cega × PESSIMISTIC

SECURITY ANALYSIS

by Pessimistic

This report is public

September 6, 2024

Abstract 2
Disclaimer 2
Summary 2
General recommendations 2
Project overview 3
Project description 3
Codebase update 3
Audit process 4
Manual analysis 5
Critical issues 5
C01. Repeated order filling (fixed) 5
Medium severity issues 6
M01. Excessive amount of ether sent (fixed) 6
M02. Potential overflows (fixed) 6
Low severity issues 6
L01. Gas/readability (fixed) 6
Notes 7
N01. Arbitrary takerAsset (commented) 7
N02. Arbitrum timestamp specification (commented) 7
N03. Unlimited approval (commented) 7
N04. User interactions (commented) 7

ABSTRACT

In this report, we consider the security of smart contracts of Cega LP Offramp project. Our task is to find and describe security issues in the smart contracts of the platform.

DISCLAIMER

SUMMARY

In this report, we considered the security of Cega LP Offramp smart contracts. We described the audit process in the section below.

The audit showed one critical issue: Repeated order filling. The audit also revealed two issues of medium severity: Excessive amount of ether sent and Potential overflows. Moreover, one low-severity issue was found.

After the initial audit, the developers provided us with a new version of the code. In that version, they fixed all found issues.

The overall code quality is good.

GENERAL RECOMMENDATIONS

We recommend fixing and enhancing CI to run tests, calculate code coverage, and analyze code with linters and security tools.

PROJECT OVERVIEW

Project description

For the audit, we were provided with Cega LP Offramp project on a private GitHub repository, commit ae71c52ff30fff00c36fb4b0b8925a87dc4f435d.

The scope of the audit included changes added with commit ae71c52 (**Structs.sol**, **ICegaCombinedEntry.sol**, **OfframpEntry.sol**, **IOfframpEntry.sol**, **OrderLib.sol**, **OfframpStorage.sol**, **Errors.sol**).

Other parts of the protocol and the potential effects of the Offramp on those parts were left out of the scope of this audit.

The documentation for the project included a private google document.

All 13 tests pass successfully. The code coverage could not be calculated.

The total LOC of audited sources is 265.

Codebase update

After the audit, the developers provided us with a new commit: f74844fdc840eabe09b71a75369e30cb83a5b271. In that update, they fixed all of the found issues.

13 tests pass successfully.

AUDIT PROCESS

We started the audit on August 27, 2024, and finished on August 29, 2024.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project. After a discussion, we performed preliminary research.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- Whether user funds are safe,
- Whether order fill calculations are correct,
- Whether it is possible to harm the Cega project with a malicious order,
- Whether the contracts are compatible with the Arbitrum chain.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts.

We ran tests.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

After the initial audit, we discussed the results with the developers. On September 4, 2024 the developers provided us with an updated version of the code. In this update, they fixed all of the issues from our report.

We reviewed the updated codebase, run the aforementioned tools and updated the report accordingly.

MANUAL ANALYSIS

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

C01. Repeated order filling (fixed)

The function `fillOrder` of the **OfframpEntry** contract does not check if the order has already been filled out. Orders that do not allow partial fill can be filled repeatedly until the maker runs out of LP tokens. Besides, the maker cannot limit it via `approve`, as the contract holds unconditional approvals for all LP tokens.

> **The issue has been fixed and is not present in the latest version of the code.**

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. **Excessive amount of ether sent** (fixed)

The function `fillOrder` of the **OfframpEntry** contract can accept extra native currency. The `msg.value` check at line 130 does not prevent sending excessive amounts of ether. It seems suboptimal, as one can calculate precisely the required amount of native currency in advance.

> The issue has been fixed and is not present in the latest version of the code.

M02. **Potential overflows** (fixed)

Intermediate results of calculations at lines 47-49 and at line 50 of the **OrderLib** contract, are limited to `uint128`, which might not be enough. As a result, some orders might become impossible to execute. We recommend using `uint256` at least for intermediate results, and preferably avoid small integer types.

> The issue has been fixed and is not present in the latest version of the code.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. **Gas/readability** (fixed)

The struct `OfframpOrder` in the **Structs.sol** is never saved to storage, yet it utilizes small integer types. We recommend declaring all non-storage variables (struct fields) as `uint256`, since it reduces the risk of overflows, produces cleaner bytecode, and usually saves a bit of gas.

> The issue has been fixed and is not present in the latest version of the code.

Notes

N01. **Arbitrary takerAsset** (commented)

Users can specify arbitrary `takerAsset`, including one they fully control. Attacker can use it to scam other users or as a building block of a complex hack.

> `Comment from the developers:`
> Protocol allows arbitrary takerAsset by design, to make various swaps possible. Taker asset is part of maker's signature, so it will always be the asset user intended to trade against.

N02. **Arbitrum timestamp specification** (commented)

The codebase relies on `block.timestamp`, which behaves slightly differently on Arbitrum. Make sure it does not affect your project.

> `Comment from the developers:`
> Differences in block.timestamp are negligible in our case and don't allow to exploit contract or it's funds.

N03. **Unlimited approval** (commented)

The diamond holds unconditional approvals for all LP tokens. It might break the security assumptions of third-party contracts and lead to LP token theft via malicious orders.

> `Comment from the developers:`
> Unlimited approval for LP tokens held by CegaEntry was always part of the protocol and doesn't bring any new risks. Signing malicious data is considered a user's mistake, not a protocol vulnerability.

N04. **User interactions** (commented)

One can create and execute an order without ever interacting with the application. One can also cancel an order without creating it first. While it does not affect the contract in any way, these scenarios may disrupt the work of frontend and backend.

> `Comment from the developers:`
> Cega's frontend and backend always assume that user can interact with protocol directly through smart contracts.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Yhtyyar Sahatov, Security Engineer
Konstantin Zherebtsov, Business Development Lead
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

September 6, 2024