Smart Contract Audit Report

July, 2022

SiriusHandler.sol

DEFIMOON PROJECT

Audit and
Development

CONTACTS

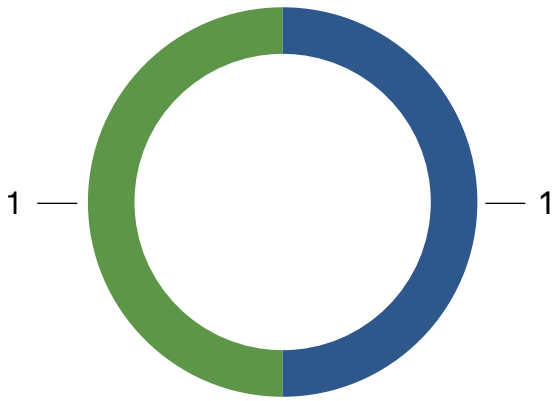
https://defimoon.org
audit@defimoon.org
defimoon_org
defimoonorg
defimoon
defimoonorg

DEFIMOON
be secure

July 23th 2022
This audit report was prepared by Defimoon for Algem

Audit information

Description	Utility contract for calculating the share of receipt tokens
Auditors	Cyrill Minyaev, Alex Zhelyabin
Approved by	Artur Makhnach
Audited file	SiriusHandler.sol
Timeline	22th - 23th July 2022
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Manual Review
Specification	Documentation
Docs quality	High
Source code	Algem git repo / Commit: 8de0cbb
Chain	Astar Network



	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

No major security issues were found.

However some points should be taken into the consideration.
These findings represent a "good to know while interacting with the project" information, but don't directly damage the project in its current state.

The SiriusHandler contract is a utility contract used to calculate user nToken share for the pool. A number of small issues are present, one of them is a missing zero-address validation (DFM-2), which depending on interfaced contracts implementation, may lead to the loss of functionality, however is highly unlikely. Some best practices could also be implemented.

Defimoon has only audited the SiriusHandler.sol contract and did not audit the contracts outside of that scope.

Summary of findings

According to the standard audit assessment, the audited solidity smart contract is well-secured. The contract has successfully passed the audit.

ID	Description	Severity
DFM-1	Use of unknown interfaces	Informational
DFM-2	Missing zero check	Low Risk

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	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 « Use of unknown interfaces»

Severity: Informational

Description:
SiriusHandler utilizes 3 interfaces, however the interfaced contracts are out of scope of this audit and their exact functioning in unknown. This is not considered a problem as the interfaced contracts are from a verified and trusted application and the use of those doesn't not damage Algem's reputation or functional directly.

Recommendation:
It is recommended to attach the audits (if present) of the interfaced contracts with this report, or to audit the interfaced contracts.

DFM-2 « Missing zero check»

Severity: Low Risk

Description:
Function `calc()` accepts `address _user` as an argument, however it is missing a zero check. Even though this is not a direct threat, depending on the interfaced functions implementation, this could lead to a loss of contract functionality.

Exploit Scenario:

1. If the `balanceOf()` function called via `lp` and `gauge` interfaces behaves unexpectedly when given zero (0x00) as an argument or without passing an argument, this could potentially cause issues with protocol functionality.

Recommendation:
It is recommended to check that the `address _user` in not zero.

Automated Analyses

Slither

Slither has reported 11 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.

Adherence to Best Practices

1. Code is lacking in-depth documentation. It is recommended to always utilize NatSpec commenting style to improve projects documentation.

Methodology

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

Closed	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