DEFIMOON

be secure

# Smart Contract Audit Report

September, 2022

BERLINER

CONTACTS


DEFIMOON PROJECT

Audit and Development

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

DEFIMOON

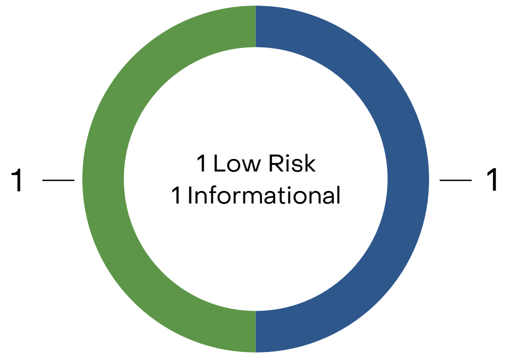
be secure

September 12th 2022

This audit report was prepared by Defimoon for Berliner

## Audit information

| | |
|---|---|
| Description | The contract implements the Initial ERC20 token Initial Offering |
| Project website | https://www.berliner.vip/ |
| Audited files | Coin.sol |
| Timeline | 12th September |
| Audited by | Cyrill Novoseletskyi |
| Approved by | Artur Makhnach, Cyrill Minyaev |
| Languages | Solidity |
| Methods | Architecture Review, Unit Testing, Functional Testing, Manual Review |
| Specification | Documentation |
| Docs quality | High |
| Source code | https://bscscan.com/address/0x411c9119a976c238fd4b172b9dda11d7addb10c7#code |
| Network | Binance smart chain |
| 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

**No Major security issues were found.**

Descriptions of functions do not comply with generally accepted standard called NatSpec, which impairs readability and understanding of the code (DFM-3).

In the `updateCooldown` function the passed `_time` value does not change the `coolDownTime` variable (DFM-4).

## Summary of findings

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

| ID | Description | Severity | Status |
|---|---|---|---|
| DFM-1 | Unlimited transferFrom | High Risk | Resolved |
| DFM-2 | Use of unsafe transfer function | Medium Risk | Resolved |
| DFM-3 | Lack of input parameter validation | Low Risk | Resolved |
| DFM-4 | The value does not change | Low Risk | Acknowledge |
| DFM-5 | Redundant timestamp multiplier | Informational | Resolved |
| DFM-6 | Non-adherence to the "NatSpec" comment format | Informational | Acknowledged |

## Launchpad Check list

| Description | Status |
|---|---|
| No mint function found, owner cannot mint tokens after initial deploy | ✅ |
| Owner can't set max tx amount [1] | ❌ |
| Owner can't set fees over 25% | ✅ |
| Owner can't pause trading | ✅ |
| Owner can't blacklist wallets | ✅ |

[1]
```
function updateMaxTxLimit(uint256 maxBuy, uint256 maxSell,
uint256 maxWallet) external onlyOwner {
        require(maxBuy >= 1_000, "Cannot set max buy amount lower
than 0.1%");
        require(maxSell >= 1_000, "Cannot set max sell amount
lower than 0.1%");
        require(maxWallet >= 10_000, "Cannot set max wallet amount
lower than 1%");
        maxBuyLimit = maxBuy * 10**decimals();
        maxSellLimit = maxSell * 10**decimals();
        maxWalletLimit = maxWallet * 10**decimals(); }
```

## 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 « Unlimited transferFrom»

Severity: High Risk

Status: Resolved

Description:
Function transferFrom describes the logic in which the `_transfer` method is executed first and only after that there is a check on how many tokens the user can spend. This can lead to unlimited spending by users from other users' wallets.

Recommendations:
Move `require(currentAllowance >= amount, "BEP20: transfer amount exceeds allowance”);` to the very beginning of the function and only then execute the transfer logic.

### DFM-2 «Use of unsafe transfer function»

Severity: Medium Risk

Status: Resolved

Description:
Classic `transfer` mechanism of sending native tokens used in the `Liquify` function is not secure.

Recommendations:
It is recommended to use the method from the connected library - `sendValue`.

## DFM-3 «Lack of input parameter validation»

Severity: Low Risk

Status: Resolved

Description:

When reassigning a router in the `updateRouterAndPair` function, it is necessary to pass the address of the contract, but this is not checked in any way, which can lead to the user passing any address which will lead to errors and gas waste.

Recommendations:

It is better to use the method from the connected library - `isContract`.

## DFM-4 «The value does not change»

Severity: Low Risk

Status: Acknowledged

Description:

in the `updateCooldown` function, passing the `_time` value, it is not assigned to the `coolDownTime` variable, which makes calling the current method useless.

Recommendations:

Add assignment of a new value to the `coolDownTime` variable.

DFM-5 «Redundant timestamp multiplier»

Severity: Informational

Status: Resolved

Description:
Transmitted timestamp in the `updateCooldown` function does not need to be multiplied by `1 seconds` since it is assumed that the timestamp itself is the number of seconds.

Recommendation:
Remove multiplication of `time` by `1 seconds`.

DFM-6 «Non-adherence to the "NatSpec" comment format»

Severity: Informational

Status: Acknowledged

Description: Documentation and commenting in the current contract is not standardized. It is not informative enough, which makes the code difficult to read. Most importantly, it also don't follow the semantic rules required for the web3 applications (blockchain explorers) to process contracts properly.

Recommendation:
It is recommended at least to add attributes in comments such as "@notice", "@param".
You can read about it here.

## Automated Analyses

### Slither

Slither has reported 56 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.

# 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 |