DEFIMOON

be secure

# Smart Contract Audit Report

January, 2024


DEFIMOON PROJECT

Audit and
Development

CONTACTS

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

DEFIMOON

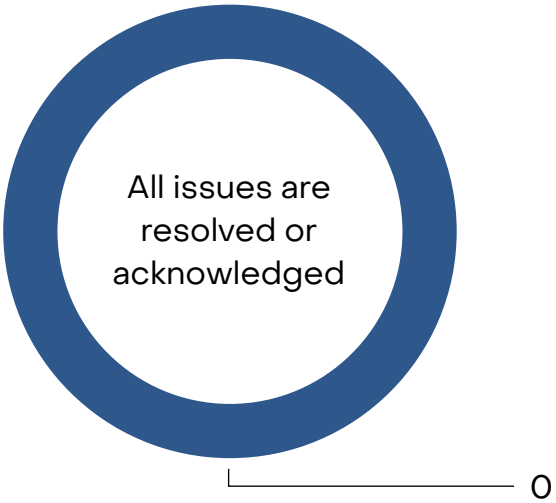
be secure

11 Jan 2024

This audit report was prepared by DefiMoon for DOTins.

## Audit information

| | |
|---|---|
| Description | Omni-Inscriptions Marketplace |
| Timeline | 8-11 Jan 2024 |
| Approved by | Artur Makhnach, Kirill Minyaev |
| Audit Scope | Dotins.sol |
| Languages | Solidity |
| Methods | Architecture Review, Unit Testing, Functional Testing, Manual Review |
| Project Site | https://www.dotins.io/ \|. https://dotins.gitbook.io/dotins/intro |
| Source code | https://gist.github.com/donmoonix/082912ce4a10741eeeabc69409be88b5 |
| Reaudit Source code | https://gist.github.com/donmoonix/a456f2e4b2e0078a9bc731072a0d3227 |
| Network | EVM-like |
| 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 vulnerability has been found.**

The contract contains one High Risk vulnerability and several fairly non-serious vulnerabilities.

We recommend recommend fixing vulnerabilities, adding invents, checking the code logic, developing detailed unit tests with full coverage, and performing manual testing considering a wide variety of scenarios.

## Reaudit overview

**All major vulnerabilities resolved.**

## Summary of findings

| ID | Description | Severity | Status |
|---|---|---|---|
| DFM-1 | Mint doesn't increase total supply | High Risk | Resolved |
| DFM-2 | Mint domains with the same names | Low Risk | Acknowledged |
| DFM-3 | The number of characters in the domain is not calculated correctly | Low Risk | Acknowledged |
| DFM-4 | Excess payments are not returned to the sender | Low Risk | Acknowledged |
| DFM-5 | The value of the variable does not change | Low Risk | Resolved |
| DFM-6 | Gas optimization | Information | Partially Resolved |
| DFM-7 | Safer redeployment | Information | Acknowledged |
| DFM-8 | Additional events | Information | 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

| | |
|---|---|
| Solidity version not specified | Passed |
| Solidity version too old | Passed |
| Integer overflow/underflow | Passed |
| Function input parameters lack of check | Not Passed |
| Function input parameters check bypass | Passed |
| Function access control lacks management | Passed |
| Critical operation lacks event log | Passed |
| Human/contract checks bypass | Not 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 «Mint doesn't increase total supply»

**Severity:** High Risk

**Status:** Resolved

**Description:** Calling the mintToken function does not increment totalSupply, even though the function implies that it should be changed. Thus, it is possible to bypass the maxSupply limitation.

**Recommendation:** Add an increase in totalSupply by amount like this:

```
tokenTickers[encodedTicker].totalSupply += amount;
```

## DFM-2 «Mint domains with the same names»

**Severity:** Low Risk

**Status:** Acknowledged

**Description:** In the mintDomain function, multiple domains with the same name can be minted either within different Top Level Domains or within a single Top Level Domain. In this case, if you mint domains with the same domainName within one Top Level Domain, then totalSupply will increase, despite the fact that the actual number of domains does not increase.

**Recommendation:** We recommend banning mint domains that already exist like this:

```
mapping(string => mapping(string => bool)) public existingDomains;

// ...

function calculateMintDomainCost(
    string memory domainName,
    string memory topLevelDomain
) public view returns (uint256) {
      require(!existingDomains[topLevelDomain][domainName], "Domain already
exists");

      // ...
}

function mintDomain(
    string memory domainName,
    string memory topLevelDomain
) external payable whenNotPaused {
      // ...

      uint256 totalMintFee = calculateMintDomainCost(domainName,
topLevelDomain);
      existingDomains[topLevelDomain][domainName] = true;

      //...
}
```

## DFM-3 «The number of characters in the domain is not calculated correctly»

**Severity:** Low Risk

**Status:** Acknowledged

**Description:** The calculateMintDomainCost function uses the bytes(domainName).length expression to calculate the number of characters in domainName, but it is not valid for all character types. Not all characters are encoded strictly in one byte, for example the character £ takes 2 bytes (0xc2a3). Thus, it is possible to bypass the minimum character limit or to achieve a smaller lengthMultiplier value with fewer characters.

**Recommendation:** We recommend that you keep this character encoding feature in mind and change the function appropriately if necessary.

## DFM-4 «Excess payments are not returned to the sender»

**Severity:** Low Risk

**Status:** Acknowledged

**Description:** The deployToken, mintToken and mintDomain functions accept msg.value to pay the internal fee, but in case msg.value exceeds the internal fee the sender is not returned the excess and revert is not called. In this case, users may mistakenly send more funds than necessary, resulting in losing some of their funds.

**Recommendation:** We recommend that you disallow sending msg.value that exceeds the required amount with fee:

```
require(msg.value == totalMintFee, "DotINS: Invalid value");
```

or return the excess funds back to sender (at the end of the function):

```
if (msg.value > totalMintFee) {
      (bool success, ) = payable(msg.sender).call{value: totalMintFee -
msg.value}("");
    require(success, "DotINS: Transfer failed");
} else {
      require(msg.value == totalMintFee, "DotINS: Insufficient domain mint
fee");
}
```

## DFM-5 «The value of the variable does not change»

**Severity:** Low Risk

**Status:** Resolved

**Description:** The redeploymentComplete variable is initiated as false and is not changed anywhere else, but the adminRedeployTokenFromV1 function checks the value of the variable. In the current implementation, the expression !redeploymentComplete will always be true.

**Recommendation:** Make sure that the contract contains all the necessary logic and functions to handle the redeploymentComplete variable and add them if necessary.

## DFM-6 «Gas optimization»

**Severity:** Informational

**Status:** Partially Resolved

**Description:** We recommend not duplicating functionality and utilizing existing functions and variables to save gas and reduce the size of the contract bytecode.

```
function release(address payable account) public {
    uint256 amount = creatorsPendingBalances[account];
    require(amount > 0, "DotINS: No earnings to withdraw");

    totalPendingBalances -= amount;
    creatorsPendingBalances[account] = 0;

    (bool success, ) = account.call{value: amount}("");
    require(success, "DotINS: Transfer failed");
}
```

---

```
function mintToken(string memory ticker, uint256 amount) external payable
whenNotPaused {
    bytes32 encodedTicker = keccak256(abi.encodePacked(ticker));

    require(
        tokenTickers[encodedTicker].totalSupply + amount <=
tokenTickers[encodedTicker].maxSupply,
        "DotINS: Token max supply reached"
    );

    uint256 totalCost = calculateMintTokenCost(ticker, amount);
    require(msg.value >= totalCost, "DotINS: Insufficient mint payment");

    uint256 tempCost = totalCost - protocolTokenMintFee;
    creatorsPendingBalances[tokenTickers[encodedTicker].creator] += tempCost;
    totalPendingBalances += tempCost;

    string memory output = string.concat(
        "data:,",
        '{"p":"moon-20","op":"mint","tick":"',
        ticker,
        '","amt":"',
        Strings.toString(amount),
        '"}'
    );
    emit dotins_protocol_CreateInscriptionV1(msg.sender, output);
}
```

---

```
function mintDomain(
    string memory domainName,
    string memory topLevelDomain
) external payable whenNotPaused {
    TopLevelDomain storage tld = topLevelDomains[topLevelDomain];

      uint256 totalMintFee = calculateMintDomainCost(domainName,
topLevelDomain);
    require(msg.value >= totalMintFee, "DotINS: Insufficient domain mint fee");

    string memory output = string.concat("data:,", domainName, ".",
topLevelDomain);

    if (tld.creator != address(0)) {
        uint256 tempCost = totalMintFee - protocolDomainMintFee;
        creatorsPendingBalances[tld.creator] += tempCost;
        totalPendingBalances += tempCost;
    }
    tld.totalSupply += 1;

    emit dotins_protocol_CreateInscriptionV1(msg.sender, output);
}
```

## DFM-7 «Safer redeployment»

**Severity:** Informational

**Status:** Acknowledged

**Description:** If the adminRedeployTokenFromV1 function is used to migrate from an older version of a contract, we recommend getting the data directly from the old contract rather than through function arguments to avoid making mistakes. Like this:

```
interface IOldDotins {
    struct TokenTick {
        uint256 maxSupply;
        uint256 totalSupply;
        uint256 mintPrice;
        uint256 mintLimit;
        address creator;
    }

      function tokenTickers(bytes32) external view returns (TokenTick memory);
}

// ...

function adminRedeployTokenFromV1(
    string memory ticker
) external onlyRole(MANAGER_ROLE) {
      IOldDotins oldContract = IOldDotins(OLD_CONTRACT_ADDR);
    bytes32 encodedTicker = keccak256(abi.encodePacked(ticker));

    require(tokenTickers[encodedTicker].creator == address(0), "DotINS: Ticker
already exists");
    require(!redeploymentComplete, "DotINS: Finished redeployment");

    tokenTickers[encodedTicker] = oldContract.tokenTickers(encodedTicker);
}
```

## DFM-8 «Additional events»

**Severity:** Informational

**Status:** Acknowledged

**Description:** We recommend adding indexed field events to all core functions. Events may come in handy in the future when collecting statistics, automating and integrating contracts with UI.

## 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 |
| Protocol's Design | Assumed by the protocol design as a necessary functionality that will work properly within this application |
| Acknowledged | Project team is made aware of the finding |
| Open | The finding was not addressed |
| Not Actual | Not relevant after protocol logic changes |