OpenZeppelin | security

Safe 4337 Module Audit

February 7, 2024

Table of Contents

Summary

Type	Account Abstraction
Timeline	From 2024-01-16 To 2024-01-23
Languages	Solidity

Total Issues	10 (8 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	3 (3 resolved)
Notes & Additional Information	6 (4 resolved)

Scope

We audited the safe-global/safe-modules repository at commit 3853f34.

In scope were the following files:

```
modules
└── 4337
    └── contracts
        ├── AddModulesLib.sol
        └── Safe4337Module.sol
```

System Overview

The `Safe4337Module` contract serves as an important extension to the existing Safe contracts. Specifically, it introduces compatibility with the ERC-4337 standard for account abstraction.

Key Features of `Safe4337Module`

1. **ERC-4337 Compliance**: The module ensures that Safe contracts can interact with the EVM chains following the ERC-4337 account abstraction model. This includes validating user operations and interacting with the entry point as defined in the ERC-4337 specification.

2. **Seamless Integration with Safe**: The module is designed to integrate smoothly with the existing Safe contracts without disrupting their core functionalities. It extends the capabilities of these contracts to support new transaction types and user operation models as introduced in ERC-4337.

3. **Enhanced User Experience**: By adopting the ERC-4337 standard, Safe users benefit from an improved transaction experience. This includes features like sponsored transactions, where transaction fees can be paid in tokens other than ETH, and more flexible transaction structures.

`AddModulesLib`

The primary function of this utility contract is to facilitate the deployment of safes with integrated ERC-4337 support. When the `EntryPoint` contract interacts with an account that has not yet been deployed, it is necessary to provide the required initialization data for its deployment. The `AddModulesLib` contract plays a vital role in this context. This is because a safe must execute a self-call to enable a module.

However, prior to executing the setup method, the safe's address remains unknown as it depends on the setup parameters. Therefore, methods like `MultiSend` are ineffective for establishing the correct `msg.sender` for a self-call. To address this, the initialization data includes a call to the `enableModules` function from the `AddModulesLib` contract. This is

executed as a post-deployment action through a delegate call, which activates the module in the same transaction that creates the safe.

Security Model

The security model of the `Safe4337Module` contract revolves around several key aspects:

1. **Account Abstraction Compliance**: Ensuring that the `EntryPoint` contract adheres to the specifications and security standards of ERC-4337. Not only is it dependent on the API of this contract, but the security of the safe module also depends on the entry point to perform certain validation checks such as nonce replay protection. One important thing to note is that the entry point address that the module points to is specified in the constructor and cannot be changed later on.
2. **Interaction with Safe Contracts**: Verifying the security and integrity of the interactions between the `Safe4337Module` and Safe contracts, especially in terms of permissions, transaction execution, and state changes.
3. **Module Isolation and Access Control**: Ensuring that the module operates within its designated boundaries, with strict access controls to prevent unauthorized interactions.

Privileged Roles

- **Safe Owner**: Each safe has an owner who has administrative privileges over it. This owner can enable or disable modules, guards, or the fallback. Furthermore, they can change important parameters within the safe which affects the validity of the associated signatures. If the signatures extracted from the user operation are invalid, user operations will not be successful.
- **Entry Point:** The only entity that can trigger both the validation and execution of user operations.

Medium Severity

M-01 Potential Obsolescence for `Safe4337Module`

The `Safe4337Module` contract is currently aligned with version 0.6.0 of the account abstraction repository. However, the imminent release of version 0.7.0 could render the module outdated upon market entry. This misalignment poses a risk of obsolescence and may impact the module's relevance and compatibility with newer systems.

To mitigate this, it is advisable to actively monitor updates in the account abstraction repository. In addition, establishing a communication channel with the repository developers could facilitate timely updates to the `Safe4337Module` contract. This proactive approach ensures synchronization with the latest developments, enhancing the module's longevity and utility in the evolving market.

Furthermore, all the current end-to-end tests performed in the repository use a bundler from version 0.6.1 along with the smart contract from version 0.6.0. Therefore, the passing of these tests does not guarantee compatibility and correctness when using version 0.7.0 of account abstraction, which may present breaking changes.

As such, Consider implementing end-to-end testing with the latest versions of the account abstraction components.

Update: *Resolved in pull request #245 at commit 2759608. The Safe team stated:*

> *Implemented by adding an E2E test against the bundler from the `main` branch. This should give a heads-up once a new version of the 4337 implementations is pushed out as the reference bundler includes and deploys the `EntryPoint` contract it supports, and the new CI test will fail if there are compatibility changes.*

Low Severity

L-01 Floating Pragma

A floating pragma (>=0.8.0 <0.9.0, which is equivalent to ^0.8.0) is being used for all the contracts within the scope of this audit. Floating pragmas are typically suitable for public contract libraries or contracts intended for inheritance by various projects. Its application in other contexts might inadvertently introduce risks associated with compiling contracts against different minor versions of the Solidity compiler, each potentially having distinct features, bug fixes, and deprecated elements.

To mitigate these risks, consider adopting fixed pragma versions specific to the Solidity compiler version that has been thoroughly tested and vetted for use with the ERC-4337 safe module contracts. By pinning down the Solidity version, the codebase will benefit from enhanced predictability and stability which is crucial for maintaining robust smart contracts.

Update: *Resolved in pull request #239 at commit df64c63.*

L-02 Misleading Documentation in `Safe4337Module` Contract

The `Safe4337Module` contract contains instances of misleading documentation that could potentially lead to confusion about the code's functionality and intent. Primarily, the immutable variable `SUPPORTED_ENTRYPOINT` is currently described inaccurately in its docstring as an EIP-712 type-hash, whereas it actually represents the address of the `EntryPoint` contract.

In addition, similar discrepancies are present elsewhere in the codebase. For instance, line 99 of `Safe4337Module` erroneously states that the entry point address is appended to the calldata within the `HandlerContext` contract. The correct implementation, however, is observed in the `FallbackManager` where this operation actually occurs.

Consider fixing the identified documentation issues and conducting a thorough review of the entire codebase to rectify any other documentation inaccuracies.

Update: *Resolved in pull request #240 at commit cd14438.*

L-03 Missing Docstrings

The `AddModulesLib.sol` contract lacks essential docstrings for the `enableModules` function on line 8.

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not `public`, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #241 at commit dd4a6b7.*

Notes & Additional Information

N-01 Inefficiency in `enableModules` Function Due to Reverse Loop Operation

The `AddModulesLib` contract's `enableModules` function implements a reverse loop for iterating over an array of module addresses. Using a reverse loop, especially with large arrays, could lead to higher gas costs as it involves more operations, as `i` is decremented in each iteration. The current implementation lacks clarity regarding the choice of reverse iteration as no justification is provided in the code or its documentation.

To enhance code clarity and efficiency, it is recommended to document the rationale behind opting for reverse iteration. If the choice of reverse iteration is arbitrary, refactoring the loop to a forward iteration is advisable to optimize gas usage, particularly in cases where numerous modules are going to be enabled. In this scenario, it would be advised to store `modules.length` in a temporary variable outside of the loop to prevent redundant calculation.

Update: *Resolved in pull request #241 at commit 9ea1a34.*

N-02 Misleading Contract Name

The `AddModuleLib` contract has a misleading name as it is a contract and not a library. Furthermore, the name gives a reader the impression that it is through this contract that all modules are enabled within the Safe system. However, this is not the typical flow as to how modules are generally enabled, but rather only for a specific scenario when the account that the `EntryPoint` should interact with is not deployed yet.

To provide clarity, consider renaming both the file and the contract to names that are more descriptive of the scenario in which this contract is used.

Update: *Resolved in pull request #241 at commit 942f2aa.*

N-03 Inefficient Modifier Usage Leading to Increased Bytecode Size

The `onlySupportedEntryPoint` modifier includes a `require` statement that verifies whether the value returned by `_msgSender()` matches `SUPPORTED_ENTRYPOINT`. This approach, while functionally correct, has an unintended consequence on the contract's bytecode size. During the compilation, modifiers are replicated wherever they are used. Given that `onlySupportedEntryPoint` is utilized in three distinct instances within the contract, this replication results in a noticeable increase in the overall bytecode size.

To optimize bytecode size and enhance contract efficiency, it is advisable to encapsulate the `require` statement in an internal function, and then invoke that function inside the modifier. This modification ensures that, instead of being duplicated, the logic for entry point validation is defined once and called multiple times.

Update: *Acknowledged, not resolved. The Safe team is expecting thousands of calls to this module which would actually make the suggestion cost more gas. The Safe team stated:*

> *Seeing as this module is to be shared across all Safes, we believe that an increase in the one-time deployment cost is the correct trade-off to allow for gas savings in the commonly used functions.*

N-04 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability

in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so.

In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for the maintainers of those libraries to contact the appropriate person about the problem and provide mitigation instructions. Throughout the codebase, there are contracts that do not have a security contact:

- The `AddModulesLib` contract
- The `Safe4337Module` contract

Consider adding a NatSpec comment containing a security contact above the contract definitions. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Resolved in pull request #244 at commit d6d057f and in pull request #241 at commit dd4a6b7.*

N-05 Use Custom Errors

Since Solidity version 0.8.4, custom errors provide a cleaner and more cost-efficient way to explain to users why an operation failed. Multiple instances of `require` statements were found within `Safe4337Module.sol`:

- The `require` statement with the message "Invalid entry point"
- The `require` statement with the message "Unsupported entry point"
- The `require` statement with the message "Invalid caller"
- The `require` statement with the message "Unsupported execution function id"
- The `require` statement with the message "Execution failed"

For conciseness and gas savings, consider replacing `require` statements with custom errors.

Update: *Acknowledged, not resolved. The Safe team stated:*

> *The suggestion to use custom error codes instead of revert messages was implemented for the Safe 4337 module contract. This had a positive impact on the code size of the contracts. However, the 4337 entry point contract v0.6.0 has special support for revert strings. As such, we do not believe this to be a positive change in the context that the 4337 module is designed to work for the v0.6.0 contracts. In particular, we believe that*

> *prioritizing debuggability is more important than contract code size improvements or potential gas savings in the error paths. With that in mind, we have decided to not include this change for the module supporting the v0.6.0 version of the entry point contract. We acknowledge the issue and will implement the change once the contracts are upgraded to support the v0.7.0 module which no longer has special support for handling revert strings.*

N-06 Inconsistent Use of Named Returns

The `Safe4337Module` contract exhibits inconsistent return styles: while most functions use named returns, the `domainSeparator()` function does not.

Consider using named returns for all functions in a contract for consistency.

Update: *Resolved in pull request #242 at commit b2ff724.*

Conclusion

This new module for the Safe contract acts as an extension that implements the ERC-4337 interface for an account and complies with the standard. It validates the user operation, executes a module transaction to pay the required prefund, and executes another module transaction to complete the user operation.

Only one medium severity issue was found during the audit and we found the code quality to be good. While the test suite present was robust with 100% coverage, end-to-end testing, and even formal verification through Certora, all tests were run with respect to account abstraction version 0.6.0. As such, we recommend extending this approach to test against the most recent version of account abstraction in order to remain compliant with any possible breaking changes. Throughout the audit period, the Safe team was helpful and responsive in answering our questions.