DEFIMOON

be secure

Smart Contract Audit Report

April, 2023

Rewardz® Network

DEFIMOON PROJECT

Audit and
Development

CONTACTS

defimoon.org
audit@defimoon.org
defimoon_org
defimoonorg
defimoon
defimoonorg

DEFIMOON

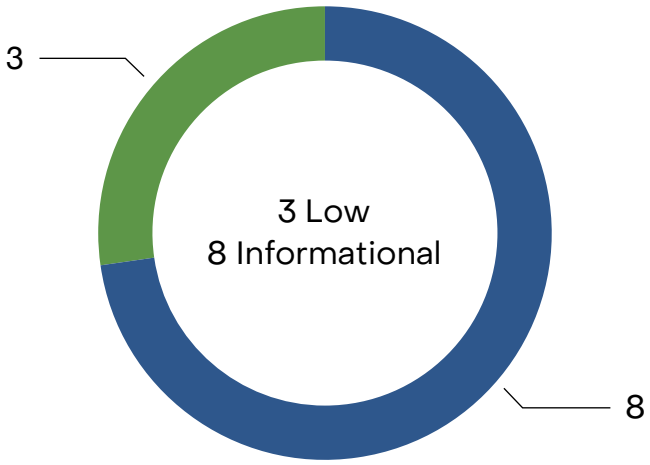
be secure

12 May 2023

This audit report was prepared by DefiMoon for RAYN.

Audit information

Description	ERC20 token with advanced trading functionality and liquidity management.
Audited files	Rewardz.sol
Timeline	10 May 2023 - 12 May 2023
Audited by	Ilya Vaganov
Approved by	Artur Makhnach, Kirill Minyaev
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Manual Review
Source code	https://github.com/rewardz-network/rayn-token/tree/e472892132682ea46c78fbfb9098846c9a5b2065
Chain	Ethereum
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

Major issues were not found.

Summary of findings

ID	Description	Severity
DFM-1	Function not working	Low Risk
DFM-2	No success check	Low Risk
DFM-3	Possible loss of ownership	Low Risk
DFM-4	Extra type conversion	Informational
DFM-5	Extra ownership transfer	Informational
DFM-6	Percentages do not change values dynamically	Informational
DFM-7	Additional permissions for the router	Informational
DFM-8	Change by parameter regardless of function	Informational
DFM-9	Gas optimization	Informational
DFM-10	Incorrect type conversion	Informational
DFM-11	Different logic for burn addresses	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	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 «Function not working»

Severity: Low Risk

Description: The returnToNormalTax function assumes that taxes are changed back to normal values, but it contains a require that will always fail.

```
function returnToNormalTax() external onlyOwner {
    sellOperationsFee = 100;
    sellLiquidityFee = 0;
    sellDevFee = 0;
    sellBurnFee = 0;
    sellTotalFees = sellOperationsFee + sellLiquidityFee + sellDevFee + sellBurnFee;
    require(sellTotalFees <= 30, "Must keep fees at 30% or less!");

    buyOperationsFee = 300;
    buyLiquidityFee = 0;
    buyDevFee = 0;
    buyBurnFee = 0;
    buyTotalFees = buyOperationsFee + buyLiquidityFee + buyDevFee + buyBurnFee;
    require(buyTotalFees <= 30, "Must keep fees at 30% or less!");
}
```

The total fees will always exceed the maximum allowable value of 30.

Recommendation: Fix the function by changing the values for fees or the limit for require.

DFM-2 «No success check»

Severity: Low Risk

Description: The swapBack, withdrawStuckETH functions use a call to transfer ETH, but the success result is not checked.

```
(success,) = address(devAddress).call{value: ethForDev}("");
(success,) = address(operationsAddress).call{value: address(this).balance}("");
```

```
bool success;
(success,) = address(msg.sender).call{value: address(this).balance}("");
```

Similarly, there is no check for _sent in the transferForeignToken function.

```
_sent = IERC20(_token).transfer(_to, _contractBalance);
```

As a result, if the calls fail, the execution of the function will not be interrupted.

Recommendation: The best practice is to add a require check for variables that characterize the success of the call.

DFM-3 «Possible loss of ownership»

Severity: Low Risk

Description: The inherited contract Ownable contains a renounceOwnership function that resets the owner of the contract.

Recommendation: To avoid accidental loss of ownership of the contract, it is better to remove this function or override:

```
function renounceOwnership() external override onlyOwner {
    revert("Not allowed");
}
```

DFM-4 «Extra type conversion»

Severity: Informational

Description: The lpPair variable is already of type address, no need to convert to address.

```
_excludeFromMaxTransaction(address(lpPair), true);
_setAutomatedMarketMakerPair(address(lpPair), true);
```

Recommendation: Remove conversion to address.

DFM-5 «Extra ownership transfer»

Severity: Informational

Description: The inherited contract Ownable already defines msg.sender as the owner. Re-identification of the owner is not necessary.

```
transferOwnership(newOwner);
```

Recommendation: Remove re-identification of the owner.

DFM-6 «Percentages do not change values dynamically»

Severity: Informational

Description: The upper bound for swapTokensAtAmount has limits based on totalSupply, but totalSupply can change over time as tokens are burned.

```
require(newAmount <= totalSupply() * 1 / 1000, "Swap amount cannot be higher than 0.1% total supply!");
swapTokensAtAmount = newAmount;
```

Recommendation: Please make sure that the code matches the intended logic. If the value of swapTokensAtAmount still need to change relative to totalSupply dynamically, then fix it.

DFM-7 «Additional permissions for the router»

Severity: Informational

Description: For a liquid pair, the _excludeFromMaxTransaction and _setAutomatedMarketMakerPair permissions are set, but calls also go through the router.

```
_excludeFromMaxTransaction(address(lpPair), true);
_setAutomatedMarketMakerPair(address(lpPair), true);
```

Recommendation: It would be better to add a permission for the router too.

DFM-8 «Change by parameter regardless of function»

Severity: Informational

Description: In the swapBack function, the reset of the tokensForBurn variable occurs regardless of the burning of tokens.

```
if (tokensForBurn > 0 && balanceOf(address(this)) >= tokensForBurn) {
    _burn(address(this), tokensForBurn);
}
tokensForBurn = 0;
```

As a result, the tokensForBurn parameter may always be set to zero and the burn will never be performed.

Recommendation: Make sure the implementation matches your intent and fix it if necessary.

DFM-9 «Gas optimization»

Severity: Informational

Description: The raynDrop function contains a loop with a potentially large number of iterations, which can cost a lot of gas.

Recommendation: The best gas optimization solution would be to bring the function to this format:

```
function raynDrop(address[] calldata wallets, uint256[] calldata amountsInTokens) external
onlyOwner {
    uint256 l = wallets.length;
    require(l == amountsInTokens.length, "Arrays must be the same length");
    require(l < 600, "Can only airdrop 600 wallets per txn due to gas limits");

    address sender = msg.sender;
    for (uint256 i; i < l; ) {
        super._transfer(sender, wallets[i], amountsInTokens[i]);
        unchecked { ++i; }
    }
}
```

DFM-10 «Incorrect type conversion»

Severity: Informational

Description: In the setOperationsAddress and setDevAddress functions, the variables are given the address payable data type, but the variables are declared simply as address.

```
operationsAddress = payable(_operationsAddress);

devAddress = payable(_devAddress);
```

Recommendation: Remove incorrect conversion to payable. You can use local conversion where necessary.

DFM-11 «Different logic for burn addresses»

Severity: Informational

Description: Some functions and conditions use the dead address to check the burning address, and some use both the dead address and the zero address, although both of these addresses perform the same function.

Recommendation: Please make sure the implementation is as intended and fix it if necessary. It is also worth remembering that using zero address is more popular than using dead address, so it is not advisable to ignore zero address in smart contract logic.

Automated Analyses

Slither

Slither's automatic analysis not found vulnerabilities, or these false positives results .

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