DEFIMOON

be secure

Smart Contract Audit Report

September, 2022

SolarDiamond

DEFIMOON PROJECT

Audit and
Development

CONTACTS

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

DEFIMOON

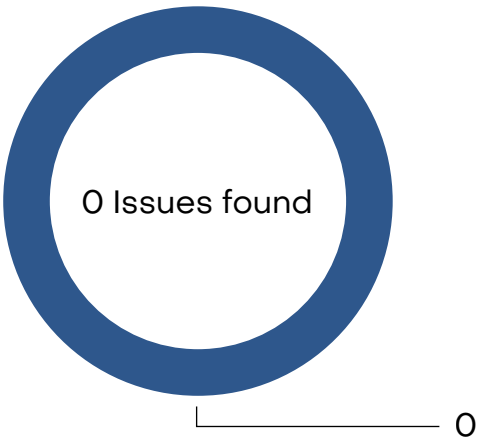
be secure

September 12th 2022

This audit report was prepared by Defimoon for SolarProtocol

Audit information

Description	Diamond paradigm implementation of different industry standards and token interaction tools.
Audited files	Contracts from SolarDiamond and VoidDiamond repository.
Timeline	6th September 2022 - 12th September 2022
Audited by	Daniil Rashin, Ilya Vaganov
Approved by	Artur Makhnach, Cyrill Minyaev
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Manual Review
Specification	Source code documentation
Docs quality	High
Source code	Commit from git repository
Network	Not specified
Status	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.

SolarDiamond Audit overview

Libraries overview

No major issues were found

Simple code to access such things as msg.sender.

Contract name	Status
LibContext.sol	Passed

Access module overview

No major issues were found

Access control implementation.

Contract name	Status
AccessControlFacet.sol	Passed
LibAccessControl.sol	Passed
LibRoles.sol	Passed

Blacklist module overview

No major issues were found

Functions to manage blacklist of addresses with easy access from anywhere.

Contract name	Status
LibSimpleBlacklist.sol	Passed
SimpleBlacklistFacet.sol	Passed

Diamond module overview

No major issues were found

Diamond implementation.

Contract name	Status
Diamond.sol	Passed
LibDiamondExtras.sol	Passed

Migratable module overview

No major issues were found

Migrations implementation.

Contract name	Status
LibMigratable.sol	Passed
MigratableFacet.sol	Passed
MigratableInit.sol	Passed
MigrationPlaceholderFacet.sol	Passed

Pausable module overview

No major issues were found

Pause/unpause functionality.

Contract name	Status
LibPausable.sol	Passed
PausableFacet.sol	Passed

Solo-token module overview

No major issues were found

ERC20 & ERC777 implementation and general token related code in library.

Contract name	Status
LibSoloToken.sol	Passed
ERC20Facet.sol	Passed
ERC777Facet.sol	Passed

Token-distributor module overview

No major issues were found

Token distributor with mutable strategies, also has uniswap options.

Contract name	Status
LibTokenDistributor.sol	Passed
LibTokenDistributorV1Migration.sol	Passed
TokenDitributorFacet.sol	Passed

Token taxes module overview

No major issues were found

Efficient way to distribute and control token taxes, enable/disable both globally and for specific addresses.

Contract name	Status
LibTokenTaxes.sol	Passed
TokenTaxesFacet.sol	Passed

Uniswap module overview

No major issues were found

Generally uniswap wrappers.

The intermediate wallet solution to overcome uniswap limitation is considered as safe, as this wallet receives and transfers tokens in the same transaction.

Contract name	Status
LibUniswap.sol	Passed
UniswapFacet.sol	Passed

VoidDiamond Audit overview

Migrations overview

No major issues were found

Contracts to migrate different parts of diamond. Nothing complicated, no dynamically calculated parameters, chosen stablecoin $DAI is also considered safe enough to be used in modern projects.

Contract name	Status
0001_setup_access_control.sol	Passed
0002_setup_initial_token.sol	Passed
0003_set_erc165_interfaces.sol	Passed
0004_setup_token_distributor.sol	Passed
0005_setup_token_taxes.sol	Passed
0006_setup_token_reflections.sol	Passed
0007_decay_launch_taxes_to_28 copy.sol	Passed
0008_decay_launch_taxes_to_25.sol	Passed
9999_localMigration.sol	Passed

Summary of findings

According to the standard audit assessment, the audited solidity smart contracts are secure and ready for production.

Overall code quality and documentation is at a really high level, though diamond paradigm is complicated in terms of faucet upgrading and admin access control.
It is also recommended to control the faucet versions accurately, to keep things more transparent. Best approach would be to encapsulate all the core modules into separate repository, so it can be reused when needed.

The most important thing when working with stablecoins is to keep up to date with the level of trust given to desired coin by user and of course developers. All the rules on this topic are well-known and based on the audit you are already familiar with them.

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

Automated Analyses

Slither

Unfortunately, diamond paradigm does not fit well with automated contract analysis tools. Though particular contracts can be analysed, Slither hangs for unknown time when passing the project directory.

Individual contract findings were marked as false-positive after additional checks.

Adherence to Best Practices

1. Please, stick to you current rules and practices — code styling, documentation and the rest of the project is a real pleasure to read and work with.
2. Pay attention to core functionality version control.
3. Refactor project dependencies to use less access controlled repos, so it is easier to share it.

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

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