given threshold. However, the execution always fails because the check is performed for the address of the Safe smart account (the address of `SafeProxy`) and not the address of the executor module.

Exploit scenario

Safe owners install a new executor module for automated token transfers.

The executor module calls the `executeFromExecutor` function on the Safe account (`SafeProxy`). The execution drops to the Safe fallback handler, which calls `Safe7579.executeFromExecutor` from the Safe account as an external call. Given this execution, `msg.sender` used in the `withRegistry` modifier is the address of the Safe account, while the executor module address is encoded at the end of the call data.

Recommendation

Replace `msg.sender` with `_msgSender()` in the `withRegistry` modifier to check the executor module address and allow executor modules to be used.

Fix 1.1

The finding was fixed by replacing `msg.sender` with `_msgSender()` in the `withRegistry` modifier.

[Go back to Findings Summary]

M1: Missing event and `onInstall` call in `_initModules`

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	Initializer.sol	Type:	Bad implementation

Description

Modules installed via the `_initModules` function do not invoke the module's `onInstall` function and do not emit an event. This issue regards all module types and their respective install functions:

- `_installValidator`
- `_installExecutor`
- `_installFallbackHandler`
- `_installHook`

Exploit scenario

Without the `onInstall` call, modules that rely on it for initialization steps may face functional issues, potentially rendering them inoperable. This call is not guaranteed in the current setup.

Recommendation

The `_install…` functions return `moduleInitData` bytes, intended for module initialization upon installation. Fix the code by passing this data to `ModuleInstallUtil.installModule`, which calls the module's `onInstall` function and emits the required event.

Fix 1.1

The finding was fixed by catching the `moduleInitData` bytes and passing them to `ModuleInstallUtil.installModule`.

[Go back to Findings Summary](#)

M2: `BatchedExecUtil._tryExecute` inverted `success`

Medium severity issue

Impact:	Low	Likelihood:	High
Target:	DCUtil.sol	Type:	Logic bug

Description

The function `BatchedExecUtil._tryExecute` returns `success = true` if the `call` opcode reverts (returns `0`) and `false` if it succeeds.

The inverted `success` result is not used in the current state and therefore does not cause unexpected behavior like reverts or wrong events emitting. However, it is a major bug in the logic and should be fixed.

Listing 3. Excerpt from [BatchedExecUtil]()

```
            success := iszero(call(gas(), target, value, result,
    callData.length, codesize(), 0x00))
```

Exploit scenario

1. The developer utilizes the `success` result in some conditional logic.
2. The inverted value will cause unexpected behavior.

Recommendation

Remove the `iszero` opcode from the `success` value assignment in the `BatchedExecUtil._tryExecute` function.

```
success := call(gas(), target, value, result, callData.length, codesize(), 0x00)
```

Fix 1.1

Fixed, the `iszero` opcode was removed.

Go back to Findings Summary

M3: `BatchedExecUtil.tryExecute` single return value

Medium severity issue

Impact:	Low	Likelihood:	High
Target:	DCUtil.sol	Type:	Bad implementation

Description

The function `BatchedExecUtil.tryExecute` returns only the result of the last execution in the batch. In combination with the previous issue, the actual result is even inverted.

Listing 4. Excerpt from BatchedExecUtil

```
    function tryExecute(Execution[] calldata executions) external returns (bool success) {
        uint256 length = executions.length;

        for (uint256 i; i < length; i++) {
            Execution calldata _exec = executions[i];
            (success,) = _tryExecute(_exec.target, _exec.value, _exec.callData);
        }
    }
```

Exploit scenario

1. Batch execution gets executed.
2. `executions[0]` returns `success = false`.
3. `executions[1]` returns `success = false`.
4. `executions[2]` returns `success = true`.

5. `BatchedExecUtil.tryExecute` returns `true` even though the first two executions failed.

Recommendation

Use `bool[]` as the return type for the correct and exact batch execution results.

```
function tryExecute(Execution[] calldata executions) external returns (bool[]
memory result) {
    uint256 length = executions.length;
    result = new bool[](length);

    for (uint256 i; i < length; i++) {
        Execution calldata _exec = executions[i];
        (bool success,) = _tryExecute(_exec.target, _exec.value,
_exec.callData);
        result[i] = success;
    }
}
```

Fix 1.1

Fixed, the return value was removed from the `tryExecute` function.

[Go back to Findings Summary]

M4: `ModuleManager._installHook` SIG hook overwriting

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	ModuleManager.sol	Type:	Bad implementation

Description

The `ModuleManager._installHook` function allows overwriting SIG hooks although it is unwanted behavior. The `currentHook` local variable is not assigned before the revert condition `currentHook != address(0)`; therefore, the function does not revert even if the SIG hook is already installed, resulting in the SIG hook being overwritten.

Listing 5. Excerpt from [ModuleManager]()

```
        } else if (hookType == HookType.SIG) {
            // Dont allow hooks to be overwritten. If a hook is currently installed, it must be
            // uninstalled first
            if (currentHook != address(0)) {
                revert HookAlreadyInstalled(currentHook);
            }
            currentHook = $hookManager[msg.sender][selector];
            $hookManager[msg.sender][selector] = hook;
```

Exploit scenario

There are two possible ways the user can experience this behavior.

Installation:

1. The user installs the SIG hook for the specific function signature.

2. The user installs a different SIG hook for the same function signature or multitype module that contains this kind of hook.
3. The user's original SIG hook is being overwritten without warning.

Initialization:

1. The user passes two SIG hooks for the same function signature to the `Initializer._initModules` function.
2. Hooks get installed in the loop.
3. The second SIG hook in the array overwrites the first one without warning.

Recommendation

Move the assignment of the `currentHook` local variable above the revert condition.

```
currentHook = $hookManager[msg.sender][selector];
if (currentHook != address(0)) {
 revert HookAlreadyInstalled(currentHook);
}
```

Fix 1.1

Fixed, the `currentHook` assignment was moved according to the recommendation.

[Go back to Findings Summary](#)

M5: Locked Ether

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	Safe7579.sol, Initializer.sol, ModuleManager.sol, Safe7579Launchpad.sol	Type:	Loss of funds

Description

The contract `Safe7579` and parent contracts `Initializer` and `ModuleManager` can receive Ether but never send it. According to developers, the `payable` modifiers on functions are used to be gas optimization.

List of `payable` functions: * `Safe7579` - `execute`, `executeFromExecutor`, `installModule`, `uninstallModule`, `validateUserOp` * `Initializer` - `launchpadValidators`, `initializeAccount` * `ModuleManager` - `fallback` * `Safe7579Launchpad` - `receive`

Exploit scenario

However it is unlikely, the following scenario is possible.

1. User calls `Safe7579` some `payable` function with `msg.value` using `Safe.execTransaction` or `Safe.execTransactionFromModule` by setting the address of `Safe7579` as the target.
2. User's funds get irreversibly locked in the contract.

Recommendation

Remove the `payable` modifier where is not necessary.

Decorate the `receive` function in the `Safe7579Launchpad` contract with the `onlyProxy` modifier to prevent locked Ether on the launchpad implementation contract.

Fix 1.1

Fixed. The `payable` modifier was removed from functions. However, the `ModuleManager.fallback` remains `payable`.

> Cheaper on gas. the `fallback` in safe accounts does not forward `msg.value` so nothing should be able to get stuck here.
>
> — Rhinestone

The `onlyProxy` modifier was added to the `Safe7579Launchpad.receive` function.

[Go back to Findings Summary]

L1: Fallback handler `CallType` validation

Low severity issue

Impact:	Low	Likelihood:	Medium
Target:	ModuleManager.sol	Type:	Data validation

Description

EIP-7579's fallback handlers that can be installed through the `Safe7579` module only support a subset of all possible call types (static call, single call, batch call, etc).

However, the `ModuleManager` contract responsible for registering new fallback handlers and for dispatching to fallback handlers does not perform any data validation for the supported call types when installing a new fallback handler.

Listing 6. Excerpt from ModuleManager._installFallbackHandler

```
        (bytes4 functionSig, CallType calltype, bytes memory initData) =
            abi.decode(params, (bytes4, CallType, bytes));

        // disallow calls to onInstall or onUninstall.
        // this could create a security issue
        if (
            functionSig == IModule.onInstall.selector || functionSig == IModule.onUninstall.selector
        ) revert InvalidFallbackHandler(functionSig);
        if (_isFallbackHandlerInstalled(functionSig)) revert FallbackInstalled(functionSig);

        FallbackHandler storage $fallbacks = $fallbackStorage[msg.sender][functionSig];
        $fallbacks.calltype = calltype;
```

When dispatching to a given fallback handler, no data validation is performed either, ignoring unsupported call types.

Listing 7. Excerpt from [ModuleManager._callFallbackHandler]()

```
        if (calltype == CALLTYPE_STATIC) {
            return _staticcallReturn({
                safe: ISafe(msg.sender),
                target: handler,
                callData: abi.encodePacked(callData, _msgSender()) // append ERC2771
            });
        }
        if (calltype == CALLTYPE_SINGLE) {
            return _execReturn({
                safe: ISafe(msg.sender),
                target: handler,
                value: 0,
                callData: abi.encodePacked(callData, _msgSender()) // append ERC2771
            });
        }
```

Ignoring unsupported call types may be confusing to users and lead to unexpected behavior.

Exploit scenario

A user accidentally installs a new fallback handler with the batch call type. When using the fallback handler, the user assumes that the execution is succeeds, but no operation is performed without the user being notified.

Recommendation

When installing a new fallback handler, check for the supported call types and revert the transaction if an unsupported call type is provided.

Fix 1.1

Fixed by adding the following `if` condition to the `_installFallbackHandler` function:

```
        // disallow unsupported calltypes
        if (calltype != CALLTYPE_SINGLE && calltype != CALLTYPE_STATIC) {
            revert InvalidCallType(calltype);
        }
```

Go back to Findings Summary

ackee | blockchain security

L2: Domain-specific message encoding missing with `signedMessages`

Low severity issue

Impact:	Medium	Likelihood:	Low
Target:	Safe7579.sol	Type:	Signature schemes

Description

The function `Safe7579.isValidSignature` is responsible for approving arbitrary operations using the EIP-1271 standard.

The implementation handles multiple different scenarios:

- a hash is already approved in the `signedMessages` mapping in Safe,
- a hash is signed with Safe owners,
- a hash needs to be verified using one of the installed validator modules.

However, the first scenario is implemented differently from the implementation in Safe's `CompatibilityFallbackHandler` contract.

In the `Safe7579` contract, the hash is directly checked against `signedMessages`.

Listing 8. Excerpt from Safe7579

```
    function isValidSignature(
        bytes32 hash,
        bytes calldata data
    )
        external
        view
        returns (bytes4 magicValue)
    {
        ISafe safe = ISafe(msg.sender);
```

```

        // check for safe's approved hashes
        if (data.length == 0 && safe.signedMessages(hash) != 0) {
            // return magic value
            return IERC1271.isValidSignature.selector;
        }
```

The implementation in the `CompatibilityFallbackHandler` contract first encodes the hash into bytes, prepares [EIP-712](EIP-712) `messageData`, and then checks the hash against the `signedMessages` mapping.

```
    function isValidSignature(bytes32 _dataHash, bytes calldata _signature)
external view returns (bytes4) {
        ISignatureValidator validator = ISignatureValidator(msg.sender);
        bytes4 value = validator.isValidSignature(abi.encode(_dataHash),
_signature);
        return (value == EIP1271_MAGIC_VALUE) ? UPDATED_MAGIC_VALUE : bytes4(0);
    }

    function isValidSignature(bytes memory _data, bytes memory _signature)
public view override returns (bytes4) {
        // Caller should be a Safe
        Safe safe = Safe(payable(msg.sender));
        bytes memory messageData = encodeMessageDataForSafe(safe, _data);
        bytes32 messageHash = keccak256(messageData);
        if (_signature.length == 0) {
            require(safe.signedMessages(messageHash) != 0, "Hash not approved");
        } else {
            safe.checkSignatures(messageHash, messageData, _signature);
        }
        return EIP1271_MAGIC_VALUE;
    }
```

The implementation in the `CompatibilityFallbackHandler` works in conjunction with the `SignMessageLib` helper contract:

```
    function signMessage(bytes calldata _data) external {
        bytes32 msgHash = getMessageHash(_data);
        signedMessages[msgHash] = 1;
        emit SignMsg(msgHash);
```

```
    }

    function getMessageHash(bytes memory message) public view returns (bytes32)
{
        bytes32 safeMessageHash = keccak256(abi.encode(SAFE_MSG_TYPEHASH,
keccak256(message)));
        return keccak256(abi.encodePacked(bytes1(0x19), bytes1(0x01),
Safe(payable(address(this))).domainSeparator(), safeMessageHash));
    }
```

The approach in `Safe7579` does not guarantee that `signedMessages` will be called with the same pre-image data that was signed by the Safe owners.

Exploit scenario

Safe owners sign a message X representing a funds transfer using the `SignMessageLib` contract, producing a hash H and storing it in the `signedMessages` mapping.

An external contract uses EIP-1271 to verify a signature on a contract before transferring funds. Since the external contract is not aware that the implementation of `isValidSignature` in `Safe7579` does not encode the message, it will pass the message Y directly to the `isValidSignature` function. As a consequence:

- a call to the external contract with the message X (which was signed by the Safe owners) fails because X is not equal to H,
- a call to the external contract with the message H (which was not signed by the Safe owners) succeeds.

Recommendation

Use the same approach as in the `CompatibilityFallbackHandler` contract to achieve consistency and prevent attacks originating from the message encoding missing in the `Safe7579` contract.

Fix 1.1

The following code fixing the issue was added to the respective `if` block:

```
if (data.length == 0) {
    bytes32 messageHash = keccak256(
        EIP712.encodeMessageData(
            safe.domainSeparator(),
            SAFE_MSG_TYPEHASH,
            abi.encode(keccak256(abi.encode(hash)))
        )
    );

    require(safe.signedMessages(messageHash) != 0, "Hash not approved");
    // return magic value
    return IERC1271.isValidSignature.selector;
}
```

[Go back to Findings Summary]

L3: ERC-4337 factory standard violation

Low severity issue

Impact:	Low	Likelihood:	Medium
Target:	SafeProxyFactory.sol	Type:	ERC violation

Description

The flow when a new Safe with ERC-7579 enabled features is created through the `Safe7579Launchpad` contract expects Safe's `SafeProxyFactory` to be used as the ERC-4337 factory.

ERC-4337 defines the following assumption on the behavior of the factory:

> If the factory does use CREATE2 or some other deterministic method to create the wallet, it's expected to return the wallet address even if the wallet has already been created. This is to make it easier for clients to query the address without knowing if the wallet has already been deployed, by simulating a call to `entryPoint.getSenderAddress()`, which calls the factory under the hood.

However, the function `SafeProxyFactory.createProxyWithNonce` used to create the smart is implemented as follows:

```
    function createProxyWithNonce(address _singleton, bytes memory initializer,
uint256 saltNonce) public returns (SafeProxy proxy) {
        // If the initializer changes the proxy address should change too.
Hashing the initializer data is cheaper than just concatinating it
        bytes32 salt = keccak256(abi.encodePacked(keccak256(initializer),
saltNonce));
        proxy = deployProxy(_singleton, initializer, salt);
        emit ProxyCreation(proxy, _singleton);
```

```
    }

    function deployProxy(address _singleton, bytes memory initializer, bytes32
salt) internal returns (SafeProxy proxy) {
        require(isContract(_singleton), "Singleton contract not deployed");

        bytes memory deploymentData =
abi.encodePacked(type(SafeProxy).creationCode, uint256(uint160(_singleton)));
        assembly {
            proxy := create2(0x0, add(0x20, deploymentData),
mload(deploymentData), salt)
        }
        require(address(proxy) != address(0), "Create2 call failed");

        // rest of the implementation omitted
    }
```

This violates the standard because the function `createProxyWithNonce` reverts if the proxy has already been created at the given address.

Exploit scenario

A bundler implementation assumes that the factory behaves according to the [ERC-4337](ERC-4337) standard. Due to misbehavior, the bundler will refuse to process user operations working with the factory.

Recommendation

Create a wrapper that calls the `createProxyWithNonce` function and catches the revert, returning the proxy address if the proxy has already been created.

Solution 1.1

The Rhinestone team acknowledged the finding. Safe currently uses the same approach, and bundlers currently do not rely on the behavior described in the ERC-4337 standard. Rhinestone is aware it may be necessary to change the factory's behavior in the future to comply with the standard.

[Go back to Findings Summary]

L4: `_multiTypeInstall` module type validation

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	ModuleManager.sol	Type:	Data validation

Description

The function `ModuleManager._multiTypeInstall` is a helper to install an [ERC-7579](#) module as different module types at once while calling the function `onInstall` only once. The implementation decodes module types and the respective initialization data from a bytes payload and install the module with respect to the decoded types.

Listing 9. Excerpt from [ModuleManager._multiTypeInstall](#)

```
        for (uint256 i; i < length; i++) {
            uint256 _type = types[i];

            /*´:°•.°+.*•´.*:˚.°*.˚•´.°:°•.°•.*•´.*:˚.°*.˚•´.°:°•.°+.*•´.*:*/
            /*                      INSTALL VALIDATORS                    */
            /*.•°:°.´+˚.*°.˚:*.´•*.+°.•°:´*.´•*.•°.•°:°.´:•˚°.*°.˚:*.´+°.•*/
            if (_type == MODULE_TYPE_VALIDATOR) {
                _installValidator(module, contexts[i]);
            }
            /*´:°•.°+.*•´.*:˚.°*.˚•´.°:°•.°•.*•´.*:˚.°*.˚•´.°:°•.°+.*•´.*:*/
            /*                       INSTALL EXECUTORS                    */
            /*.•°:°.´+˚.*°.˚:*.´•*.+°.•°:´*.´•*.•°.•°:°.´:•˚°.*°.˚:*.´+°.•*/
            else if (_type == MODULE_TYPE_EXECUTOR) {
                _installExecutor(module, contexts[i]);
            }
            /*´:°•.°+.*•´.*:˚.°*.˚•´.°:°•.°•.*•´.*:˚.°*.˚•´.°:°•.°+.*•´.*:*/
            /*                       INSTALL FALLBACK                     */
            /*.•°:°.´+˚.*°.˚:*.´•*.+°.•°:´*.´•*.•°.•°:°.´:•˚°.*°.˚:*.´+°.•*/
            else if (_type == MODULE_TYPE_FALLBACK) {
                _installFallbackHandler(module, contexts[i]);
            }
            /*´:°•.°+.*•´.*:˚.°*.˚•´.°:°•.°•.*•´.*:˚.°*.˚•´.°:°•.°+.*•´.*:*/
```

```
            /*          INSTALL HOOK (global or sig specific)           */
            /*.•°:°.´+˚.*°.˚:*.´•*.+°.•°:´*.´•*.•°.•°:°.´:•˚°.*°.˚:*.´+°.•*/
            else if (_type == MODULE_TYPE_HOOK) {
                _installHook(module, contexts[i]);
            }
        }
```

However, there is no `else` branch handling the case when a decoded module type does not match one of the expected values.

Exploit scenario

A user accidentally supplies an unknown module type instead of the fallback module type. With no validation in place, the transaction is executed, but the fallback module is not installed. This can lead to unexpected behavior and potential security vulnerabilities because the user expects the fallback module to be installed.

Recommendation

Revert the execution if one of the decoded module types in `_multiTypeInstall` does not match one of the expected values.

Fix 1.1

Fixed by adding an `else` branch reverting the execution.

[Go back to Findings Summary]

W1: `postCheck` function is different from the EIP-7579 interface

Impact:	Warning	Likelihood:	N/A
Target:	IERC7579Module.sol	Type:	Bad implementation

Description

The `postCheck` function differs from the `IHook` interface specified in EIP-7579.

The EIP-7579's `IHook` interface specifies the function as follows.

```
interface IHook is IModule {
    function preCheck(
        address msgSender,
        uint256 value,
        bytes calldata msgData
    )
        external
        returns (bytes memory hookData);


    function postCheck(
        bytes calldata hookData,
        bool executionSuccess,
        bytes calldata executionReturn
    ) external;
}
```

In the codebase, the function is missing the second and third parameters.

Listing 10. Excerpt from IHook

```
interface IHook is IModule {
    function preCheck(
        address msgSender,
        uint256 msgValue,
```

```
        bytes calldata msgData
    )
        external
        returns (bytes memory hookData);

    function postCheck(bytes calldata hookData) external;
}
```

Recommendation

Ensure the `postCheck` function is consistent with the EIP-7579 specification. Unify the function signatures across both the interface and implementation to maintain compliance.

Fix 1.1

The EIP-7579 specification was fixed to match the implementation.

[Go back to Findings Summary]

W2: `uninstallModule` reverts on multi-type module

Impact:	Warning	Likelihood:	N/A
Target:	Safe7579.sol	Type:	User experience

Description

The `installModule` function supports installing multi-type modules, whereas the `uninstallModule` function does not and reverts when passed such a module (specifically, when `moduleType == MULTITYPE_MODULE`). Although such a module can still be uninstalled by calling the `uninstallModule` function separately for each type the module represents, this behavior could confuse users and module developers.

Listing 11. Excerpt from [Safe7579]()

```
    function uninstallModule(
        uint256 moduleType,
        address module,
        bytes calldata deInitData
    )
        external
        payable
        override
        withHook(IERC7579Account.uninstallModule.selector)
        onlyEntryPointOrSelf
    {
        // internal uninstall functions will decode the deInitData param, and return sanitzied
        // moduleDeInitData. This is the initData that will be passed to Module.onUninstall()
        bytes memory moduleDeInitData;
        if (moduleType == MODULE_TYPE_VALIDATOR) {
            moduleDeInitData = _uninstallValidator(module, deInitData);
        } else if (moduleType == MODULE_TYPE_EXECUTOR) {
            moduleDeInitData = _uninstallExecutor(module, deInitData);
        } else if (moduleType == MODULE_TYPE_FALLBACK) {
            moduleDeInitData = _uninstallFallbackHandler(module, deInitData);
```

```
        } else if (moduleType == MODULE_TYPE_HOOK) {
            moduleDeInitData = _uninstallHook(module, deInitData);
        } else {
            revert UnsupportedModuleType(moduleType);
        }
```

Furthermore, when calling `uninstallModule` function for each type individually, the module's `onUninstall` function will be triggered multiple times. Although this doesn't pose a vulnerability for the smart account since the module will still be removed even if an issue arises and the `onUninstall` function reverts, this unexpected behavior could create problems for module developers and lead to unintended consequences within the module's implementation.

Additionally, supporting multi-type modules in the `uninstallModule` function would make the uninstallation of such modules more gas-efficient.

Recommendation

Add support for multi-type modules in the `uninstallModule`.

Fix 1.1

The finding was fixed by creating a new `_multiTypeUninstall` function, which is used in the `uninstallModule` function to add support for multi-type modules.

[Go back to Findings Summary](#)

W3: Hooks can prevent module uninstallation

Impact:	Warning	Likelihood:	N/A
Target:	Safe7579.sol	Type:	Denial of service

Description

The `uninstallModule` function has the `withHook` modifier, which triggers a `preCheck` or `postCheck` call to the hook. A malicious hook could be installed to prevent modules (and the hook itself) from being uninstalled by reverting during these `preCheck` or `postCheck` calls when `uninstallModule` is invoked.

A simple exploit scenario involves packaging a malicious module with a hook that prevents the malicious module from being uninstalled as a multi-type module.

Currently, there is no recovery mechanism in place to address this issue.

Recommendation

Removing hooks from the `uninstallModule` function could hinder the smart account's ability to perform extended account management. As an alternative, additional checks could be implemented to prevent the hooks from impeding their uninstallation. Mainly, hooks could be designed not to execute if the target of the `uninstallModule` function is the hook itself. This adjustment would enable the removal of the hook to allow the malicious module to be uninstalled.

Fix 1.1

The issue was resolved by creating a specific `tryWithHook` modifier for the `uninstallModule` function. This modifier ensures that the hook's `preCheck` and `postCheck` functions execute, except when the target of the `uninstallModule` function is the hook itself.

[Go back to Findings Summary](#)

W4: Missing data validations

Impact:	Warning	Likelihood:	N/A
Target:	Safe7579Launchpad.sol	Type:	Data validation

Description

The `Safe7579Launchpad.setupSafe` function contains comments ”setupTo should be this launchpad” and ”setupData should be a call to this.initSafe7579()”, but the data validations are missing.

Listing 12. Excerpt from [Safe7579Launchpad]()

```
    // sload inithash from SafeProxy storage
    function _initHash() public view returns (bytes32 value) {
        // solhint-disable-next-line no-inline-assembly
        assembly ("memory-safe") {
            value := sload(INIT_HASH_SLOT)
        }
    }
```

Recommendation

Add proper data validations according to comments.

```
if (initData.setupTo != address(this)) revert InvalidSetupAddress();
if (bytes4(initData.setupData[:4]) != INIT_SAFE7579_SIGHASH) revert
InvalidSetupData();
```

Fix 1.1

Fixed, the following data validations were implemented.

```
if (initData.setupTo != SELF) revert InvalidSetup();
if (bytes4(initData.setupData[:4]) != this.initSafe7579.selector) revert
InvalidSetup();
```

Go back to Findings Summary

ackee | blockchain security

W5: Underscore prefixed public function

Impact:	Warning	Likelihood:	N/A
Target:	Safe7579Launchpad.sol	Type:	Best practices

Description

The `Safe7579Launchpad._initHash` function name contains the underscore prefix typical for `private`/`internal` functions, but the actual function visibility is `public`.

Listing 13. Excerpt from [Safe7579Launchpad]

```
    // sload inithash from SafeProxy storage
    function _initHash() public view returns (bytes32 value) {
        // solhint-disable-next-line no-inline-assembly
        assembly ("memory-safe") {
            value := sload(INIT_HASH_SLOT)
        }
    }
```

Recommendation

Change the function visibility to `internal`.

Fix 1.1

Fixed by changing the function name to `getInitHash`.

[Go back to Findings Summary]

W6: Hardcoded Enum.Operation values

Impact:	Warning	Likelihood:	N/A
Target:	ExecutionHelper.sol	Type:	Best practices

Description

The `ExecutionHelper` contract uses hardcoded `enum` values in `execTransactionFromModule` calls, which can be confusing and lead to overlooked errors during development.

Listing 14. Excerpt from [Safe7579Launchpad]()

```
        bool success = safe.execTransactionFromModule(target, value,
    callData, 0);
```

Listing 15. Excerpt from [Safe7579Launchpad]()

```
        bool success = safe.execTransactionFromModule(target, 0, callData,
    1);
```

Listing 16. Excerpt from [Safe7579Launchpad]()

```
        (success, retData) = safe.execTransactionFromModuleReturnData(target,
    value, callData, 0);
```

Listing 17. Excerpt from [Safe7579Launchpad]()

```
        (success, retData) = safe.execTransactionFromModuleReturnData(target,
    0, callData, 1);
```

Listing 18. Excerpt from [Safe7579Launchpad]()

```
        bool success = safe.execTransactionFromModule(target, value,
    callData, 0);
```

Listing 19. Excerpt from [Safe7579Launchpad]

```
        bool success = safe.execTransactionFromModule(target, 0, callData,
    1);
```

Listing 20. Excerpt from [Safe7579Launchpad]

```
        (success, retData) =
    safe.execTransactionFromModuleReturnData(target, value, callData, 0);
```

Listing 21. Excerpt from [Safe7579Launchpad]

```
        (success, retData) =
    safe.execTransactionFromModuleReturnData(target, 0, callData, 1);
```

Recommendation

Use proper `Enum.Operation.Call` and `Enum.Operation.DelegateCall` for better readability and maintenance.

Fix 1.1

Fixed, the `ISafe.Operation` enum was introduced and used.

[Go back to Findings Summary]

W7: Incomplete unused Safe7579UserOperationBuilder

Impact:	Warning	Likelihood:	N/A
Target:	Safe7579UserOperationBuilder.sol	Type:	Code quality

Description

The `Safe7579UserOperationBuilder` contract contains TODOs and commented-out code. However, this contract is not used by other contracts.

Listing 22. Excerpt from [Safe7579UserOperationBuilder]()

```
    // TODO: change it to address[] and bytes[] to be able to
    // stack policies for a permission
    // as of now it is enough to have a single policy for demo purposes
```

Listing 23. Excerpt from [Safe7579UserOperationBuilder]()

```
        // TODO: add delegatecall, tryExecute and other execution modes
handling
```

Listing 24. Excerpt from [Safe7579UserOperationBuilder]()

```
            /* commented this out bc currently deployed permission validator
is hardcode to
```

Also, the `getCallData` and `getDummySignature` functions visibility can be restricted to `pure` in the current state.

Recommendation

Remove the unused code and finish the incomplete code. Restrict the visibility of the `getCallData` and `getDummySignature` functions to `pure`.

Fix 1.1

Not fixed, mo changes were made to the `Safe7579UserOperationBuilder` contract.

[Go back to Findings Summary]

W8: Missing `TryExecutionFailed` emits

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

Description

The event `TryExecutionFailed` is emitted when a try execution fails in the `ExecutionHelper` helper contract.

Listing 25. Excerpt from [ExecutionHelper]()

```
    event TryExecutionFailed(ISafe safe, uint256 numberInBatch);
```

However, when calling a batch of executions, the event is only emitted when the whole batch fails due to the implementation, when a batch is internally executed through a single delegatecall.

Listing 26. Excerpt from [ExecutionHelper]()

```
    function _tryExec(ISafe safe, Execution[] calldata executions) internal {
        _tryDelegatecall({
            safe: safe,
            target: UTIL,
            callData: abi.encodeCall(BatchedExecUtil.tryExecute, executions)
        });
    }
```

Listing 27. Excerpt from [ExecutionHelper]()

```
    function _tryDelegatecall(ISafe safe, address target, bytes memory callData) internal {
        bool success = safe.execTransactionFromModule(target, 0, callData, 1);
        if (!success) emit TryExecutionFailed(safe, 0);
    }
```

As a consequence, the `numberInBatch` parameter is always `0` when the event is emitted, which can be misleading.

Recommendation

Either change the implementation to allow logging the event with the correct `numberInBatch` or remove the `numberInBatch` parameter from the event to avoid confusion and save gas.

Fix 1.1

Fixed by emitting the correct event in `function _tryExecReturn(ISafe safe, Execution[] calldata executions)` while keeping the implementation for `function _tryExec(ISafe safe, Execution[] calldata executions)`.

[Go back to Findings Summary]

I1: Duplicated code

Impact:	Info	Likelihood:	N/A
Target:	Safe7579Launchpad.sol	Type:	Code quality

Description

When installing a module, a call to `ModuleInstallUtil.installModule` is used to invoke the module's `onInstall` function and emits the required event.

Listing 28. Excerpt from [ModuleInstallUtil]()

```
    function installModule(
        uint256 moduleTypeId,
        address module,
        bytes calldata initData
    )
        external
    {
        IERC7579Module(module).onInstall(initData);
        emit ModuleInstalled(moduleTypeId, address(module));
    }
```

`Safe7579Launchpad.validateUserOp` does not reuse `ModuleInstallUtil.installModule` when setting up validators and duplicates its code.

Listing 29. Excerpt from [Safe7579Launchpad]()

```
        uint256 validatorsLength = initData.validators.length;
        for (uint256 i; i < validatorsLength; i++) {
            address validatorModule = initData.validators[i].module;
IValidator(validatorModule).onInstall(initData.validators[i].initData);
            emit ModuleInstalled(1, validatorModule);

            if (validatorModule == validator) userOpValidatorInstalled =
true;
```

```
198         }
```

Recommendation

Replace the duplicated code with a call to `ModuleInstallUtil.installModule`.

Fix 1.1

Acknowledged.

> Would require more gas. It's worth it to keep the gas minimal.
>
> — Rhinestone

[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.

Interfaces - `IValidator`, `IExecutor`, `IFallback`.

ISafe7579 - errors `LinkedListError`, `InitializerError`, `ValidatorStorageHelperError`, `HookPostCheckFailed`, - unused `using ModeLib for ModeCode`.

IERC7484 - event `NewTrustedAttesters` (not even part of the [ERC-7484](ERC-7484)).

IModule - errors `AlreadyInitialized`, `NotInitialized`.

IValidator - error `InvalidTargetAddress`.

IERC7579Account - event `AccountInitializationFailed`.

IERC7579AccountEvents - events `ModuleInstalled`, `ModuleUninstalled`.

ExecutionHelper - event `TryExecutionsFailed`.

Safe7579UserOperationBuilder - The contract is not used, - `getCallData` function contains unused parameters `smartAccount` and `context`, - `getDummySignature` function contains unused parameters `smartAccount` and `executions`.

Safe7579Launchpad - constants `DOMAIN_SEPARATOR_TYPEHASH`, `SAFE_INIT_TYPEHASH`, - function `_domainSeparator`.

Recommendation

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

Fix 1.1

Fixed.

> Fixed all but the try exec.
>
> — Rhinestone

[Go back to Findings Summary]

I3: Typos and incorrect documentation

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

Description

There are several typos and documentation issues across the project.

- The `Safe7579.sol` file contains comments that are incorrectly switched on L95 and L100, as well as L197 and L202.
- The `ModuleManager.getValidatorPaginated` function name should be `getValidatorsPaginated`. Also, the function uses inconsistent parameter naming - `cursor` instead of `start` like in other functions and `SentinelList`.
- False comment in `ModuleManager` L489 - `bytes[] moduleInitData` should be `bytes moduleInitData`.
- `Safe7579.sol` L177 - "need need" → "need".
- `Safe7579.sol` L383 - "sanitzied" → "sanitized".
- `Safe7579.sol` L424 - "sanitzied" → "sanitized".
- `Initializer.sol` L58 - "deplomet" → "deployment".

Recommendation

Fix the typos and documentation to improve code quality.

Fix 1.1

Fixed.

Go back to Findings Summary

I4: Code structure

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

Description

Summary of findings regarding the code structure.

- The `Safe7579.sol` file contains also the `EIP712` library. Move the library to a separate file into the `lib` directory.
- The `Safe7579DCUtil.sol` file contains `Safe7579DCUtilSetup` contract. Unify the naming.
- In the `Safe7579Launchpad` contract, modifiers are placed below the constructor. Move modifiers above the constructor.
- The `ISafe7579` interface is placed in the root. Move the file into the `interfaces` directory.

Recommendation

Fix these minor issues to improve code quality and readability.

Fix 1.1

Acknowledged.

> Actually prefer the readability as it is now.
>
> — Rhinestone

[Go back to Findings Summary]

6. Report revision 1.1

6.1. System Overview

There are not any important changes in the codebase.

7. Report revision 2.0

7.1. System Overview

A few significant changes were made to the codebase.

The `isModuleInstalled` function in the `Safe7579` contract now returns `true` for the address of `msg.sender` (the address of Safe under normal circumstances) and the validator module type. This is motivated by the fact that the Safe singleton itself may validate user operations.

A new helper function `addSafe7579` was introduced to the `Safe7579Launchpad` contract, allowing migration of an existing Safe account to Safe7579 in a single transaction.

The initial user operation performed on the `Safe7579Launchpad` contract can now be validated by the Safe owners, not only by validator modules.

The storage variables in the `Safe7579` contract were reworked to be fully compatible with ERC-4337 storage restrictions in all cases.

The `EncodedSafeOpStruct` struct used to compute the user operation hash signed by Safe owners was updated to the latest version.

Minor refactoring and compatibility improvements were made to the `Safe7579` and `Safe7579Launchpad` contracts.

W9: Safe does not implement validator interface

Impact:	Warning	Likelihood:	N/A
Target:	Safe7579.sol	Type:	EIP violation

Description

The function `isModuleInstalled` in the contract `Safe7579` returns a boolean flag of whether a given module is installed.

The following condition also returns `true` if the module address is `msg.sender`:

Listing 30. Excerpt from [Safe7579.isModuleInstalled]()

```
        if (moduleType == MODULE_TYPE_VALIDATOR) {
            // Safe7579 adapter allows for validator fallback to Safe's
checkSignatures().
            // It can thus be considered a valid validtor module
            if (module == msg.sender) return true;
            return _isValidatorInstalled(module);
```

[ERC-7579]() states that a validator module should implement the following functions:

```
function validateUserOp(PackedUserOperation calldata userOp, bytes32 userOpHash)
external returns (uint256);

function isValidSignatureWithSender(address sender, bytes32 hash, bytes calldata
signature) external view returns (bytes4);
```

In `isModuleInstalled`, `msg.sender` should always be a Safe account. However, Safe does not implement the validator interface, even through the fallback mechanism.

Recommendation

Reconsider the new if condition, as it does not comply with the standard and may lead to confusion.

[Go back to Findings Summary](#)

W10: Inconsistent signature checking

Impact:	Warning	Likelihood:	N/A
Target:	Safe7579Launchpad.sol	Type:	Data validation

Description

The function `_isValidSafeSigners` in the `Safe7579Launchpad` contract is used to verify the signatures of Safe owners before the singleton address in the `SafeProxy` contract is changed to the Safe singleton. So, `Safe7579Launchpad` users may expect the signature validation to behave like in the Safe singleton.

However, there is a difference in checking the `s` signature part in the case when the signer is an [EIP-1271](EIP-1271) contract.

The check in the function `Safe.checkNSignatures`

```
// Check that signature data pointer (s) is not pointing inside the static part
of the signatures bytes
// This check is not completely accurate, since it is possible that more
signatures than the threshold are send.
// Here we only check that the pointer is not pointing inside the part that is
being processed
require(uint256(s) >= requiredSignatures.mul(65), "GS021");
```

is different from the check in `CheckSignatures.recoverNSignatures`

```
// Check that signature data pointer (s) is not pointing inside the static part
of
// the signatures bytes
// Here we check that the pointer is not pointing inside the part that is being
// processed
if (uint256(s) < 65) {
    revert WrongContractSignatureFormat(uint256(s), 0, 0);
}
```

used by the _isValidSafeSigners function

Listing 31. Excerpt from [Safe7579Launchpad._isValidSafeSigners]

```
        address[] memory signers = _hash.recoverNSignatures(signatures,
safeSetupCallData.threshold);
```

The condition serves as a safety check to ensure that the EIP-1271 data pointer is not pointing inside the first part of the signature data. The difference in the checks may lead to different behavior in the signature validation process, confusing users of the Safe7579Launchpad contract.

Recommendation

Ensure that the signature checking in Safe7579Launchpad is consistent with the signature checking in the Safe singleton.

[Go back to Findings Summary]

I5: Unused using-for

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

Description

The codebase contains multiple unused using-for directives. See Appendix C for the complete list of all occurrences.

Recommendation

Remove the unused using-for directives to improve code maintainability and readability.

Go back to Findings Summary

I6: Typo

Impact:	Info	Likelihood:	N/A
Target:	Safe7579.sol	Type:	Code quality

Description

There is a typo in the `Safe7579.isModuleInstalled` function:

Listing 32. Excerpt from [Safe7579.isModuleInstalled]

```
473                 // It can thus be considered a valid validtor module
```

Recommendation

Fix the typo.

[Go back to Findings Summary]

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Rhinestone: Safe7579, 5.7.2024.

ackee | blockchain security

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 contains the outputs from the Wake development and testing framework used during the audit.

C.1. C1 proof of concept script

```
launchpad = Safe7579Launchpad.deploy(ENTRY_POINT, IERC7484(registry))
safe_factory = SafeProxyFactory("0x4e1DCf7AD4e460CfD30791CCC4F9c8a4f820ec67")
# init_data - crafted by a legitimate user

factory_call = abi.encode_call(
    launchpad.preValidationSetup,
    [launchpad.hash(init_data), Address.ZERO, b""],
)
salt_nonce = random_bytes(32)
init_code = bytes(safe_factory.address) + abi.encode_call(
    safe_factory.createProxyWithNonce,
    [launchpad, factory_call, int.from_bytes(salt_nonce, "big")],
)
predicted = launchpad.predictSafeAddress(
    launchpad,
    safe_factory,
    safe_factory.proxyCreationCode(),
    salt_nonce,
    factory_call,
)

# attack starts here
attacker = random_account()
# deploy SafeProxy with attacker, using the "official" SenderCreator used by
EntryPoint
creator = SenderCreator("0xefc2c1444ebcc4db75e7613d20c6a62ff67a167c")
tx = creator.createSender(init_code, from_=attacker)
assert tx.return_value == predicted
# change the singleton to one under attacker's control
tx = Safe7579Launchpad(tx.return_value).preValidationSetup(
    bytes32(0),
    malicious_contract,
    abi.encode_call(MaliciousContract.changeSingleton, []),
)
```

C.2. I5 detections

wake detect unused-using-for

```
[WARNING][LOW] Unused contract in using-for directive [unused-using-for]
 * "executeTransactionFromModule" features.
 */
contract Safe7579 is ISafe7579, SafeOp, SupportViewer, AccessControl, I
    using UserOperationLib for PackedUserOperation;
    using ExecutionLib for bytes;


src/Safe7579.sol
```

```
[WARNING][LOW] Unused contract in using-for directive [unused-using-for]
    SupportViewer,
    IERC7579AccountEvents
{
    using UserOperationLib for PackedUserOperation;
    using LibSort for address[];
    using CheckSignatures for bytes32;

src/Safe7579Launchpad.sol
```

```
[WARNING][LOW] Unused contract in using-for directive [unused-using-for]
 */
abstract contract Initializer is ISafe7579, ModuleManager {
    using SentinelList4337Lib for SentinelList4337Lib.SentinelList;
    using SentinelListLib for SentinelListLib.SentinelList;

    event Safe7579Initialized(address indexed safe);

src/core/Initializer.sol
```

```
[WARNING][LOW] Unused contract in using-for directive [unused-using-for]
 * respective section
 */
abstract contract ModuleManager is ISafe7579, AccessControl, Receiver,
    using SentinelListLib for SentinelListLib.SentinelList;
    using SentinelList4337Lib for SentinelList4337Lib.SentinelList;


src/core/ModuleManager.sol
```

Figure 2. Unused using-for directives

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://twitter.com/AckeeBlockchain