ackee
blockchain security

CoW

Flash Loan Router

26.3.2025

Contents

1. Document Revisions

1.0-draft	Draft Report	21.03.2025
1.0	Final Report	24.03.2025
1.1	Fix Review	26.03.2025

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
Michal Převrátil	Lead Auditor
Dmytro Khimchenko	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

CoW Flash Loan Router is an extension of the CoW Protocol that enables trade solvers to execute multiple flash loans prior to trade settlements. The system integrates with various loan providers through dedicated adapter contracts, allowing for sequential flash loan executions during the settlement process.

Revision 1.0

CoW engaged Ackee Blockchain Security to perform a security review of the CoW protocol with a total time donation of 5 engineering days in a period between March 17 and March 21, 2025, with Michal Převrátil as the lead auditor. Ackee Blockchain Security allocated 1 additional engineering day to ensure high confidence, particularly regarding the integration of the audited code with the CoW Protocol Core.

The audit was performed on the commit `930914f`[1]. The scope included all Solidity files in the `src` directory, excluding the `src/vendored` directory. The vendored files were reviewed only in terms of their usage in the codebase, with their implementation being out of scope.

We began our audit with a thorough analysis of the contract logic, identifying potential attack vectors and trust model implications. We then employed static analysis tools, including [Wake](), to verify the absence of common issues. During the review, we focused on ensuring:

- assembly code contains no logic errors, including memory safety violations;

- trade settlement payloads remain tamper-proof;

- reentrancy attacks are prevented;

- solvers, tokens, lenders, and borrowers cannot compromise user funds;

- compliance with the [ERC-3156](#) standard;

- correct usage of transient storage; and

- identification of common issues and gas optimization opportunities.

Our review resulted in 4 findings, ranging from Info to Warning severity.

The code demonstrated exceptional quality, with findings primarily related to code and gas optimization improvements. The codebase features comprehensive documentation, including clear explanations of caveats and code correctness reasoning. The system trust model, expected usage, and security assumptions are thoroughly documented.

Ackee Blockchain Security recommends CoW to review and address the identified findings.

See [Report Revision 1.0](#) for the system overview and trust model.

Revision 1.1

CoW engaged Ackee Blockchain Security to perform a fix review of the findings from the previous revision. The review was done on the given commit `f9c1867`[2].

3 out of 4 findings were fixed, and [I2](#) was acknowledged to benefit from flash loan fee waiving. No new findings were discovered.

The `Bytes` and `LoansWithSettlement` libraries were updated with documentation comments that describe usage assumptions and possible pitfalls when used incorrectly.

[1] full commit hash: `930914fd8b73a9f9c6a628beada616110fb8ec35`

[2] full commit hash: `f9c18679df0f5cda19611b05fca1c982786071eb`

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	0	0	0	1	3	4

Table 2. Findings Count by Severity

Findings in detail:

Finding title	Severity	Reported	Status
W1: Missing events	Warning	1.0	Fixed
I1: Documentation errors	Info	1.0	Fixed
I2: Aave flash loan call optimization	Info	1.0	Acknowledged
I3: Missing `view` in interfaces	Info	1.0	Fixed

Table 3. Table of Findings

Report Revision 1.0

Revision Team

Member's Name	Position
Michal Převrátil	Lead Auditor
Dmytro Khimchenko	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

CoW Flash Loan Router is an extension of the CoW Protocol that allows solvers, entities responsible for fullfilling user trades, to take flash loans before performing settlement of user trades.

Solvers are free to use any flash loan adapter, designed for interaction with any flash loan provider. The CoW Flash Loan Router guarantees that the trades settlement payload is not tampered with by the flash loan provider and adapter.

Trust Model

Tokens interacted with, flash loan adapters and flash loan providers are trusted not to disrupt the transaction execution, causing the transaction to revert. These entities are also trusted not to abuse the front-running opportunity to worsen the market conditions up to the slippage tolerance set in the trades to be settled.

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

W1: Missing events

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

Description

The codebase does not define a single event to support off-chain infrastructure.

The CoW Settlement contract emits an event upon successful settlement:

```
emit Settlement(msg.sender);
```

`msg.sender` in this case refers to the solver performing the settlement.

When using the `FlashLoanRouter` contract, the `msg.sender` is the `FlashLoanRouter` contract itself, causing the loss of information about the original solver invoking `FlashLoanRouter`.

Recommendation

Consider adding an event to the `FlashLoanRouter` contract to track settlements made and solvers performing them.

Fix 1.1

A new `Settlement` event was added to the `FlashLoanRouter` contract to track solvers calling the `flashLoanAndSettle` function.

Listing 1. Excerpt from FlashLoanRouter.flashLoanAndSettle

```
77 // The event is emitted before the actual settlement is executed to
78 // avoid having to carry `msg.sender` up the call stack (and related gas
79 // overhead). The contract guarantees that the call reverts if no
80 // settlement is executed.
```

```
81 emit Settlement(msg.sender);
```

[Go back to Findings Summary](#)

I1: Documentation errors

Impact:	Info	Likelihood:	N/A
Target:	Borrower.sol, LoansWithSettlement.sol	Type:	Code quality

Description

The `Borrower.sol` and `LoansWithSettlement.sol` contracts contain typographical and grammatical errors in their comments:

The article "An" should be "A" in the following code listing:

Listing 2. Excerpt from [Borrower](#)

```
12 /// @notice An generic implementation of a borrower that is designed to make
   it
```

The word "fist" should be "first" in the following code listing:

Listing 3. Excerpt from [LoansWithSettlement](#)

```
73 // Keep track of the fist yet-unwritten-to byte
```

Recommendation

Fix the documentation errors.

Fix 1.1

The documentation errors were fixed.

[Go back to Findings Summary](#)

I2: Aave flash loan call optimization

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

Description

The `AaveBorrower` contract uses the `flashLoan` function to perform a flash loan call to the Aave pool.

Listing 4. Excerpt from AaveBorrower.triggerFlashLoan

```
26  address[] memory assets = new address[](1);
27  assets[0] = address(token);
28  uint256[] memory amounts = new uint256[](1);
29  amounts[0] = amount;
30  uint256[] memory interestRateModes = new uint256[](1);
31  // Don't open any debt position, just revert if funds can't be
32  // transferred from this contract.
33  interestRateModes[0] = 0;
34  // The next value is technically unused, since `interestRateMode` is 0.
35  address onBehalfOf = address(this);
36  bytes calldata params = callBackData;
37  // Referral supply is currently inactive
38  uint16 referralCode = 0;
39  IAavePool(lender).flashLoan(
40      address(receiverAddress), assets, amounts, interestRateModes, onBehalfOf,
    params, referralCode
41  );
```

The `flashLoan` function is a generic flash loan entry point that supports multiple assets and flash loan fee waiving if eligible.

However, the code performs a flash loan call for only one asset. If only Aave v3 is needed to be supported and the fee waiving is not expected to be used, the `flashLoan` call can be replaced with `flashLoanSimple` call for better gas efficiency.

Recommendation

Consider replacing the `flashLoan` call with `flashLoanSimple` call if only Aave v3 is needed to be supported and the fee waiving is not expected to be used.

Acknowledgment 1.1

The finding was acknowledged. The client decided to keep the `flashLoan` call to maintain the fee waiving benefit.

[Go back to Findings Summary](#)

I3: Missing `view` in interfaces

Impact:	Info	Likelihood:	N/A
Target:	ICowSettlement.sol, IFlashLoanRouter.sol	Type:	Code quality

Description

The following functions in the interfaces lack the `view` keyword in their declarations:

Listing 5. Excerpt from [ICowSettlement](#)

```
41 function authenticator() external returns (address);
```

Listing 6. Excerpt from [IFlashLoanRouter](#)

```
44 function settlementContract() external returns (ICowSettlement);
```

Listing 7. Excerpt from [IFlashLoanRouter](#)

```
48 function settlementAuthentication() external returns (ICowAuthentication);
```

Using `view` and `pure` in interface functions improves code readability, allows the compiler to enforce state immutability in inheriting contracts, allows the compiler to use `staticcall` and enforce state immutability in the contract being called through the interface, and helps tooling to correctly recognize functions to be invoked as calls versus transactions.

Recommendation

Add the `view` keyword to the corresponding functions in the `ICowSettlement` and `IFlashLoanRouter` interfaces.

Fix 1.1

The `view` keyword was added to the corresponding functions in the
`ICowSettlement` and `IFlashLoanRouter` interfaces.

[Go back to Findings Summary](#)

Report Revision 1.1

Revision Team

Member's Name	Position
Michal Převrátil	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

Overview

Since there were no comprehensive changes in this revision, the complete overview is listed in the Executive Summary section Revision 1.1.

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain Security, CoW: Flash Loan Router, 26.3.2025.

ackee
blockchain security

Thank You

Ackee Blockchain a.s.

Rohanske nabrezi 717/4
186 00 Prague
Czech Republic

hello@ackee.xyz