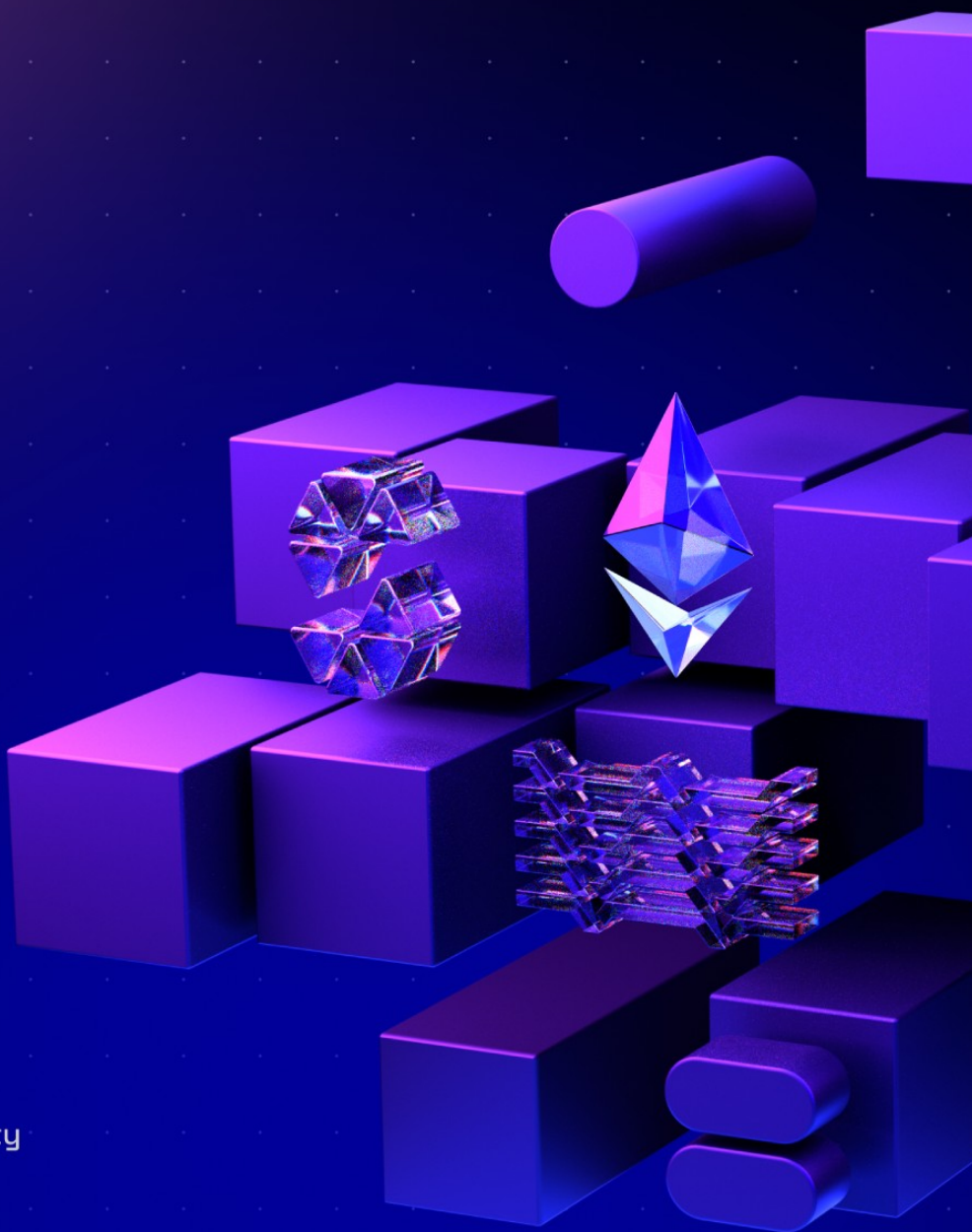
ackee
blockchain security

Fluidkey

Earn module

4.3.2025



Ackee Blockchain Security

Contents

1. Document Revisions

1.0-draft	Draft Report	21.02.2025
1.0	Final Report	27.02.2025
1.1	Final Report	04.03.2025

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain Security

Ackee Blockchain Security is an in-house team of security researchers performing security audits focusing on manual code reviews with extensive fuzz testing for Ethereum and Solana. Ackee is trusted by top-tier organizations in web3, securing protocols including Lido, Safe, and Axelar.

We develop open-source security and developer tooling Wake for Ethereum and Trident for Solana, supported by grants from Coinbase and the Solana Foundation. Wake and Trident help auditors in the manual review process to discover hardly recognizable edge-case vulnerabilities.

Our team teaches about blockchain security at the Czech Technical University in Prague, led by our co-founder and CEO, Josef Gattermayer, Ph.D. As the official educational partners of the Solana Foundation, we run the School of Solana and the Solana Auditors Bootcamp.

Ackee's mission is to build a stronger blockchain community by sharing our knowledge.

Ackee Blockchain a.s.
Rohanske nabrezi 717/4
186 00 Prague, Czech Republic
https://ackee.xyz
hello@ackee.xyz

2.2. Audit Methodology

1. **Verification of technical specification**
   The audit scope is confirmed with the client, and auditors are onboarded to the project. Provided documentation is reviewed and compared to the audited system.

2. **Tool-based analysis**
   A deep check with Solidity static analysis tool Wake in companion with Solidity (Wake) extension is performed, flagging potential vulnerabilities for further analysis early in the process.

3. **Manual code review**
   Auditors manually check the code line by line, identifying vulnerabilities and code quality issues. The main focus is on recognizing potential edge cases and project-specific risks.

4. **Local deployment and hacking**
   Contracts are deployed in a local Wake environment, where targeted attempts to exploit vulnerabilities are made. The contracts' resilience against various attack vectors is evaluated.

5. **Unit and fuzz testing**
   Unit tests are run to verify expected system behavior. Additional unit or fuzz tests may be written using Wake framework if any coverage gaps are identified. The goal is to verify the system's stability under real-world conditions and ensure robustness against both expected and unexpected inputs.

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* (system settings or parameters, such as deployment scripts, compiler configurations, using 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**	**N/A**
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*, 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* 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

The following table lists all contributors to this report. For authors of the specific revision, see the “Revision team” section in the respective “Report revision” chapter.

Member’s Name	Position
Martin Veselý	Lead Auditor
Štěpán Šonský	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

Fluidkey Earn module is a protocol designed to automate operations on multiple blockchain networks. The protocol's infrastructure includes Safe module integration for secure asset management. The system maintains security through proper authorization mechanisms and provides automated functionality for supported blockchains.

Revision 1.0

Fluidkey engaged Ackee Blockchain Security to perform a security review of the Fluidkey protocol with a total time donation of 3 engineering days in a period between February 17 and February 21, 2025, with Martin Veselý as the lead auditor.

The audit was performed on the commit `6aca8f`[1] and the scope was the following:

- `src/FluidkeyEarnModule.sol`

Our review process combined automated analysis using Wake and manual review of the codebase. For testing purposes, we utilized the Wake testing framework. During the review, we paid special attention to:

- ensuring the arithmetic of the system is correct;
- detecting possible reentrancies in the code;
- ensuring access controls are not too relaxed or too strict;
- checking the signature verification;
- writing comprehensive unit tests using Wake framework;
- looking for common issues such as data validation.

Our review resulted in 9 findings, ranging from Info to High severity. The most

severe one H1 which describes the possible replay attack on the protocol. And finding M1 poses a risk of exceeding the `MAX_TOKENS` limit leading to unexpected behavior.

Ackee Blockchain Security recommends Fluidkey:

- fix the `MAX_TOKENS` validation in `setConfig` function;
- implement proper cleanup during module uninstallation;
- add chain ID to signature verification;
- address all other reported issues.

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

Revision 1.1

The review was done on the given commit `665108`[2]. The Fluidkey team has fixed all issues reported in the previous revision.

[1] full commit hash: `6aca8f7e203cadad6287f643cb3ef050ffe1ef01`

[2] full commit hash: `665108134f4ca9a9d78b11396a16711a09b8f7eb`

4. Findings Summary

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

- *Description*
- *Exploit scenario* (if severity is low or higher)
- *Recommendation*
- *Fix* (if applicable).

Summary of findings:

Critical	High	Medium	Low	Warning	Info	Total
0	1	1	0	3	4	9

Table 2. Findings Count by Severity

Findings in detail:

Finding title	Severity	Reported	Status
H1: Cross-chain replay attack vulnerability	High	1.0	Fixed
M1: `MAX_TOKENS` limit bypass via `setConfig` leading to unintended module persistence	Medium	1.0	Fixed
W1: Unchecked return value	Warning	1.0	Fixed
W2: Module installation allows empty configuration	Warning	1.0	Fixed
W3: Misleading event in deleteConfig	Warning	1.0	Fixed

Finding title	Severity	Reported	Status
I1: Variable can be immutable	Info	1.0	Fixed
I2: Incorrect usage of immutable instead of constant	Info	1.0	Fixed
I3: Misleading documentation	Info	1.0	Fixed
I4: Unnecessary external call	Info	1.0	Fixed

Table 3. Table of Findings

Report Revision 1.0

Revision Team

Member's Name	Position
Martin Veselý	Lead Auditor
Štěpán Šonský	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

Fluidkey is a protocol designed to automate operations on multiple blockchain networks. The protocol utilizes a modular architecture that enables authorized relayers to trigger automated operations through a Safe module. The system implements comprehensive security measures, including signature verification for transaction initiation and authorization checks for all operations. The protocol's infrastructure is built to handle automated asset management tasks while maintaining strict security requirements.

Trust Model

Users must trust the Safe module implementation to handle deposits securely and not contain vulnerabilities that could compromise funds. The protocol relies on authorized relayers to initiate automated deposits, requiring trust in their behavior and key management. The integrated ERC-4626 vaults must be trusted to properly manage deposited assets.

Findings

The following section presents the list of findings discovered in this revision. For the complete list of all findings, Go back to Findings Summary

H1: Cross-chain replay attack vulnerability

High severity issue

Impact:	High	Likelihood:	Medium
Target:	FluidkeyEarnModule.sol	Type:	Replay attack

Description

The `autoEarn` function generates a signature verification hash without including the chain ID in its calculation. This implementation enables potential signature replay attacks when the module is deployed across multiple chains.

Listing 1. Excerpt from FluidkeyEarnModule

```
291 bytes32 hash = keccak256(abi.encodePacked(token, amountToSave, safe,
    nonce));
```

Exploit scenario

Alice signs a valid transaction on Chain A.

Bob observes Alice's transaction and signature on Chain A.

The same module is deployed on Chain B.

Bob replays Alice's transaction with the captured signature on Chain B.

The signature verification succeeds because the chain ID is not included in the verification hash.

As a result, Bob successfully executes an unauthorized operation using Alice's signature.

Recommendation

Include the chain ID in the hash calculation for signature verification to ensure signatures are only valid for the specific chain they were intended for.

Fix 1.1

The issue was fixed by adding the chain ID to the hash calculation.

Go back to Findings Summary

M1: `MAX_TOKENS` limit bypass via `setConfig` leading to unintended module persistence

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	FluidkeyEarnModule.sol	Type:	Data validation

Description

The module implements a `MAX_TOKENS` limit in the `onInstall` function to restrict the number of token configurations per Safe. However, this limit can be bypassed using the `setConfig` function, which lacks the same validation check. This allows a Safe to configure more tokens than intended, leading to the following issues:

- the `MAX_TOKENS` limit becomes ineffective as a security control;
- excessive configurations persist and remain active after module removal;
- the `autoEarn` function remains callable for persisted configurations even when `isInitialized` returns false, allowing unauthorized token movements; and
- while this vulnerability requires an authorized relayer, which reduces exploitation likelihood, it represents an architectural flaw where an uninstalled module retains functionality.

Exploit scenario

Alice installs the module with 90 token-vault configurations through the `onInstall` function, where `MAX_TOKENS` is limited to 100.

Alice adds 12 more configurations through the `setConfig` function, and the `MAX_TOKENS` limit is bypassed.

When Alice calls `onUninstall`, the function only removes the first 100 configurations, leaving 2 configurations active in the system.

Bob, an authorized relayer, executes `autoEarn` function calls for these remaining configurations. This allows unauthorized token movements even after the module is supposedly uninstalled.

Recommendation

Add a validation check in the `setConfig` function to ensure the number of configurations does not exceed the `MAX_TOKENS` limit. Additionally, implement proper cleanup logic in the `onUninstall` function to ensure all configurations are completely removed from the system.

Fix 1.1

The issue was fixed by adding a validation check in the `setConfig` function. Cleanup logic in the `onUninstall` function was not implemented.

Go back to Findings Summary

W1: Unchecked return value

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

Description

The `execTransactionFromModule` call does not check its return value. This function returns a boolean indicating success or failure, and the missing validation could mask potential failures in the operation.

Listing 2. Excerpt from FluidkeyEarnModule

```
351 if (token == address(ETH)) {
352     safeInstance.execTransactionFromModule(
353         address(wrappedNative),
354         amountToSave,
355         abi.encodeWithSelector(IWrappedNative.deposit.selector),
356         0
357     );
```

Recommendation

Add a check for the return value of `execTransactionFromModule` and revert the transaction if the wrapping operation fails.

Fix 1.1

The issue was fixed by adding a check for the return value of `execTransactionFromModule` and reverting the transaction if the wrapping operation fails.

Go back to Findings Summary

W2: Module installation allows empty configuration

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

Description

The `onInstall` function does not validate whether the `_configs` array passed in the `data` parameter is empty. If an empty array is provided, the function will still proceed with the initialization process and emit the `ModuleInitialized` event without setting up any configurations.

Recommendation

Add validation to check if the input array is empty and handle this case appropriately.

Fix 1.1

The issue was fixed by adding a validation check in the `onInstall` function.

Go back to Findings Summary

W3: Misleading event in deleteConfig

Impact:	Warning	Likelihood:	N/A
Target:	FluidkeyEarnModule.sol	Type:	Logging

Description

The function `deleteConfig` emits the same event as the `setConfig` function, which can lead to misleading event logs.

Recommendation

Add a new event `ConfigDeleted` for the `deleteConfig` function.

Fix 1.1

The issue was fixed by adding a new event `ConfigDeleted` for the `deleteConfig` function.

Go back to Findings Summary

I1: Variable can be immutable

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

Description

The `wrappedNative` address is declared as a state variable but is only set once during initialization and never modified afterwards. This variable can be declared as `immutable` to save gas costs, as `immutable` variables are cheaper to read than regular state variables.

Listing 3. Excerpt from FluidkeyEarnModule

```
57 address public wrappedNative;
```

Listing 4. Excerpt from FluidkeyEarnModule

```
59 constructor(address _authorizedRelayer, address _wrappedNative)
   Ownable(msg.sender) {
60     authorizedRelayers[_authorizedRelayer] = true;
61     emit AddAuthorizedRelayer(_authorizedRelayer);
62     wrappedNative = _wrappedNative;
63 }
```

Recommendation

Change the declaration to use `immutable`.

Fix 1.1

The issue was fixed by changing the declaration to `immutable`.

Go back to Findings Summary

I2: Incorrect usage of immutable instead of constant

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

Description

The `ETH` address is declared as `immutable` when it should be `constant` since it's a fixed value that never changes and is known at compile time.

Listing 5. Excerpt from FluidkeyEarnModule

```
56 address public immutable ETH = 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE;
```

Recommendation

Change the declaration to use `constant`.

Fix 1.1

The issue was fixed by changing the declaration to `constant`.

Go back to Findings Summary

I3: Misleading documentation

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

Description

The documentation for the `prevToken` parameter is unclear as it doesn't explain that the parameter is a pointer in a linked list implementation.

Listing 6. Excerpt from FluidkeyEarnModule

```
220 * @param prevToken address of the token stored before the token to be
    deleted
```

Recommendation

Update the parameter documentation to clearly explain its role in the linked list structure.

Fix 1.1

The issue was fixed by updating the parameter documentation.

Go back to Findings Summary

I4: Unnecessary external call

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

Description

The `getAllConfigs` function makes an unnecessary external call to `this.getTokens()` instead of directly accessing the internal state. This pattern forces the EVM to make an external call to the same contract, which is more gas expensive than directly accessing the data.

Recommendation

Remove the external call and use direct internal access to the `tokens` mapping.

Fix 1.1

The issue was fixed by removing the external call and using direct internal access to the `tokens` mapping.

Go back to Findings Summary

Report Revision 1.1

Revision Team

Revision team is the same as in Report Revision 1.0.

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain Security, Fluidkey: Earn module, 4.3.2025.

Appendix B: Wake Findings

This section lists the outputs from the Wake framework used for testing and static analysis during the audit.

ackee
blockchain security

Thank You

Ackee Blockchain a.s.

Rohanske nabrezi 717/4
186 00 Prague
Czech Republic

hello@ackee.xyz