ackee | blockchain security

ZK Email

Email Recovery

by Ackee Blockchain

5.8.2024

ackee | blockchain security

Contents

1. Document Revisions

1.0-draft	Report draft	15.7.2024
1.0	Final report	17.7.2024
1.1	Fix review	30.7.2024
1.2	Fix review	5.8.2024

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
Lukáš Rajnoha	Lead Auditor
Michal Převrátil	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

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

3. Executive Summary

Revision 1.0

ZK Email engaged Ackee Blockchain to perform a security review of the ZK Email protocol with a total time donation of 8 engineering days in a period between July 4 and July 12, 2024, with Lukáš Rajnoha as the lead auditor.

The audit was performed on the commit 4e70316 [1] and the scope was the following:

- ./EmailRecoveryManager.sol
- ./modules/EmailRecoveryModule.sol
- ./modules/UniversalEmailRecoveryModule.sol
- ./handlers/EmailRecoverySubjectHandler.sol
- ./libraries/EnumerableGuardianMap.sol
- ./libraries/GuardianUtils.sol
- ./handlers/SafeRecoverySubjectHandler.sol
- ./factories/EmailRecoveryFactory.sol
- ./factories/EmailRecoveryUniversalFactory.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 initialization and configuration processes of recovery modules,
- ensuring proper guardian state management,
- checking event emission consistency and completeness,

- checking gas optimization and efficiency in smart contract operations,
- interaction with the ERC-7579 standard,
- 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 26 findings, ranging from High to Info severity. The most severe one (H1) originates from the ability to initialize the system without guardians and a zero threshold, which can lead to an invalid configuration and inconsistent guardian state. Another high severity issue (H2) refers to premature update of the guardian configuration in the `addGuardian` function, which can lead to a situation where the `totalWeight` value (the sum of weights of guardians) does not accurately reflect the total weight from accepted guardians, potentially making recovery impossible. There are additionally 3 medium severity issues related mainly to the configuration of validators in the modules and support for custom templates. The code also contains multiple low severity issues with warnings/infos, which are mostly overlooked trivial mistakes.

Ackee Blockchain recommends ZK Email to:

- disallow initialization of the system without guardians and a zero threshold,
- ensure that the system accurately tracks the sum of weights from accepted guardians,
- optimize gas usage of the contracts,
- address all other reported issues.

See Report revision 1.0 for the system overview and trust model.

Revision 1.1

ZK Email engaged Ackee Blockchain to perform a fix review, which was done on July 31 on the given commit: `88371b8` [2]. From 26 findings, 23 issues were fixed, and two warnings and one informational issue were acknowledged. No additional changes were made to the codebase in scope outside of the fixes. The status of all reported issues has been updated and can be seen in the findings table.

After ongoing discussion with the client, an additional potential issue was identified (M5).

See Report revision 1.1 for the revision overview.

Revision 1.2

An incremental fix review was performed on the given commit: `5b26a9a` [3] to review an additional fix for the issue M5.

See Report revision 1.2 for the revision overview.

[1] full commit hash: 4e7031693d8e97cfcbc42b7d063a748a0a53b952

[2] full commit hash: 88371b81a3dd4347dac8f2a5690c1434e86ff55f

[3] full commit hash: 5b26a9ade08257ccfcba14fe675f5343e306aa57

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.

Critical	High	Medium	Low	Warning	Info	Total
0	2	5	3	7	10	27

Table 2. Findings Count by Severity

Finding title	Severity	Reported	Status
H1: Multiple vulnerabilities in recovery configuration process	High	1.0	Fixed
H2: Premature guardian configuration update in `addGuardian` function	High	1.0	Fixed

Finding title	Severity	Reported	Status
M1: `templateIdx` function parameter check is in incorrect place	Medium	1.0	Fixed
M2: Maximum guardians DoS	Medium	1.0	Fixed
M3: Selector collisions in `UniversalEmailRecoveryModule`	Medium	1.0	Fixed
M4: MAX + 1 validators may be configured in `UniversalEmailRecoveryModule`	Medium	1.0	Fixed
L1: Validators can be added/removed before module initialization in `UniversalEmailRecovery`	Low	1.0	Fixed
L2: `UniversalEmailRecovery` validators cannot be disallowed after being uninstalled	Low	1.0	Fixed
L3: `cancelRecovery` function does not revert when no recovery is in process	Low	1.0	Fixed
W1: `isInitialized` function returns false if initialized without guardians	Warning	1.0	Fixed
W2: Unused `bytes32` function parameter in `EmailRecoveryManager`	Warning	1.0	Acknowledged

Finding title	Severity	Reported	Status
W3: Unnecessary computation of `calldataHash` value in `validateRecoverySubject` function	Warning	1.0	Fixed
W4: Gas inefficiencies in `UniversalRecoveryModule`	Warning	1.0	Fixed
W5: Events missing parameters	Warning	1.0	Fixed
W6: Missing `AddedGuardian` event emission in `setupGuardians` function	Warning	1.0	Fixed
W7: ERC-4337 violation in `onInstall`	Warning	1.0	Acknowledged
I1: `getTrustedRecoveryManager` function returns public variable `emailRecoveryManager`	Info	1.0	Fixed
I2: Non-immutable state variables in `EmailRecoveryManager` contract	Info	1.0	Fixed
I3: Misleading naming	Info	1.0	Fixed
I4: Unchecked return values in `EnumerableGuardianMap` library	Info	1.0	Fixed

Finding title	Severity	Reported	Status
I5: Use `calldata` in favor of `memory` in function parameters	Info	1.0	Fixed
I6: Floating pragma	Info	1.0	Acknowledged
I7: Missing zero-address validation in constructors	Info	1.0	Fixed
I8: Modifiers not above constructors	Info	1.0	Fixed
I9: Safe `validateRecoverySubject` optimization	Info	1.0	Fixed
I10: Unused using-for directive	Info	1.0	Fixed
M5: `UniversalRecoveryModule` arbitrary Safe recovery call	Medium	1.1	Fixed

Table 3. Table of Findings

Report revision 1.0

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.

EmailRecoveryModule

The `EmailRecoveryModule` is the ERC-7579 module installed on the smart account, enabling account recovery. It is used with a specific validator, which must be installed on the account before the module can be initialized. The module enforces only a specific function selector (set during deployment) to be called on the validator during the recovery process.

It contains the necessary `onInstall` and `onUninstall` functions. Both functions delegate most of the recovery initialization and deinitialization functionality to the `EmailRecoveryManager` contract, which stores most of the recovery and configuration data.

The most important, `recover` function, is called from the `EmailRecoveryManager.completeRecovery` function after a recovery threshold is met to finalize the recovery request. After validity checks, the function calls the specific validator, passing the given calldata received in the function to finalize the recovery on the validator.

Helper functions such as `getTrustedRecoveryManager`, `isInitialized`, `isModuleType` and `isAuthorizedToRecover` are included together with metadata

getters - module name and version.

UniversalEmailRecoveryModule

The `UniversalEmailRecoveryModule` functions similarly to `EmailRecoveryModule` but offers broader functionality by allowing recovery via any validator. This module maintains a mapping of allowed validators for each smart account using the recovery module and the specific function selectors that will be called on these validators. Validator management is handled through the `allowValidatorRecovery` and `disallowValidatorRecovery` functions. Additional helper functions, `getAllowedValidators` and `getAllowedSelectors`, are also included.

EmailRecoveryManager

Both `EmailRecoveryModule` and `UniversalEmailRecoveryModule` use the `EmailRecoveryManager` contract. It works closely with its coupled module, being responsible for: * recovery initialization via the `configureRecovery` function, * configuration management via the `changeThreshold` and `updateRecoveryConfig` functions, * the recovery process itself, where the `processRecovery` function is called once per each guardian for a recovery request, with the `completeRecovery` function used for finalizing the recovery process once the threshold is met.

For each smart account with the recovery module installed, the manager stores recovery configurations and requests data, together with guardian storage and their configuration.

Helper functions such as recovery configuration, requests, and template getters are also present in the manager.

EnumerableGuardianMap

The account recovery module uses a custom `GuardianStorage` struct to

manage guardian state. The `EnumerableGuardianMap` library, based on Open Zeppelin's `EnumerableMap` library, maps guardian addresses to their respective data structs. The library includes helper functions for adding, removing, and updating guardians in the mapping and checking if a given guardian exists in the mapping.

GuardianUtils

The `GuardianUtils` library contains functions for guardian management, such as initializing, adding and removing, accepting, and updating guardians' status.

EmailRecoverySubjectHandler

`EmailRecoverySubjectHandler` defines and validates the subjects for recovery emails. It handles two subject types: acceptance and recovery. The acceptance subject is used when a guardian needs to accept becoming a guardian for an account, while the recovery subject is used when an account is being recovered. The handler includes functions to extract the account address from both subject types, with additional helper functions.

SafeRecoverySubjectHandler

Same as `EmailRecoverySubjectHandler`, but specifically made ERC-7579 compatible Safes.

EmailRecoveryFactory

Factory used to setting up a new `EmailRecoveryModule`, deploys the module together with its coupled `EmailRecoveryManager`, and initializes both contracts as needed.

EmailRecoveryUniversalFactory

Same as `EmailRecoveryFactory`, but specifically made for

`UniversalEmailRecoveryModule`.

Actors

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

Smart Account

The smart account is the account that is being secured by the account recovery module.

Email Recovery Module

The Email Recovery Module is an ERC-7579 compatible executor module, which is installed on the smart account, providing account recovery functionality.

Guardian

An entity that can help recover the account. The smart account owner adds guardians, who must accept the role before they can participate in the recovery process. Depending on the configuration of the recovery module, multiple guardians might be required for account recovery.

Validator

Validators are a specific type of ERC-7579 modules used during the validation phase to determine if a transaction is valid and should be executed on the account. Validators are called at the last stage in the recovery process to recover lost access to the account. The specific implementation of the validator recovery process can vary depending on the specific validator used.

Email Recovery Manager

The Email Recovery Manager contract works closely with its coupled Email Recovery Module, managing the recovery process. Most interactions during

the recovery process are done through the manager, which stores most of the recovery and configuration data.

Subject Handler

Subject handlers define the subjects for recovery emails and how they should be validated.

Factory

The Email Recovery Factories are helper contracts that deploy the recovery modules with their respective managers. There is a specific factory for each recovery module type.

Trust Model

Smart account owners have to trust the ERC-7579 validator modules installed on the account and the guardians they add to the recovery configuration, as they can start a new recovery process once accepted.

Smart account owners can trust the recovery module and the recovery manager to handle the recovery process correctly.

Other actors should have no control over the smart account.

H1: Multiple vulnerabilities in recovery configuration process

High severity issue

Impact:	High	Likelihood:	Medium
Target:	EmailRecoveryManager.sol	Type:	Logical error, Double initialization

Description

The recovery configuration process in the `EmailRecoveryManager` contract contains multiple vulnerabilities that can lead to inconsistent states. These issues stem from the ability to initialize the system without guardians and inconsistencies in how the initialization state is checked and maintained.

The complex vulnerability mainly stems from the following two factors:

1. Allowing initialization without guardians and a zero threshold

   The system allows initialization without guardians and with a zero threshold, which can lead to issues when guardians are added later without updating the threshold.

2. Insufficient initialization check

   The configureRecovery function checks for initialization by verifying that the threshold is zero instead of checking the `initialized` parameter in the `GuardianConfig` struct. This approach allows the function to be called multiple times if the system was initially configured without guardians (with a zero threshold).

Listing 1. Excerpt from EmailRecoveryManager

```
227 function configureRecovery(
228     address[] memory guardians,
229     uint256[] memory weights,
230     uint256 threshold,
231     uint256 delay,
232     uint256 expiry
233 )
234     external
235 {
236     address account = msg.sender;
237     // Threshold can only be 0 at initialization.
238     // Check ensures that setup function can only be called once.
239     if (guardianConfigs[account].threshold > 0) {
240         revert SetupAlreadyCalled();
241     }
```

These issues create two main vulnerabilities:

1. Initializing the module without guardians and a zero threshold does not require raising the threshold afterward when guardians are added. This leads to an invalid recovery configuration. The guardians can start a new recovery process through the `processRecovery` function; however, the recovery will fail to be completed due to the zero threshold check at the `completeRecovery` function. Considering the previous scenario — adding guardians without updating the threshold — the user can call `configureRecovery` again to set up a new guardian configuration, overriding the already set-up `GuardianConfigs`. This results in more guardians being stored in the recovery than accounted for.

Exploit scenario

1. The user initializes the module without guardians (with the threshold set to zero).
2. The user adds several guardians using the `addGuardian` function without

updating the threshold. The system now has more than one guardian but still has a zero threshold, thus being in an invalid configuration.

3. The user calls the `configureRecovery` function again, setting up new guardians. This overrides the `totalWeight` and `guardianCount` fields in the `GuardianConfigs` struct, ignoring previously added guardians. The system now has more guardians than accounted for.

Recommendation

To address these vulnerabilities, consider the following changes:

1. Disallow initialization without guardians.

   Modify the `configureRecovery` function to require at least one guardian to be set up and a non-zero threshold.

2. Use threshold to determine initialization status.

   Remove the `initialized` field in the `GuardianConfigs` struct and use the threshold to check if the system has been initialized.

Fix 1.1

The issue was fixed by disallowing the initialization of the system without guardians and a zero threshold. The initialization status of the system is now solely determined by the threshold.

Go back to Findings Summary

H2: Premature guardian configuration update in `addGuardian` function

High severity issue

Impact:	High	Likelihood:	Medium
Target:	GuardianUtils.sol	Type:	Logical error

Description

In the `GuardianUtils` library, the `addGuardian` function updates the `guardianCount` and `totalWeight` fields in the `GuardianConfigs` struct before the guardian is accepted. This premature update can lead to a situation where the `totalWeight` does not accurately reflect the sum of weights from accepted guardians.

Listing 2. Excerpt from GuardianUtils

```
147 function addGuardian(
148     mapping(address => EnumerableGuardianMap.AddressToGuardianMap) storage
    guardiansStorage,
149     mapping(address => IEmailRecoveryManager.GuardianConfig) storage
    guardianConfigs,
150     address account,
151     address guardian,
152     uint256 weight
153 )
154     internal
155 {
156     // Initialized can only be false at initialization.
157     // Check ensures that setup function should be called first
158     if (!guardianConfigs[account].initialized) {
159         revert SetupNotCalled();
160     }
161     if (guardian == address(0) || guardian == account) {
162         revert InvalidGuardianAddress();
163     }
164     GuardianStorage memory guardianStorage =
    guardiansStorage[account].get(guardian);
```

```
165     if (guardianStorage.status != GuardianStatus.NONE) {
166         revert AddressAlreadyGuardian();
167     }
168     if (weight == 0) {
169         revert InvalidGuardianWeight();
170     }
171     guardiansStorage[account].set({
172         key: guardian,
173         value: GuardianStorage(GuardianStatus.REQUESTED, weight)
174     });
175     guardianConfigs[account].guardianCount++;
176     guardianConfigs[account].totalWeight += weight;
177     emit AddedGuardian(account, guardian);
178 }
```

`totalWeight` should only account for the sum of weights from accepted guardians, which is not the case in the current implementation. This potential difference allows users to accidentally set up an invalid configuration, making recovery impossible in specific scenarios. Moreover, the recovery can be initiated despite the configuration being invalid.

Exploit scenario

Consider the following exploit:

1. The user initially starts with 2 guardians with `weight = 1` each, `threshold` set to 2 (`totalWeight = 2`, `threshold = 2`)
2. The users adds a third guardian with `weight = 1`, not yet accepted (`totalWeight = 3`, `threshold = 2`)
3. The user increases the threshold to 3 (`totalWeight = 3`, `threshold = 3`)

Although `totalWeight` is 3, the third guardian has not yet accepted, so the actual usable weight is 2. In such a case, recovery will be impossible until the third guardian accepts (which is not guaranteed).

Recommendation

To fix the issue, ensure that recovery can be initiated only when the sum of weight for accepted guardians reaches the threshold.

Fix 1.1

The issue was fixed by adding a dedicated `acceptedWeight` variable to track the sum of weights from accepted guardians. The `acceptedWeight` variable is used to determine if the recovery threshold can be met and if the recovery process can be initiated.

Go back to Findings Summary

M1: `templateIdx` function parameter check is in incorrect place

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	EmailRecoverySubjectHandler.sol, SafeRecoverySubjectHandler.sol	Type:	Code quality

Description

The `acceptGuardian` and `processRecovery` functions in the `EmailRecoveryManager` contract validate the `templateIdx` function parameter, reverting if it is non-zero. This validation seems to occur in the wrong place and should be moved to the `validateAcceptanceSubject` and `validateRecoverySubject` functions of `EmailRecoverySubjectHandler` and `SafeRecoverySubjectHandler` contracts. Additionally, this condition hinders the ability to use custom subject handlers with different templates.

Exploit scenario

Consider creating a new example subject handler using a different template (presumably using a non-zero `tempalteIdx` parameter) in the future. The new subject handler will not be usable since `acceptGuardian` and `processRecovery` functions will revert when `templateIdx != 0`.

Recommendation

Move `templateIdx` parameter validation to `EmailRecoverySubjectHandler` and `SafeRecoverySubjectHandler` contracts.

Fix 1.1

The issue was fixed by moving the `templateIdx` parameter validation to `EmailRecoverySubjectHandler` and `SafeRecoverySubjectHandler` contracts.

Go back to Findings Summary

M2: Maximum guardians DoS

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	EnumerableGuardianMap.sol	Type:	Denial of service

Description

The library `EnumerableGuardianMap` is a modified version of the `EnumerableMap` library from OpenZeppelin. It allows adding, updating, and removing guardians from a guardian map. The add and update operations are both implemented in a single `set` function.

Listing 3. Excerpt from EnumerableGuardianMap

```
62 function set(
63     AddressToGuardianMap storage map,
64     address key,
65     GuardianStorage memory value
66 )
67     internal
68     returns (bool)
69 {
70     uint256 length = map._keys.length();
71     if (length >= MAX_NUMBER_OF_GUARDIANS) {
72         revert MaxNumberOfGuardiansReached();
73     }
74     map._values[key] = value;
75     return map._keys.add(key);
76 }
```

Because of the `MAX_NUMBER_OF_GUARDIANS` check, the execution reverts when updating an already inserted guardian with the maximum number of guardians registered.

The function `set` is used in the update context in the

`GuardianUtils.updateGuardianStatus` function and, consequently, in the `EmailRecoveryManager.acceptGuardian` function.

Listing 4. Excerpt from EmailRecoveryManager.acceptGuardian

```
330 GuardianStorage memory guardianStorage = getGuardian(account, guardian);
331 if (guardianStorage.status != GuardianStatus.REQUESTED) {
332     revert InvalidGuardianStatus(guardianStorage.status,
    GuardianStatus.REQUESTED);
333 }
334 guardiansStorage.updateGuardianStatus(account, guardian,
    GuardianStatus.ACCEPTED);
```

As a result, a guardian cannot accept the invitation if the maximum number of guardians is registered.

Exploit scenario

A user registers the maximum number of guardians (32). Due to the incorrect implementation of the `set` function, the guardians cannot accept the invitation until one of the guardians is removed.

Recommendation

Use the return value of `map._keys.add(key)` indicating whether the key was not already present in the map. Perform the `MAX_NUMBER_OF_GUARDIANS` check only if the guardian was not already present in the map.

Fix 1.1

The issue was fixed by modifying the `EnumerableGuardianMap.set` function, which now checks the return value of `map._keys.add(key)` and uses the > inequation sign instead of >= in the `MAX_NUMBER_OF_GUARDIANS` check.

Listing 5. Excerpt from EnumerableGuardianMap

```
62 function set(
```

```
63      AddressToGuardianMap storage map,
64      address key,
65      GuardianStorage memory value
66  )
67      internal
68      returns (bool)
69  {
70      map._values[key] = value;
71      bool success = map._keys.add(key);
72      uint256 length = map._keys.length();
73      if (success && length > MAX_NUMBER_OF_GUARDIANS) {
74          revert MaxNumberOfGuardiansReached();
75      }
76      return success;
```

Go back to Findings Summary

M3: Selector collisions in `UniversalEmailRecoveryModule`

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	UniversalEmailRecoveryModule.sol	Type:	Data validation

Description

The contract `UniversalEmailRecoveryModule` is a generalized ERC-7579 executor module for recovery of smart accounts. It allows registering multiple validator modules that can be recovered. In order to select the correct validator to recover based on a function selector, the `selectorToValidator` mapping is used.

Listing 6. Excerpt from UniversalEmailRecoveryModule

```
66 mapping(bytes4 selector => mapping(address account => address validator))
   internal
67     selectorToValidator;
```

However, the `UniversalEmailRecoveryModule` contract does not handle cases where two or more validator modules are registered with the same function selector. In such cases, the `selectorToValidator` mapping will be overwritten, leading to a collision and the inability to recover the original validator module.

Exploit scenario

A user accidentally registers two validator modules, A and B (in this order), with the same function selector. The `selectorToValidator` mapping will contain only the last registered validator module, B, and the original validator module, A, cannot be recovered.

Recommendation

Either revert the execution when registering a validator module with a colliding function selector or implement a mechanism to handle collisions.

Fix 1.1

> The simplest solution to this was to remove the `selectorToValidator` mapping and just pass the validator in with the calldata to `recover`.
>
> — ZK Email Team

Fixed by removing the `selectorToValidator` mapping. In the `recover` function, the validator is now decoded from the calldata:

Listing 7. Excerpt from UniversalEmailRecoveryModule

```
278 function recover(address account, bytes calldata recoveryData) external {
279     if (msg.sender != emailRecoveryManager) {
280         revert NotTrustedRecoveryManager();
281     }
282     (address validator, bytes memory recoveryCalldata) =
283         abi.decode(recoveryData, (address, bytes));
284     bytes4 selector;
285     assembly {
286         selector := mload(add(recoveryCalldata, 32))
287     }
```

Go back to Findings Summary

M4: MAX + 1 validators may be configured in `UniversalEmailRecoveryModule`

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	UniversalEmailRecoveryModule.sol	Type:	Logical error

Description

The following if condition in the `UniversalEmailRecoveryModule` contract should ensure that no more than `MAX_VALIDATORS` validators are configured.

Listing 8. Excerpt from UniversalEmailRecoveryModule.allowValidatorRecovery

```
151 if (validatorCount[msg.sender] > MAX_VALIDATORS) {
152     revert MaxValidatorsReached();
153 }
154 validators[msg.sender].push(validator);
155 validatorCount[msg.sender]++;
```

However, due to the incorrect inequality operator, the condition allows configuring `MAX_VALIDATORS + 1` validators.

Exploit scenario

A `UniversalEmailRecoveryModule` user accidentally configures `MAX_VALIDATORS + 1` (33) validators. Because the function `getAllowedValidators` uses the `MAX_VALIDATORS` constant, metadata for the 33rd validator is not cleared in `onUninstall`.

Listing 9. Excerpt from UniversalEmailRecoveryModule.onUninstall

```
208 address[] memory allowedValidators = getAllowedValidators(msg.sender);
```

```
209 for (uint256 i; i < allowedValidators.length; i++) {
210     bytes4 allowedSelector =
    allowedSelectors[allowedValidators[i]][msg.sender];
211     delete selectorToValidator[allowedSelector][msg.sender];
212     delete allowedSelectors[allowedValidators[i]][msg.sender];
213 }
214 validators[msg.sender].popAll();
215 validatorCount[msg.sender] = 0;
```

When the `UniversalEmailRecoveryModule` is installed again, the validator is still considered valid, and due to the `validatorCount` counter being reset to zero, removing the validator is impossible.

Recommendation

Change the inequation sign from `>` to `>=` in the `allowValidatorRecovery` function to ensure that no more than `MAX_VALIDATORS` validators can be configured.

Fix 1.1

The issue was fixed by changing the inequation sign from `>` to `>=` in the `allowValidatorRecovery` function.

Listing 10. Excerpt from UniversalEmailRecoveryModule

```
151 function allowValidatorRecovery(
152     address validator,
153     bytes memory isInstalledContext,
154     bytes4 recoverySelector
155 )
156     public
157     onlyWhenInitialized
158     withoutUnsafeSelector(recoverySelector)
159 {
160     if (
161         !IERC7579Account(msg.sender).isModuleInstalled(
162             TYPE_VALIDATOR, validator, isInstalledContext
163         )
164     ) {
```

```
165         revert InvalidValidator(validator);
166     }
167     if (validatorCount[msg.sender] >= MAX_VALIDATORS) {
168         revert MaxValidatorsReached();
169     }
```

Go back to Findings Summary

L1: Validators can be added/removed before module initialization in `UniversalEmailRecovery`

Low severity issue

Impact:	Medium	Likelihood:	Low
Target:	UniversalEmailRecoveryModule.sol	Type:	Logical error

Description

The intended flow for initializing the `UniversalEmailRecoveryModule` is first installing the module, during which the `onInstall` function is called. This function initializes the validators linked list via the `validators[msg.sender].init()` function. Then, more validators can potentially be added with the `allowValidatorRecovery` function. However, `allowValidatorRecovery` does not check if the module has yet been installed on `msg.sender`. Linked lists used in the code should be initialized before use, which is not guaranteed here. Otherwise, the linked list is incorrectly set up. The same issue is present in the `disallowValidatorRecovery` function.

Example from `allowValidatorRecovery`:

Listing 11. Excerpt from UniversalEmailRecoveryModule

```
135 function allowValidatorRecovery(
136     address validator,
137     bytes memory isInstalledContext,
138     bytes4 recoverySelector
139 )
140     public
141     withoutUnsafeSelector(recoverySelector)
142 {
143     if (
144         !IERC7579Account(msg.sender).isModuleInstalled(
145             TYPE_VALIDATOR, validator, isInstalledContext
```

```
146         )
147     ) {
148         revert InvalidValidator(validator);
149     }
150     if (validatorCount[msg.sender] > MAX_VALIDATORS) {
151         revert MaxValidatorsReached();
152     }
153     validators[msg.sender].push(validator);
154     validatorCount[msg.sender]++;
155     allowedSelectors[validator][msg.sender] = recoverySelector;
156     selectorToValidator[recoverySelector][msg.sender] = validator;
157     emit NewValidatorRecovery({ validatorModule: validator,
    recoverySelector: recoverySelector });
158 }
```

Exploit scenario

The user calls the `allowValidatorRecovery` function before installing the module, which adds a new validator to the linked list. Since the linked list was not initialized, its current state is as follows:

```
SENTINEL -> new_validator
```

While the correct state (if initialized beforehand) should be:

```
SENTINEL -> new_validator -> SENTINEL
```

Recommendation

Ensure that adding and removing validators is only possible when the module is installed (thus, the linked list has been initialized). Consider adding a modifier to the `allowValidatorRecovery` and `disallowValidatorRecovery` functions, reverting if the module is not installed on `msg.sender`.

Fix 1.1

The issue was fixed by adding the `onlyWhenInitialized` modifier to the

`allowValidatorRecovery` and `disallowValidatorRecovery` functions. The modifier checks if the `validators` sentinel list has been initialized for the given account (on module initialization). If not, the function reverts.

Go back to Findings Summary

L2: `UniversalEmailRecovery` validators cannot be disallowed after being uninstalled

Low severity issue

Impact:	Low	Likelihood:	Medium
Target:	UniversalEmailRecoveryModule.sol	Type:	Logical error

Description

In the `UniversalEmailRecovery` module, to allow a validator, the validator first has to be installed on the account. Otherwise, the `allowValidatorRecovery` function in `UniversalEmailRecovery` module reverts with `InvalidValidator` error.

Listing 12. Excerpt from UniversalEmailRecoveryModule

```
135 function allowValidatorRecovery(
136     address validator,
137     bytes memory isInstalledContext,
138     bytes4 recoverySelector
139 )
140     public
141     withoutUnsafeSelector(recoverySelector)
142 {
143     if (
144         !IERC7579Account(msg.sender).isModuleInstalled(
145             TYPE_VALIDATOR, validator, isInstalledContext
146         )
147     ) {
148         revert InvalidValidator(validator);
149     }
```

This check is also present in the `disallowValidatorRecovery` function. Therefore, if an allowed validator gets uninstalled from the smart account, disallowing the validator will revert with `InvalidValidator`.

Listing 13. Excerpt from UniversalEmailRecoveryModule

```
171 function disallowValidatorRecovery(
172     address validator,
173     address prevValidator,
174     bytes memory isInstalledContext,
175     bytes4 recoverySelector
176 )
177     public
178 {
179     if (
180         !IERC7579Account(msg.sender).isModuleInstalled(
181             TYPE_VALIDATOR, validator, isInstalledContext
182         )
183     ) {
184         revert InvalidValidator(validator);
185     }
```

The user allows a validator in the module and then uninstalls this validator from the smart account. The user tries to disallow the validator afterward, but it will fail with `InvalidValidator`. The user then has to reinstall the validator to be able to disallow it in the module.

Recommendation

To address this issue, remove the check for the validator in the `disallowValidatorRecovery` function. This allows the user to remove the validator even if it was uninstalled from the account.

Fix 1.1

The issue was fixed by removing the check for the validator in the `disallowValidatorRecovery` function. This allows the user to disallow a validator even after it has been uninstalled from the smart account.

Go back to Findings Summary

L3: `cancelRecovery` function does not revert when no recovery is in process

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	EmailRecoveryManager.sol	Type:	Logical error

Description

The `cancelRecovery` function in `EmailRecoveryManager` contract does not revert when no recovery is being processed. Thus, the `RecoveryCancelled` event is emitted regardless of whether a recovery is in progress, which can cause issues with off-chain tracking of the recovery status.

Listing 14. Excerpt from EmailRecoveryManager

```
455 function cancelRecovery() external virtual {
456     delete recoveryRequests[msg.sender];
457     emit RecoveryCancelled(msg.sender);
458 }
```

Recommendation

Revert in the `cancelRecovery` function if no recovery is in process.

Fix 1.1

The issue was fixed by reverting in the `cancelRecovery` function if no recovery is in process.

Listing 15. Excerpt from EmailRecoveryManager

```
465 function cancelRecovery() external virtual {
466     if (recoveryRequests[msg.sender].currentWeight == 0) {
467         revert NoRecoveryInProcess();
```

```
468     }
469     delete recoveryRequests[msg.sender];
470     emit RecoveryCancelled(msg.sender);
471 }
```

Go back to Findings Summary

W1: `isInitialized` function returns false if initialized without guardians

Impact:	Warning	Likelihood:	N/A
Target:	EmailRecoveryModule.sol, UniversalEmailRecoveryModule.sol	Type:	Logical error

Description

The `isInitialized` function in both `EmailRecoveryModule` and `UniversalEmailRecoveryModule` contracts checks for initialization by verifying that the threshold is non-zero. The module can, however, be initialized without guardians and with a zero threshold. In such a case, the function incorrectly returns `false`.

Listing 16. Excerpt from EmailRecoveryModule

```
118 function isInitialized(address smartAccount) external view returns (bool) {
119     return
    IEmailRecoveryManager(emailRecoveryManager).getGuardianConfig(smartAccount).
    threshold
120         != 0;
121 }
```

Recommendation

This issue closely relates with H1. Fixing the related, higher-severity issue using the provided recommendations also addresses this finding.

Consider adding a function that indicates whether the module is in a state where recovery is possible. When the `isInitialized` function returns true, it might indicate that the module is ready for recovery, which might not necessarily be the case. It could happen that not enough guardians have

been accepted to reach the required threshold set during configuration, thus making recovery impossible.

Fix 1.1

The issue was fixed in conjunction with the fix for H1 by disallowing initialization without guardians and with a zero threshold. Additionally, a new `canStartRecoveryRequest` function was added to indicate whether the module is in a state where recovery is possible (i.e., enough guardians have been accepted to reach the required threshold).

Go back to Findings Summary

W2: Unused `bytes32` function parameter in `EmailRecoveryManager`

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

Description

In the `EmailRecoveryManager` contract, the functions `acceptGuardian` and `processRecovery` both have an unused function parameter of type `bytes32`. This parameter is declared without a name and never used within the function bodies. Unused parameters can lead to confusion and may unnecessarily increase gas costs.

Listing 17. Excerpt from EmailRecoveryManager

```
303 function acceptGuardian(
304     address guardian,
305     uint256 templateIdx,
306     bytes[] memory subjectParams,
307     bytes32
308 )
```

Listing 18. Excerpt from EmailRecoveryManager

```
352 function processRecovery(
353     address guardian,
354     uint256 templateIdx,
355     bytes[] memory subjectParams,
356     bytes32
```

Recommendation

Refactor the `acceptGuardian` and `processRecovery` functions to remove the unused `bytes32` parameter.

Acknowledgment 1.1

Documentation was updated, describing why the unused `bytes32` parameter is included in the given functions.

Go back to Findings Summary

W3: Unnecessary computation of `calldataHash` value in `validateRecoverySubject` function

Impact:	Warning	Likelihood:	N/A
Target:	EmailRecoveryManager.sol	Type:	Gas optimization

Description

The `processRecovery` function in the `EmailRecoveryManager` contract uses the subject handler's `validateRecoverySubject` function to validate the `subjectParams` function parameter and return the parsed `accountInEmail` and `calldataHash` values. However, the `calldataHash` value is stored only after the threshold in the `validateRecoverySubject` function is met; otherwise, the value is unused. If multiple guardians are needed for recovery, `calldataHash` is computed more than once and only used (stored) the last time. This results in unnecessary gas spending.

Listing 19. Excerpt from EmailRecoveryManager

```
352 function processRecovery(
353     address guardian,
354     uint256 templateIdx,
355     bytes[] memory subjectParams,
356     bytes32
357 )
358     internal
359     override
360 {
361     if (templateIdx != 0) {
362         revert InvalidTemplateIndex();
363     }
364     (address account, bytes32 calldataHash) =
    IEmailRecoverySubjectHandler(subjectHandler)
365         .validateRecoverySubject(templateIdx, subjectParams, address(this));
366     if
    (!IEmailRecoveryModule(emailRecoveryModule).isAuthorizedToRecover(account))
    {
367         revert RecoveryModuleNotAuthorized();
```

```
368     }
369     // This check ensures GuardianStatus is correct and also implicitly that
    the
370     // account in email is a valid account
371     GuardianStorage memory guardianStorage = getGuardian(account, guardian);
372     if (guardianStorage.status != GuardianStatus.ACCEPTED) {
373         revert InvalidGuardianStatus(guardianStorage.status,
    GuardianStatus.ACCEPTED);
374     }
375     RecoveryRequest storage recoveryRequest = recoveryRequests[account];
376     recoveryRequest.currentWeight += guardianStorage.weight;
377     uint256 threshold = guardianConfigs[account].threshold;
378     if (recoveryRequest.currentWeight >= threshold) {
379         uint256 executeAfter = block.timestamp +
    recoveryConfigs[account].delay;
380         uint256 executeBefore = block.timestamp +
    recoveryConfigs[account].expiry;
381         recoveryRequest.executeAfter = executeAfter;
382         recoveryRequest.executeBefore = executeBefore;
383         recoveryRequest.calldataHash = calldataHash;
384         emit RecoveryProcessed(account, executeAfter, executeBefore);
385     }
386 }
```

The gas required for computation varies depending on whether `EmailRecoverySubjectHandler` or `SafeRecoverySubjectHandler` is used as the subject handler. The gas spent in `SafeRecoverySubjectHandler.validateRecoverySubject` depends on the length of the owners of the Safe Smart Account.

Recommendation

To optimize the gas usage, consider splitting the `validateRecoverySubject` function into two functions:

1. `validateRecoverySubject` - validates the recovery subject and returns the `accountInEmail` value.
2. `parseRecoveryCalldataHash` - computes and returns the `calldataHash` value.

In `processRecovery` function, use `validateRecoverySubject` to get `accountInEmail` for validation purposes and only use the `parseRecoveryCalldataHash` function when the threshold is met, and `calldataHash` needs to be computed and stored.

Fix 1.1

Gas usage was optimized by splitting the `validateRecoverySubject` function into two separate functions: `validateRecoverySubject` (validates the recovery subject) and `parseRecoveryCalldataHash` (computes the `calldataHash`). The `calldataHash` value is now computed and stored only when the threshold is met.

Go back to Findings Summary

W4: Gas inefficiencies in `UniversalRecoveryModule`

Impact:	Warning	Likelihood:	N/A
Target:	UniversalEmailRecoveryModule.sol	Type:	Gas optimization

Description

The `UniversalRecoveryModule` contract is not gas-efficient. The main issues are:

- Unnecessary checks in the `recover` function.
- Inefficient implementations of `isAuthorizedToRecover` and `getAllowedSelectors` functions.

Specific issues include:

1. In the `recover` function:

Listing 20. Excerpt from UniversalEmailRecoveryModule

```
251 function recover(address account, bytes calldata recoveryCalldata)
    external {
252     if (msg.sender != emailRecoveryManager) {
253         revert NotTrustedRecoveryManager();
254     }
255     bytes4 selector = bytes4(recoveryCalldata[:4]);
256     address validator = selectorToValidator[selector][account];
257     bytes4 allowedSelector = allowedSelectors[validator][account];
258     if (allowedSelector != selector) {
259         revert InvalidSelector(selector);
260     }
261     _execute({ account: account, to: validator, value: 0, data:
    recoveryCalldata });
262     emit RecoveryExecuted();
263 }
```

The check against the `allowedSelector` variable is unnecessary and can be replaced with a simple non-zero address check for the `validator`.

2. In the `isAuthorizedToRecover` function:

Listing 21. Excerpt from UniversalEmailRecoveryModule

```
237 function isAuthorizedToRecover(address smartAccount) external view
    returns (bool) {
238     return getAllowedValidators(smartAccount).length > 0;
239 }
```

The function computes validator count through the `getAllowedValidators` function instead of using the `validatorCount` variable.

3. In the `getAllowedSelectors` function:

Listing 22. Excerpt from UniversalEmailRecoveryModule

```
294 function getAllowedSelectors(address account) external view returns
    (bytes4[] memory) {
295     address[] memory allowedValidators = getAllowedValidators(account);
296     uint256 allowedValidatorsLength = allowedValidators.length;
297     bytes4[] memory selectors = new bytes4[](allowedValidatorsLength);
298     for (uint256 i; i < allowedValidatorsLength; i++) {
299         selectors[i] = allowedSelectors[allowedValidators[i]][account];
300     }
301     return selectors;
302 }
```

The function computes validator count through the `getAllowedValidators` function instead of using the `validatorCount` variable.

Recommendation

Consider refactoring the `UniversalRecoveryModule` contract to address the gas inefficiencies.

Fix 1.1

The gas inefficiency in the `recover` fucntion was resolved in conjunction with M3. The `isAuthorizedToRecover` function was updated to use the `validatorCount` variable instead of computing the validator count through the `getAllowedValidators` function.

Go back to Findings Summary

W5: Events missing parameters

Impact:	Warning	Likelihood:	N/A
Target:	UniversalEmailRecoveryModule.sol, EmailRecoveryModule.sol, EmailRecoveryFactory.sol, EmailRecoveryUniversalFactory.sol	Type:	Code quality

Description

The following events in the `UniversalEmailRecoveryModule` and `EmailRecoveryModule` contracts are missing critical parameters:

1. The `RecoveryExecuted` event is missing the recovered account address.
2. The `NewValidatorRecovery` and `RemovedValidatorRecovery` events are missing the account address.

Listing 23. Excerpt from UniversalEmailRecoveryModule

```
39 event NewValidatorRecovery(address indexed validatorModule, bytes4
   recoverySelector);
40 event RemovedValidatorRecovery(address indexed validatorModule, bytes4
   recoverySelector);
41 event RecoveryExecuted();
```

These missing parameters reduce clarity and complicate off-chain tracking.

Additionally, both factories emit the same event, even though the deployed modules are different, which makes it impossible to distinguish between the two events.

Listing 24. Excerpt from EmailRecoveryFactory

```
26 event EmailRecoveryModuleDeployed(
27     address emailRecoveryModule, address emailRecoveryManager, address
   subjectHandler
28 );
```

Recommendation

Critical parameters should be included in the `RecoveryExecuted`, `NewValidatorRecovery`, and `RemovedValidatorRecovery` events. To improve code maintainability, consider moving these events to the `IEmailRecoveryModule` interface.

Create two separate events for `EmailRecoveryFactory` and `EmailRecoveryUniversalFactory`:

- In `EmailRecoveryFactory`, add additional `validator` and `functionSelector` parameters to the `EmailRecoveryModuleDeployed` event.
- Rename the event in `EmailRecoveryUniversalFactory` to `EmailUniversalRecoveryModuleDeployed`.

Fix 1.1

The issue was fixed by adding the missing parameters to all the specified events. Necessary changes were made to the `EmailRecoveryFactory` and `EmailRecoveryUniversalFactory` events to distinguish between the two factories.

Go back to Findings Summary

W6: Missing `AddedGuardian` event emission in `setupGuardians` function

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

Description

The `setupGuardians` function from the `GuardianUtils` library function is used to set up all guardians during module initialization. However, it does not emit the `AddedGuardian` event when adding guardians.

Listing 25. Excerpt from GuardianUtils

```
55 function setupGuardians(
56     mapping(address => IEmailRecoveryManager.GuardianConfig) storage
   guardianConfigs,
57     mapping(address => EnumerableGuardianMap.AddressToGuardianMap) storage
   guardiansStorage,
58     address account,
59     address[] memory guardians,
60     uint256[] memory weights,
61     uint256 threshold
62 )
63     internal
64 {
65     uint256 guardianCount = guardians.length;
66     if (guardianCount != weights.length) {
67         revert IncorrectNumberOfWeights();
68     }
69     if (threshold == 0) {
70         revert ThresholdCannotBeZero();
71     }
72     uint256 totalWeight = 0;
73     for (uint256 i = 0; i < guardianCount; i++) {
74         address guardian = guardians[i];
75         uint256 weight = weights[i];
76         if (guardian == address(0) || guardian == account) {
77             revert InvalidGuardianAddress();
```

```
78          }
79          // As long as weights are 1 or above, there will be enough total
    weight to reach the
80          // required threshold. This is because we check the guardian count
    cannot be less
81          // than the threshold and there is an equal amount of guardians to
    weights.
82          if (weight == 0) {
83              revert InvalidGuardianWeight();
84          }
85          GuardianStorage memory guardianStorage =
    guardiansStorage[account].get(guardian);
86          if (guardianStorage.status != GuardianStatus.NONE) {
87              revert AddressAlreadyGuardian();
88          }
89          guardiansStorage[account].set({
90              key: guardian,
91              value: GuardianStorage(GuardianStatus.REQUESTED, weight)
92          });
93          totalWeight += weight;
94      }
```

This inconsistency in event emission can lead to difficulties in tracking guardian additions off-chain.

Additionally, this function duplicates code from the `addGuardian` function, which emits the `AddedGuardian` event correctly.

Recommendation

Ensure that the `AddedGuardian` event is emitted when adding guardians in the `setupGuardians` function.

Fix 1.1

The issue was fixed by using the `addGuardian` function (which already emits the `AddedGuardian` event) to add guardians in the `setupGuardians` function.

Go back to Findings Summary

W7: ERC-4337 violation in `onInstall`

Impact:	Warning	Likelihood:	N/A
Target:	UniversalRecoveryModule.sol, EnumerableGuardianMap.sol	Type:	EIP violation

Description

ERC-4337 along with ERC-7562 define a set of rules that must be followed during the account abstraction user operation validation phase. The rules especially must be followed in the case of ERC-7579 validator modules.

The codebase contains two ERC-7579 executor modules, `EmailRecoveryModule` and `UniversalEmailRecoveryModule`. Although it is not strictly required by the ERC for these modules to follow the rules, the reference implementation of ERC-7579 smart accounts allows installation of these modules during the validation phase (initial smart account setup).

The module `UniversalEmailRecoveryModule` stores the list of allowed validators in the `validators` mapping that is accessed in the `onInstall` function.

Listing 26. Excerpt from UniversalEmailRecoveryModule

```
52 mapping(address account => SentinelListLib.SentinelList validatorList)
   internal validators;
```

Due to the implementation of `SentinelListLib.SentinelList`, the mapping is not ERC-4337 compliant.

Additionally, both modules call the `IEmailRecoveryManager.configureRecovery` function and, consequently, the `GuardianUtils.setupGuardians` function in the `onInstall` function.

The `GuardianUtils.setupGuardians` function is not ERC-4337 compliant

because it writes into the `guardiansStorage` mapping.

Listing 27. Excerpt from EmailRecoveryManager

```
82 mapping(address account => EnumerableGuardianMap.AddressToGuardianMap
   guardian) internal
83     guardiansStorage;
```

The mapping is not ERC-4337 compliant because the `EnumerableGuardianMap.AddressToGuardianMap` struct contains two nested mappings, neither of which uses the smart account address as the key.

Listing 28. Excerpt from EnumerableGuardianMap

```
45 struct AddressToGuardianMap {
46     // Storage of keys
47     EnumerableSet.AddressSet _keys;
48     mapping(address key => GuardianStorage) _values;
49 }
```

```
struct Set {
    // Storage of set values
    bytes32[] _values;
    // Position is the index of the value in the `values` array plus 1.
    // Position 0 is used to mean a value is not in the set.
    mapping(bytes32 value => uint256) _positions;
}

struct AddressSet {
    Set _inner;
}
```

Recommendation

Although it is not strictly required to have `onInstall` functions in ERC-7579 executor modules ERC-4337 compliant, it prevents users from installing the aforementioned modules during the initial smart account setup. Either well-

document that the modules cannot be installed during the smart account setup or make the modules ERC-4337 compliant.

Acknowledgment 1.1

Acknowledged by the client.

> Resolved by adding comments explaining the violation as decided it was too complex to make it compatible. Future versions could look to resolve this.
>
> — ZK Email Team

Go back to Findings Summary

I1: `getTrustedRecoveryManager` function returns public variable `emailRecoveryManager`

Impact:	Info	Likelihood:	N/A
Target:	EmailRecoveryModule.sol, UniversalEmailRecoveryModule.sol	Type:	Code quality

Description

In both `EmailRecoveryModule` and `UniversalEmailRecoveryModule` contracts, the `getTrustedRecoveryManager` function returns the `emailRecoveryManager` variable, which is already publicly accessible.

Recommendation

Either remove the `getTrustedRecoveryManager` function, or make the `emailRecoveryManager` variable private.

Fix 1.1

The issue was fixed by removing the `getTrustedRecoveryManager` function from both `EmailRecoveryModule` and `UniversalEmailRecoveryModule` contracts.

Go back to Findings Summary

I2: Non-immutable state variables in `EmailRecoveryManager` contract

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

Description

In the `EmailRecoveryManager` contract, the state variable `deployer` is not declared as `immutable`. It is likely intended to be set only once and remains unchanged throughout the contract's lifecycle.

Recommendation

Make the `deployer` variable `immutable`. Declaring variables as `immutable` can save gas and clarify code intent.

Fix 1.1

The issue was fixed by declaring the `deployer` variable as `immutable`.

Go back to Findings Summary

I3: Misleading naming

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

Description

The function name `isAuthorizedToRecover` suggests it checks if an entity is authorized to perform recovery actions. However, the intended functionality is to check if an entity is authorized to be recovered. This difference can lead to confusion about the function's purpose and its use within the system.

Recommendation

Consider renaming the `isAuthorizedToRecover` function to reflect its intended functionality better. Possibly use `isAuthorizedToBeRecovered`, which indicates that the function checks whether an entity is authorized to be the subject of a recovery process.

Fix 1.1

The issue was fixed by renaming the `isAuthorizedToRecover` function to `isAuthorizedToBeRecovered`.

Go back to Findings Summary

I4: Unchecked return values in `EnumerableGuardianMap` library

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

Description

The `EnumerableGuardianMap` library is used by the `GuardianUtils` contract to manage guardians, which are stored in a `guardiansStorage` mapping. The `set` and `remove` functions from the `EnumerableGuardianMap` library return a boolean, which indicates whether the added/removed data was present in the mapping before the operation. These return values can be used to simplify the logic in the following `GuardianUtils` functions:

- `addGuardian`
- `removeGuardian`
- `setupGuardians`

In the mentioned functions, the `guardianStorage.status != GuardianStatus.NONE` requirement can be removed in favor of reverting based on the return values from the `set` and `remove` functions, simplifying the code.

Recommendation

Consider refactoring the `addGuardian`, `removeGuardian`, and `setupGuardians` functions in `GuardianUtils` to check the return values of `set` and `remove` operations on `guardiansStorage` in favor of checking `guardianStorage.status != GuardianStatus.NONE`.

Fix 1.1

The issue was fixed by checking the return values of `set` and `remove`

operations on `guardiansStorage` in the `addGuardian`, `removeGuardian`, and `setupGuardians` functions.

Go back to Findings Summary

I5: Use `calldata` in favor of `memory` in function parameters

Impact:	Info	Likelihood:	N/A
Target:	-	Type:	Gas optimization

Description

When a function with a `memory` parameter is called externally, the function parameters are initially in calldata. To work with these parameters, Solidity has to:

- decode the ABI-encoded data in calldata;
- copy it into memory.

This process consumes more gas than if the function parameters were declared as `calldata` instead of `memory`.

Recommendation

Consider using `calldata` instead of `memory`, where arguments passed to the functions are only used and are not changing during the function call to save gas usage. The following contracts can be updated:

- `EmailRecoveryFactory`
- `EmailRecoveryManager`
- `EmailRecoverySubjectHandler`
- `EnumerableGuardianMap`
- `GuardianUtils`
- `UniversalEmailRecoveryModule`

Fix 1.1

The issue was fixed by updating the function parameters to use `calldata` instead of `memory` where suitable.

Go back to Findings Summary

I6: Floating pragma

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

Description

The project uses solidity floating pragma. A mistake in deployment can cause a version mismatch and, thus, an unexpected bug.

Recommendation

Consider fixing the pragma to a fixed version.

Acknowledgment 1.1

The issue was acknowledged.

> Chose not to implement for better compatibility with external contracts.
>
> — ZK Email Team

Go back to Findings Summary

I7: Missing zero-address validation in constructors

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

Description

The following contracts are missing data validation for address parameters that passed in their constructors:

- `UniversalEmailRecoveryModule`
- `EmailRecoveryModule`
- `EmailRecoveryFactory`
- `EmailRecoveryUniversalFactory`
- `EmailRecoveryManager`

By accident, an incorrect value (zero-address) can be passed to the constructor.

Recommendation

Consider adding zero-address checks for the address parameters.

Fix 1.1

The issue was fixed by adding zero-address checks for the address parameters in constructors.

Go back to Findings Summary

I8: Modifiers not above constructors

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

Description

The modifiers in the following contracts are placed below constructors:

- `EmailRecoveryManager`
- `UniversalEmailRecoveryModule`

Placing modifiers above the constructor is a common best practice in Solidity, which makes the code more readable.

Recommendation

Move the modifiers above the constructors.

Fix 1.1

The issue was fixed by moving said modifiers above the constructors.

Go back to Findings Summary

I9: Safe `validateRecoverySubject` optimization

Impact:	Info	Likelihood:	N/A
Target:	SafeRecoverySubjectHandler.sol	Type:	Gas optimization

Description

The function `validateRecoverySubject` in the `SafeRecoverySubjectHandler` contract validates recovery email subject parameters. As a part of the validation, the following operations are performed:

- It is checked that the old Safe owner to be replaced truly is the current Safe owner.
- All current Safe owners are requested to find an entry present before the Safe owner to be replaced inside a linked list.

Listing 29. Excerpt from SafeRecoverySubjectHandler.validateRecoverySubject

```
145 bool isOwner = ISafe(accountInEmail).isOwner(oldOwnerInEmail);
146 if (!isOwner) {
147     revert InvalidOldOwner();
148 }
149 if (newOwnerInEmail == address(0)) {
150     revert InvalidNewOwner();
151 }
```

Listing 30. Excerpt from SafeRecoverySubjectHandler.validateRecoverySubject

```
164 address previousOwnerInLinkedList =
165     getPreviousOwnerInLinkedList(accountInEmail, oldOwnerInEmail);
```

Recommendation

Both operations can be combined into a single one, requesting all current Safe owners and both checking the presence of the old Safe owner and finding the entry before it. Additionally, it can also be checked that the new Safe owner to be added is not already present in the list of current Safe owners.

Fix 1.1

The `SafeRecoverySubjectHandler.validateRecoverySubject` function was refactored as part of the fix for W3. `getPreviousOwnerInLinkedList` call was moved into the `parseRecoveryCalldata` function. The `newOwner` is now checked against existing owners.

> Didn't need to combine with `getPreviousOwnerInLinkedList` as that was moved into the `parseRecoveryCalldata` function. Did check the `newOwner` against existing owners.
>
> — ZK Email Team

Go back to Findings Summary

I10: Unused using-for directive

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

Description

The codebase contains an occurrence of an unused using-for directive. See Appendix C for more information about the using-for directive. This issue was detected using static analysis in Wake.

Recommendation

Remove the unused using-for directive.

Fix 1.1

The unused using-for directive was removed.

Go back to Findings Summary

Report revision 1.1

The main change since the previous revision 1.0 is the disallowed initialization of the system without guardians and a zero threshold. Additionally, the system accurately tracks the sum of weights from accepted guardians, disallowing entering the recovery process if the recovery threshold cannot be met. The gas usage of the contracts has been optimized, and all other reported issues have been addressed.

M5: `UniversalRecoveryModule` arbitrary Safe recovery call

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	UniversalEmailRecoveryModule.sol	Type:	Logical error

Description

`UniversalEmailRecoveryModule` can be set up to make recovery calls to arbitrary Safe/Safe7579 functions. This is possible due to Safe7579 reporting itself as a validator module, allowing the user to add a recovery method to the `UniversalEmailRecoveryModule`, setting the Safe account itself as a validator paired with any arbitrary function selector. This function selector could be set to any arbitrary Safe/Safe7579 function to be called during recovery. Several of these functions are only intended to be called from within Safe, and calling them from a recovery module could present a potential vulnerability.

The following functions pose a high risk if not properly restricted:

1. `execute` from Safe7579
2. `setFallbackHandler` from Safe
3. `setGuard` from Safe

Exploit scenario

The user sets up the `UniversalEmailRecoveryModule` with the Safe account as a validator and an arbitrary function selector. Upon recovery, guardians are then able to call the arbitrary function on the Safe account. The subsequent

exploit depends on the specific function selector set up during the recovery process. `setFallbackHandler` and `setGuard` functions would allow the attacker to install a malicious handler or guard on the Safe Smart Account, while the `execute` function allows execution any arbitrary function call.

Recommendation

Restrict the specified function selectors from being used during recovery.

Fix 1.2

Fixed by restricting the specified function selectors from being used during recovery.

Go back to Findings Summary

Report revision 1.2

A single change is present since the last revision 1.1, adding additional function selector restrictions to the recovery process to mitigate issue M5.

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, ZK Email: Email Recovery, 5.8.2024.

Appendix B: Glossary of terms

The following terms might be used throughout the document:

Superclass/Ancestor of C

A contract that C inherits/derives from.

Subclass/Child of C

A contract that inherits/derives from C.

Syntactic contract

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

Deployed contract

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

Init/initialization function

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

External entrypoint

A `public` or `external` function.

Public/Publicly-accessible function/entrypoint

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

Mutating function

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

Appendix C: Wake outputs

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

C.1. Detectors

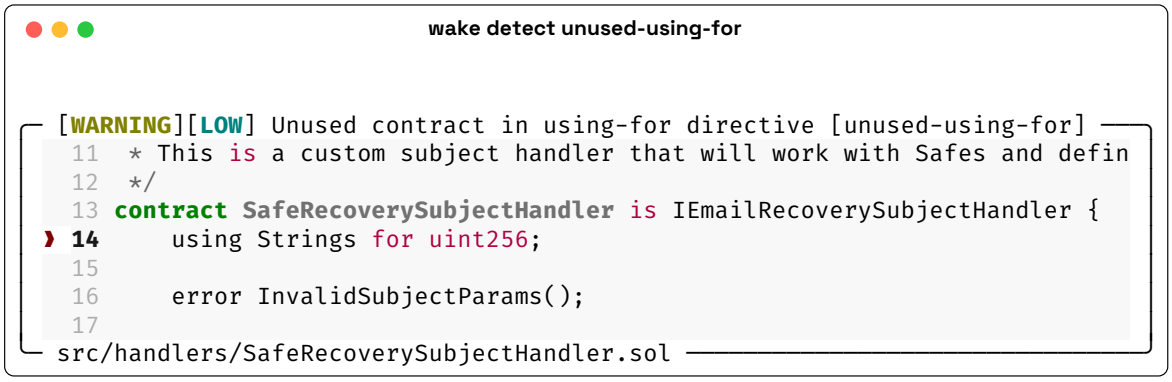

Figure 1. Unused using-for directive

ackee | blockchain security

Thank You

Ackee Blockchain a.s.


Prague, Czech Republic


hello@ackeeblockchain.com


https://twitter.com/AckeeBlockchain