Safe

Social Recovery Module

by Ackee Blockchain

14.6.2024

ackee | blockchain security

Contents

1. Document Revisions

0.1	Draft report	13.6.2024
1.0	Final report	14.6.2024
1.1	Fix review	14.6.2024

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
Jan Kalivoda	Lead 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

Safe is a smart contract wallet and the Social Recovery Module is a subcomponent of the wallet that allows to recover access to the wallet when keys from the wallet are lost.

Revision 1.0

Safe engaged Ackee Blockchain to perform a security review of the Social Recovery Module initially implemented by Candide with a total time donation of 5 engineering days where 2 days from it were dedicated to fuzzing. In a period between June 6 and June 14, 2024, with Jan Kalivoda as the lead auditor.

The audit was performed on the commit `e6d45c8` [1] and the scope was the following:

- contracts/modules/social_recovery/SocialRecoveryModule.sol
- contracts/modules/social_recovery/storage/GuardianStorage.sol

We began our review using static analysis tools, including Wake. 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 the recovery mechanism can not be bypassed,
- ensuring the arithmetic of the system is correct,
- detecting possible reentrancies in the code,
- ensuring access controls are not too relaxed or too strict,
- looking for common issues such as data validation.

Our review resulted in 2 findings, ranging from Warning to Medium severity. The most severe issue is the possibility of a wallet recovery from other modules (see M1 issue). The codebase is overall of very high quality.

With fuzz tests, we created a differential model of the system in Python and defined several flows that executed all the functions and branches in the code. During the execution, we checked for specific assertions and between the flows we checked for the following invariants:

- the guardians in the contract state match the testing model,
- the owners in the contract state match the testing model,
- the threshold of the guardians in the contract state does not go over the number of guardians in the testing model,
- the threshold of the owners in the contract state does not go over the number of owners in the testing model.

Ackee Blockchain recommends Safe:

- address all reported issues.

See Revision 1.0 for the system overview of the codebase.

Revision 1.1

The review was done on the given commit: `113d3c0` [2] and the scope were the fixes for the previous revision.

All issues from the Revision 1.0 were fixed.

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

[1] full commit hash: `e6d45c8ca0c07f90204408f79684c7fed737944e`

[2] full commit hash: `113d3c059e039e332637e8f686d9cbd505f1e738`

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
M1: Other modules can be used to gain ownership of the wallet	Medium	1.0	Fixed
W1: Confirmed hashes stay in storage	Warning	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.

SocialRecoveryModule

Safe module that allows to choose Guardians (and their threshold) that can recover the wallet (for example when keys are lost).

GuardianStorage

The contract that holds the Guardians and their threshold for the SocialRecoveryModule.

Actors

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

Owner

Safe smart account owner.

Guardian

The address that can recover the wallet with other Guardians if they reach the threshold.

5.2. Trust Model

Owners are choosing their Guardians (and threshold) that can recover the wallet. This relationship is based on trust.

M1: Other modules can be used to gain ownership of the wallet

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	SocialRecoveryModule.sol, GuardianStorage.sol	Type:	Logic error

Description

The GuardianStorage is a separate contract that stores the guardian addresses. The SocialRecoveryModule contract is interacting with the storage with external calls. These calls are restricted with the `authorized` modifier to allow only calls for the wallet from the wallet.

Listing 1. SocialRecoveryModule.addGuardianWithThreshold

```
function addGuardianWithThreshold(address _wallet, address _guardian, uint256
_threshold) external authorized(_wallet) {
    guardianStorage.addGuardianWithThreshold(_wallet, _guardian, _threshold);
}
```

The GuardianStorage public entrypoint is then accessed from the module.

Listing 2. GuardianStorage.addGuardianWithThreshold

```
function addGuardianWithThreshold(address _wallet, address _guardian, uint256
_threshold) external onlyModule(_wallet) {
    ...
}
```

The call succeeds because of the `onlyModule` modifier.

Listing 3. GuardianStorage.onlyModule

```
modifier onlyModule(address _wallet) {
    // solhint-disable-next-line reason-string
    require(ISafe(payable(_wallet)).isModuleEnabled(msg.sender), "GS: method
only callable by an enabled module");
    _;
}
```

However, the `onlyModule` modifier does not check if the module is exactly the SocialRecoveryModule. As a result, any address that is registered in the Safe wallet as a module can be used to add, remove guardians or change the threshold. The address with its own set of guardians can recover the wallet and gain ownership.

Exploit scenario

Bob's Safe has enabled some module that is restricted to very specific operations (when calling `execTransactionFromModule`) but it is possible to call other contracts on behalf of the module. Alice can't use the module to exploit Bob, because the module's public entrypoints that are calling `execTransactionFromModule` on Bob's Safe are well protected. However, Alice notices that Bob has also enabled the SocialRecoveryModule as a module. Alice then uses the first module to change the GuardianStorage and then gains access to the wallet with the SocialRecoveryModule.

Recommendation

Restrict the usage of the GuardianStorage only to the SocialRecoveryModule.

Fix 1.1

The issue is fixed by inheriting directly from the GuardianStorage contract.

Go back to Findings Summary

W1: Confirmed hashes stay in storage

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

Description

When Guardian confirms a recovery, it means he will confirm a specific set of new owners and a threshold for a given wallet. Also, the nonce is included. However, nonce is not incremented until the recovery is executed. It means the nonce can be the same for years, but guardians and threshold can change over time.

The possible scenario is that Guardian A confirms a recovery at a time when the guardian threshold is set to 3. After that, he's revoked by Owner. One year passes and the Guardian A is added again, but this time threshold is 1. So immediately anyone can execute the recovery.

Recommendation

Be aware of this behavior, inform users about it or implement additional checks (eg. time-based) to prevent this type of misuse.

Fix 1.1

The issue is fixed by adding the possibility for Owner to increment the nonce and thus invalidate the current epoch without performing a recovery.

Go back to Findings Summary

6. Report revision 1.1

The Guardian Storage is now used as a parent contract for the Social Recovery Module. Additionally, it is possible to invalidate the nonce by Owner.

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Safe: Social Recovery Module, 14.6.2024.

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.

ackee blockchain security

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://twitter.com/AckeeBlockchain