SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Modulus Domain

AUDIT
SECURITY ASSESSMENT

13. July, 2023

FOR

M MODS.
Modulus Domain Service

SOLIDProof

Introduction

SolidProof.io is a brand of the officially registered company MAKE Network GmbH, based in Germany. We're mainly focused on Blockchain Security such as Smart Contract Audits and KYC verification for project teams. Solidproof.io assess potential security issues in the smart contracts implementations, review for potential inconsistencies between the code base and the whitepaper/documentation, and provide suggestions for improvement.

Disclaimer

SolidProof.io reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof's position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of the security or functionality of the technology we agree to analyze.

Project Overview

Summary

Project Name	**Modulus Domain**
Website	https://modulus.domains/
About the project	MODS allows users to create custom names that are easier to remember and type than traditional cryptocurrency addresses. This makes it more accessible for non-technical users and provides a human-friendly interface to interact with the blockchain.
Chain	Ethereum
Language	Solidity
Codebase Link	Registry contract - 0x13ec1d5b301fe429001B51Dfa6943Ab4e3ba5Fed NFT contract - 0x530854dC46D498F8e82a2f6F1d12040E07eb89C0
Commit	N/A
Unit Tests	Not Provided

Social Medias

Telegram	https://t.me/modulusDSportal
Twitter	https://twitter.com/ModulusDomains
Medium	https://medium.com/@modulusdomains
Instagram	N/A
Github	N/A
Reddit	N/A
LinkedIn	N/A
Discord	N/A
Youtube	N/A
TikTok	N/A

Audit Summary

Version	Delivery Date	Changelog
v1.0	13. July 2023	• Layout Project • Automated- /Manual-Security Testing • Summary

Note - This Audit report consists of a security analysis of the **Modulus Domain** smart contracts. This analysis did not include functional testing (or unit testing) of the contract's logic.
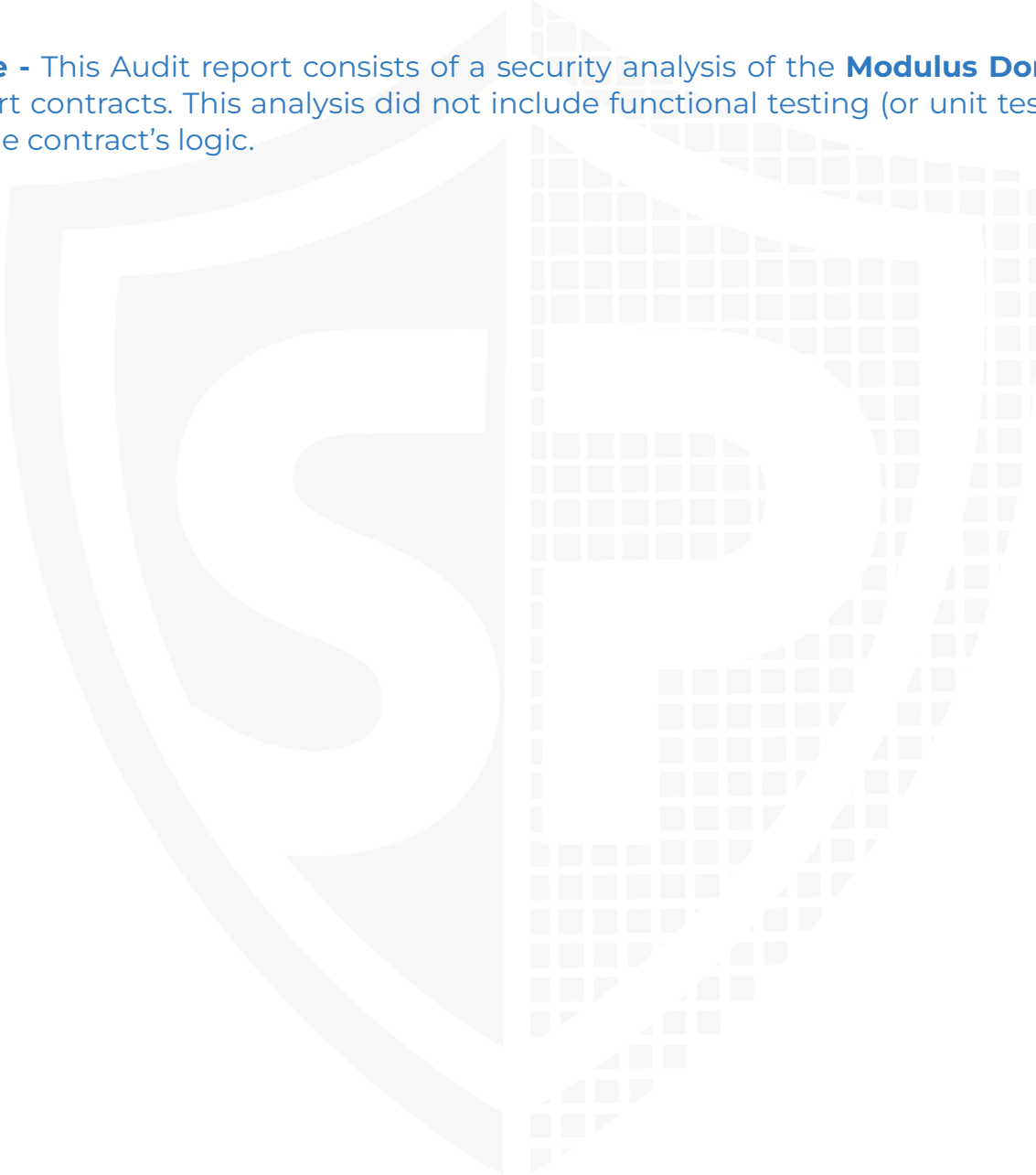
File Overview

The Team provided us with the files that should be tested in the security assessment. This audit covered the following files listed below with an SHA-1 Hash.

File Name	SHA-1 Hash
contracts/MODSMNS.sol	8d110ba92b3e7953e7811b0794580874ec44ee81
contracts/Registry.sol	c0d331e5f9d7cfba35fd7a970af008b58cabc0a2

Please note: Files with a different hash value than in this table have been modified after the security check, either intentionally or unintentionally. A different hash value may (but need not) be an indication of a changed state or potential vulnerability that was not the subject of this scan.

Imported packages

Used code from other Frameworks/Smart Contracts (direct imports).

Dependency / Import Path	Count
@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	2
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	2
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	2
@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol	1
@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol	1
@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721EnumerableUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/utils/CountersUpgradeable.sol	1
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol	1

Note for Investors: We only Audited a Registry and NFT contract for **Modulus Domain**. However, If the project has other contracts (for example, a Presale, ERC20 contract etc), and they were not provided to us in the audit scope, then we cannot comment on its security and are not responsible for it in any way.

External/Public functions

External/public functions are functions that can be called from outside of a contract, i.e., they can be accessed by other contracts or external accounts on the blockchain. These functions are specified using the function declaration's external or public visibility modifier.

State variables

State variables are variables that are stored on the blockchain as part of the contract's state. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be defined with a visibility modifier, such as public, private, or internal, which determines the access level of the variable.

Components

📝 Contracts	📚 Libraries	🔍 Interfaces	🎨 Abstract
3	0	0	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐 Public	💰 Payable
29	3

External	Internal	Private	Pure	View
4	29	0	1	11

StateVariables

Total	🌐 Public
17	14

Capabilities

Solidity Versions observed	📤 Transfers ETH	💰 Can Receive Funds	🔡 Uses Hash Functions	💣 Has Destroyable Contracts
`^0.8.11` `0.8.11`	Yes	Yes	Yes	— — — —

Inheritance Graph

An inheritance graph is a graphical representation of the inheritance hierarchy among contracts. In object-oriented programming, inheritance is a mechanism that allows one class (or contract, in the case of Solidity) to inherit properties and methods from another class. It shows the relationships between different contracts and how they are related to each other through inheritance.
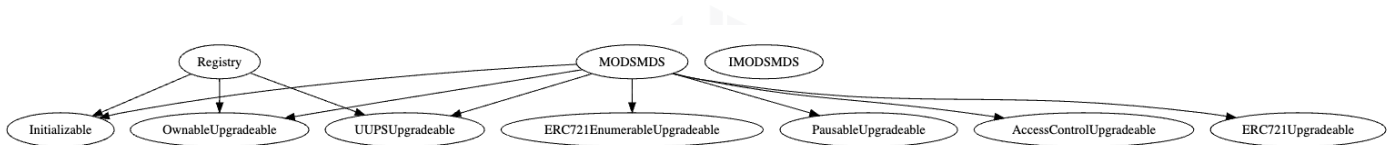
Audit Information

Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit vulnerability and the impact of that event on the organization or system. The risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon aspossible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to check the repository for security-related issues, code quality, and compliance with specifications and best practices. To this end, our team of experienced pen-testers and smart contract developers reviewed the code line by line and documented any issues discovered.

We check every file manually. We use automated tools only so that they help us achieve faster and better results.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
    a. Reviewing the specifications, sources, and instructions provided to
        SolidProof to ensure we understand the size, scope, and functionality of the
        smart contract.
    b. Manual review of the code, i.e., reading the source code line by line to identify potential vulnerabilities.
    c. Comparison to the specification, i.e., verifying that the code does what is described in the specifications, sources, and instructions provided to SolidProof.

2. Testing and automated analysis that includes the following:
    a. Test coverage analysis determines whether test cases cover code and how much code is executed when those test cases are executed.
    b. Symbolic execution, which is analysing a program to determine what inputs cause each part of a program to execute.

3. Review best practices, i.e., review smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on best practices, recommendations, and research from industry and academia.

4. Concrete, itemized and actionable recommendations to help you secure your smart contracts.

Overall Security
Medium or higher issues

Crucial issues found	❌ Contract is not safe to deploy
Description	The contract does contain issues of high or medium criticality. This means that known vulnerabilities were found in the source code and should be fixed asap.
Comment	N/A

Overall Security
Medium or higher issues

Upgradeability

Contract is an upgradeable	❌ Deployer can update the contract with new functionalities
Description	The deployer can replace the old contract with a new one with new features. Be aware of this, because the owner can add new features that may have a negative impact on your investments.
Comment	N/A

Ownership

The ownership is not renounced	✖ **The owner is not renounce**
Description	The owner has not renounced the ownership that means that the owner retains control over the contract's operations, including the ability to execute functions that may impact the contract's users or stakeholders. This can lead to several potential issues, including: • Centralizations • The owner has significant control over contract's operations
Comment	N/A

Note - If the contract is not deployed then we would consider the ownership to be not renounced. Moreover, if there are no ownership functionalities then the ownership is automatically considered renounced.

Ownership Privileges

These functions can be dangerous. Please note that abuse can lead to financial loss. We have a guide where you can learn more about these Functions.

Minting tokens

Minting tokens refer to the process of creating new tokens in a cryptocurrency or blockchain network. This process is typically performed by the project's owner or designated authority, who has the ability to add new tokens to the network's total supply.

Minter Roles addresses can mint new tokens	❌ The Minter Address is able to mint new tokens
Description	The wallets with minter roles who have the ability to mint new NFTs can reward themselves or other stakeholders, who can then sell the newly minted NFTs to raise funds. However, there is a risk that the minter may abuse this power, leading to a decrease in trust and credibility in the project or platform. If stakeholders perceive that the minter is using their power to mint new tokens unfairly or without transparency, it can result in decreased demand for the token and a reduction in its value.
Example	The minter role wallet can mint unlimited NFTs which may affect the price of other NFTs as well because it may disrupt the supply and demand chain.
Comment	We recommend to put a hard cap on the amount of tokens, the owner can mint

Line/s: L86
Codebase -

```
86      function mintTokens(address _to↑)
87          public
88          whenNotPaused
89          onlyRole(MINTER_ROLE)
90      {
91          uint256 tokenId = _tokenIdCounter.current();
92          _tokenIdCounter.increment();
93          _safeMint(_to↑, tokenId);
94      }
95
```

MODSMNS.sol

Burning tokens

Burning tokens is the process of permanently destroying a certain number of tokens, reducing the total supply of a cryptocurrency or token. This is usually done to increase the value of the remaining tokens, as the reduced supply can create scarcity and potentially drive up demand.

Contract owner cannot burn tokens	✅ The owner cannot burn tokens
Description	The owner is not able burn tokens without any allowances.
Comment	N/A

Blacklist addresses

Blacklisting addresses in smart contracts is the process of adding a certain address to a blacklist, effectively preventing them from accessing or participating in certain functionalities or transactions within the contract. This can be useful in preventing fraudulent or malicious activities, such as hacking attempts or money laundering.

Contract owner cannot blacklist addresses	✅ The owner cannot blacklist addresses
Description	The owner is not able blacklist addresses to lock funds.
Comment	N/A

Fees and Tax

In some smart contracts, the owner or creator of the contract can set fees for certain actions or operations within the contract. These fees can be used to cover the cost of running the contract, such as paying for gas fees or compensating the contract's owner for their time and effort in developing and maintaining the contract.

Contract owner cannot set fees more than 25%	✅ The owner cannot levy unfair taxes
Description	The owner is not able to set the fees above 25%
Comment	N/A

Lock User NFTs

In a smart contract, locking refers to the process of restricting access to certain tokens or assets for a specified period of time. When tokens or assets are locked in a smart contract, they cannot be transferred or used until the lock-up period has expired or certain conditions have been met.

Pauser wallet can lock the user funds	❌ The pauser wallet is able to lock the contract
Description	Locking the contract means that the owner is able to lock any funds of addresses that they are not able to transfer bought tokens anymore.
Example	An example of locking is by pausing the contract. In this case no Holder will be able to transfer their NFTs
Comment	N/A

Centralization Privileges

Centralization can arise when one or more parties have privileged access or control over the contract's functionality, data, or decision-making. This can occur, for example, if the contract is controlled by a single entity or if certain participants have special permissions or abilities that others do not.

In the project, there are authorities that have access to the following functions:

File	Privileges
1. Registry.sol	❖ onlyOwner – Update Dislike Symbol – Add/Remove addresses from reserve domains – Update the discount percentage of MODS holders to any arbitrary value – Update MODS NFT and ERC20 Token addresses – Update USDC Token address – Update the price of domain to any arbitrary value – Owner can buy reserved domains that are not already registered. But there are no limitations on how many unique domains owner can buy without paying anything. – Withdraw all the funds from the contract that were generated during the sale
2. MODSMNS.sol	❖ onlyPauser - The address with this role can pause/unpause the transfer of NFTs

Recommendations

To avoid potential hacking risks, it is advisable for the client to manage the private key of the privileged account with care. Additionally, we recommend enhancing the security practices of centralized privileges or roles in the protocol through a decentralized mechanism or smart-contract-based accounts, such as multi-signature wallets.

Here are some suggestions of what the client can do:

- Consider using multi-signature wallets: Multi-signature wallets require multiple parties to sign off on a transaction before it can be executed, providing an extra layer of security e.g. Gnosis Safe
- Use of a timelock at least with a latency of e.g. 48-72 hours for awareness of privileged operations
- Introduce a DAO/Governance/Voting module to increase transparency and user involvement

- Consider Renouncing the ownership so that the owner cannot modify any state variables of the contract anymore. Make sure to set up everything before renouncing.

Audit Results

#1 | Minter Wallet can mint unlimited tokens

File	Severity	Location	Status
MODSMNS.sol	Medium	L86	Open

Description - The Minter Address can mint unlimited NFTs without any limitation.

Remediation - We recommend putting a hard cap on the maximum amount the Minter role can mint.

#2 | Pauser wallet can lock tokens

File	Severity	Location	Status
MODSMNS.sol	Medium	L73, 77	Open

Description - The pauser wallet can pause the transfer functionality in the contract and then no user will be able to transfer their tokens

#3 | Floating Pragma

File	Severity	Location	Status
MODSMNS.sol	Informational	L2	Open

Description

- The current pragma Solidity directive is "^0.8.11". Contracts should be deployed with the same compiler version and flag that they have been tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using other versions.

Legend for the Issue Status

Attribute or Symbol	Meaning
Open	The issue is not fixed by the project team.
Fixed	The issue is fixed by the project team.
Acknowledged(ACK)	The issue has been acknowledged or declared as part of business logic.

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC Development | Marketing