defimoon.org
twitter.com/Defimoon_org
linkedin.com/company/defimoon

Rome, Italy, 00165

Smart contract’s
Audit report

Pluton

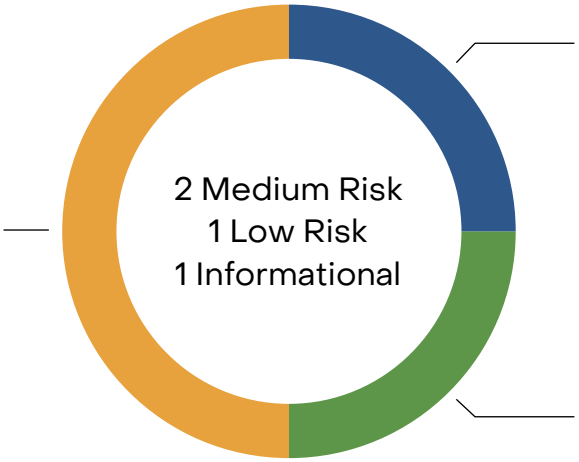
February, 2025

DEFIMOON
be secure

6 Feb 2025
This reaudit report was prepared by DefiMoon for Pluton.

Audit information

Description	Pluton bridge project smart contract
Audited files	Pluton.sol
Timeline	13 Jan 2025 - 6 Feb 2025
Approved by	Artur Makhnach, Kirill Minyaev
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Manual Review
Source code	https://github.com/pluton-bridge/pluton-evm-contract/blob/main/src/Pluton.sol
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.

Audit overview

Found issues have been acknowledged and do not pose a threat to protocol security

This document provides an audit of the Pluton smart contract. The primary purpose of this audit is to evaluate the contract for potential vulnerabilities, ensure adherence to best practices, and verify its correctness, security, and efficiency.

Scope

The scope of the audit includes the entirety of the Pluton smart contract provided, focusing on:

- Correctness of the implementation
- Security vulnerabilities
- Adherence to best practices in Solidity development
- Gas efficiency

Contract Details

- Contract Name: PlutonContract
- Compiler Version: Solidity 0.8.27
- External Dependencies:
    - OpenZeppelin Libraries
        - IERC20
        - SafeERC20
        - EIP712
        - ECDSA
        - Ownable
        - ReentrancyGuard

Summary of Findings

The contract demonstrates a good understanding of Solidity development principles and implements several security features, such as:

- Use of well-established OpenZeppelin libraries
- Protection against reentrancy attacks
- Efficient error handling with custom error types

However, some areas require attention to improve robustness and efficiency:

1. Signature verification may require additional safeguards to ensure integrity and prevent misuse.
2. Input validation is not comprehensive for critical functions, increasing the risk of misuse.
3. Gas efficiency can be optimized by restructuring the storage mappings.
4. Timestamp reliance for deadline checks introduces a minor susceptibility to miner manipulation.

The detailed findings, categorized by severity and type, are outlined in subsequent sections of this report.

Summary of findings

ID	Description	Severity	Status
DFM-1	Signature Verification Integrity	Medium Risk	Acknowledged
DFM-2	Input Validation	Medium Risk	Acknowledged
DFM-3	Gas Efficiency Optimization	Informational	Acknowledged
DFM-4	Timestamp Manipulation Risk	Low Risk	Acknowledged

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

Program version not specified	Passed
Program 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 loop	Passed
High consumption storage	Passed
“Out of Gas” Attack	Passed

Findings

DFM-1 « Signature Verification Integrity»

Severity: Medium Risk

Status: Acknowledged

Description: Signature validation assumes the owner's private key integrity and lacks additional mechanisms to mitigate misuse in case of compromise.

Recommendation: Implement multi-signature or additional layers of authentication for sensitive operations.

Client's comments:
The first problem was that the contract security would be at risk if the private key for the owner account was stolen or exposed, the first thing that should be noted is that for solvers that do not have trust in our security measures, we gave the ability to them to claim their money instantly after solving so they do not need to be worried whether or not we keep the private key safe and they can keep their claimed money for themselves so there wont be any money in the contract to be worried about

Also, it is good to point out that we know all the risks of having one wallet to be the owner of the contract, and that's why we have plan to use tss signature in the future and be fully decentralized so the minimum risk of exposing the private key be fully removed

DFM-2 «Input Validation»

Severity: Medium Risk

Status: Acknowledged

Description: Critical functions such as `claimSolver` and `refundUser` lack comprehensive validation of parameters like `amount` and `address`.

Recommendation: Include stricter checks for input parameters to prevent invalid or malicious data from being processed.

Client's comments:
For the second problem, we are strictly controlling everything in our protocol before giving the signature, so there is no need for further validation, and it would just make our contract claim and refund inefficient

DFM-3 « Gas Efficiency Optimization»

Severity: Informational

Status: Acknowledged

Description: Nested mappings for tracking claims can lead to higher gas costs during reads and writes.
Recommendation: Refactor the storage structure to optimize gas usage, possibly by consolidating data mappings.

Client's comments:
We actually thought about the third informational statement, and our first contract had only one nonce for the whole contract so that the signature wouldn't be misused, but we figured out that when the number of solvers increases not only we need one nonce for each solver, but also nonce for each token of solver so the reverted transactions would be less in high congested networks

DFM-4 « Timestamp Manipulation Risk»

Severity: Low Risk

Status: Acknowledged

Description: Reliance on `block.timestamp` for expiration checks introduces susceptibility to minor miner manipulation.

Recommendation: Use block numbers or add a buffer period to mitigate risks of timestamp manipulation.

Client's comments:
For the last one, we used timestamp because it is easier for readability, and also, it is easier for protocol to keep all the other chains unified

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