DEFIMOON

be secure

Smart Contract Audit Report

September, 2022

inverse finance

CONTACTS

DEFIMOON PROJECT

Audit and Development

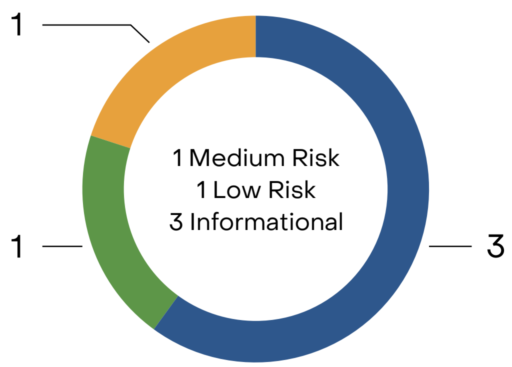
https://defimoon.org
audit@defimoon.org
defimoon_org
defimoonorg
defimoon
defimoonorg

DEFIMOON
be secure

September 23st 2022
This audit report was prepared by Defimoon for InverseFinance

Audit information

Description	The contracts implement Fed system, providing Dola liquidity to various protocols
Project website	https://www.inverse.finance/
Audited files	ConvexFed.sol, CurvePoolAdapter.sol
Audited by	Cyrill Novoseletskyi, Ilya Vaganov
Approved by	Artur Makhnach, Cyrill Minyaev
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Manual Review
Source code	https://github.com/InverseFinance/feds
Commit hash	ac88b2b
Network	Ethereum mainnet
Status	Not Passed



	High Risk	A fatal vulnerability that can cause the loss of all Tokens / Funds.
	Medium Risk	A vulnerability that can cause the loss of some Tokens / Funds.
	Low Risk	A vulnerability which can cause the loss of protocol functionality.
	Informational	Non-security issues such as functionality, style, and convention.

Disclaimer

This audit is not financial, investment, or any other kind of advice and could be used for informational purposes only. This report is not a substitute for doing your own research and due diligence should always be paid in full to any project. Defimoon is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Defimoon has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Defimoon is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. Defimoon has no connection to the project other than the conduction of this audit and has no obligations other than to publish an objective report. Defimoon will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Defimoon assumes that the provided information and materials were not altered, suppressed, or misleading. This report is published by Defimoon, and Defimoon has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Defimoon. In instances where an auditor or team member has a personal connection with the audited project, that auditor or team member will be excluded from viewing or impacting any internal communication regarding the specific audit.

Audit Information

Defimoon utilizes both manual and automated auditing approach to cover the most ground possible. We begin with generic static analysis automated tools to quickly assess the overall state of the contract. We then move to a comprehensive manual code analysis, which enables us to find security flaws that automated tools would miss. Finally, we conduct an extensive unit testing to make sure contract behaves as expected under stress conditions.

In our decision making process we rely on finding located via the manual code inspection and testing. If an automated tool raises a possible vulnerability, we always investigate it further manually to make a final verdict. All our tests are run in a special test environment which matches the "real world" situations and we utilize exact copies of the published or provided contracts.

While conducting the audit, the Defimoon security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Defimoon assesses the risks and assigns a risk level to each section together with an explanatory comment.

Audit overview

Major security issues were found.

The `lpForDola` method returns an incorrect value, since this function is used in a large layer of logic, this is a serious problem (DFM-1).
To avoid various undesired situations, it is better to make an approval for the required amount only when necessary. It is worth remembering that even if the contract itself, which is approved, has no bad intentions, it can always be hacked or attacked (DFM-2).
There are recurring functions in the contracts, which carry additional gas fees and complicate the codebase (DFM-3).
Function and variable names are not adherent to NatSpec standard (DFM-4).
Even though contracts have 2 roles and role-managment in this context is simple, it is always recommended to utilize well-known, well-tested and community accepted tools, such as `AccessControl` contract from the OpenZeppelin library (DFM-5).

Summary of findings

According to the standard audit assessment, the audited solidity smart contracts are not secure and are not ready for production.

ID	Description	Severity
DFM-1	Incorrect calculations	Medium Risk
DFM-2	Unlimited approve	Low Risk
DFM-3	Recurrent functionality	Informational
DFM-4	Naming functions and variables	Informational
DFM-5	No AccessControl	Informational
DFM-6	No licenses	Informational

Application security checklist

Compiler errors	Passed
Possible delays in data delivery	Passed
Timestamp dependence	Passed
Integer Overflow and Underflow	Passed
Race Conditions and Reentrancy	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Methods execution permissions	Passed
Private user data leaks	Passed
Malicious Events Log	Passed
Scoping and Declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Design Logic	Passed
Cross-function race conditions	Passed

Detailed Audit Information

Contract Programming

Solidity version not specified	Passed
Solidity version too old	Passed
Integer overflow/underflow	Passed
Function input parameters lack of check	Passed
Function input parameters check bypass	Passed
Function access control lacks management	Passed
Critical operation lacks event log	Passed
Human/contract checks bypass	Passed
Random number generation/use vulnerability	Passed
Fallback function misuse	Passed
Race condition	Passed
Logical vulnerability	Passed
Other programming issues	Not Passed

Code Specification

Visibility not explicitly declared	Passed
Variable storage location not explicitly declared	Passed
Use keywords/functions to be deprecated	Passed
Other code specification issues	Passed

Gas Optimization

Assert () misuse	Passed
High consumption 'for/while' loop	Passed
High consumption 'storage' storage	Passed
"Out of Gas" Attack	Passed

Findings

DFM-1 «Incorrect calculations»

Severity: Medium Risk

Description: The `lpForDola` function is used in a large cluster of contract logic, but the function itself incorrectly calculates the number of lptokens, which does not allow you to withdraw the full amount of tokens.

Example from the tests:

- Step 1
Amount to contraction 0.2
Calculated lpForDola 0.198683421801589882
LP balance before 0.993218783674628816
LP balance after 0.794535361873038934
LP withdrawn 0.198683421801589882

- Step 2
Amount to contraction 0.2
Calculated lpForDola 0.198683421808310282
LP balance before 0.794535361873038934
LP balance after 0.595851940064728652
LP withdrawn 0.198683421808310282

- Step 3
Amount to contraction 0.2
Calculated lpForDola 0.198683421815030682
LP balance before 0.595851940064728652
LP balance after 0.39716851824969797
LP withdrawn 0.198683421815030682

- Step 4
Amount to contraction 0.2
Calculated lpForDola 0.198683421821751081
LP balance before 0.39716851824969797
LP balance after 0.198485096427946889
LP withdrawn 0.198683421821751081

- Step 5
Amount to contraction 0.2
Calculated lpForDola 0.198683421828471482
LP balance before 0.198485096427946889

- Transaction was reverted!
Calculated lpForDola 0.198683421828471482
Total LP Balance 0.198485096427946889

The requested value is greater than the actual value!

Recommendation: Revise this function call chain and correct the calculations.

DFM-2 «Unlimited approve»

Severity: Low Risk

Description: In order to avoid various undesired situations, it is better to do approve for the required amount when necessary. It is worth remembering that even if the contract itself, which is being approved, does not have bad intentions, it can always be hacked or attacked.

Recommendation: Set the exact certain number of tokens for permission. After calling the desired function, it is better to set approve to zero.

DFM-3 «Recurrent functionality»

Severity: Informational

Description: The `metapoolDeposit` and `metapoolWithdraw` functions use repetitive logic that is already present in existing functions.

Recommendation:
1) In the `metapoolDeposit` function, when finding `minCrvLPOut`, use the `applySlippage` function.

2) In the `metapoolWithdraw` function, when finding `amountCRVLP`, use `lpForDola`.

DFM-4 «Naming functions and variables»

Severity: Informational

Description: The names of both variables and functions do not correspond to their visibility.

Recommendation: Variables and functions that have an internal/private scope must start with an underscore.

DFM-5 «No AccessControl»

Severity: Informational

Description: The contract has 2 roles that are used in the functionality of contracts: gov and chair. But this is not a recommended implementation, as there is a safer and more convenient tool of their OpenZeppelin library.

Recommendation: Add inheritance from the `AccessControl` contract and set roles in accordance with the recommendations of the developers of the OpenZeppelin library.

DFM-6 «No licenses»

Severity: Informational

Description: The absence of licenses in contracts increases users' distrust of them.

Recommendation: Add licenses to each contact file according to this documentation.

Methodology

Automated Analyses

Slither
Slither has reported 129 findings. These results were either related to code from dependencies, false positives or have been integrated in the findings or best practices of this report.

Manual Code Review

We prefer to work with a transparent process and make our reviews a collaborative effort. The goal of our security audits is to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, review open issue tickets, and investigate details other than the implementation.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system to make a final decision.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Appendix A — Finding Statuses

Resolved	Contracts were modified to permanently resolve the finding
Mitigated	The finding was resolved by other methods such as revoking contract ownership or updating the code to minimize the effect of the finding
Acknowledged	Project team is made aware of the finding
Open	The finding was not addressed