

Security Audit Report



NovaDAO

28th – June 2022

ISSUED BY

LAPITS TECHNOLOGIES

Blockchain Technology Company

Disclaimer

The following document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation. This report can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities have been fixed.

Document

Name	Smart Contract Security Audit For NovaDAO
Conducted By	Akshat Mishra – Smart Contract Auditor At Lapits
Type	ERC – 721 Token
Platform	Ethereum Mainnet
Methods	Architecture Review, Functional Testing, Manual Review, Automated Testing
Website	https://novadao.xyz/
Timeline	1 Week
Tools	Slither , Mythril



Table of Contents

Introduction	4
Scope	4
Vulnerability Indicators	5
Checked Items	6
Project Overview	7
Test Coverage	8
Vulnerability Report	9 - 11
Disclaimers	12



Introduction

Lapits was contacted by Raghav – A math and economics major from UCLA, for the development, review and testing of a smart contract. He has founded NovaDAO – A project that takes venture DAOs several steps forward and enables people with small capital, to invest in projects and ideas in their early stages.

The project leverages the powers of web3; which lie in trustless tokenization and funds transfer, to enable any non-whale user to invest in projects, and get profits from their staked investments; making them stakeholders in the truest sense. This report presents the findings of the security assessment of our project's smart contracts.

Scope

The scope of this project has been mentioned below

Repository: <https://github.com/lapitstechnologies/audits/tree/main/NovaDAO>

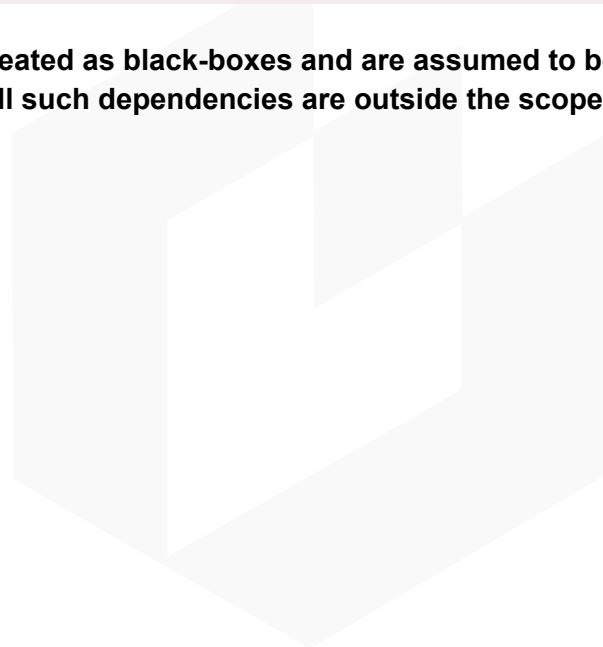
Technical Documentation: <https://medium.com/@Nova.DAO/novadao-whitepaper-v2-44c9e9e1ddd>

Tokenomics: <https://medium.com/@Nova.DAO/novanomics-78a1f9edf7d5>






Type: White Paper & Tokenomics (Requirements Provided)

Files: NovaGenesisPass.sol

Note - All external dependencies have been treated as black-boxes and are assumed to be free of any vulnerabilities. Unless explicitly mentioned, all such dependencies are outside the scope of this report.



Vulnerability Indicators

Indicator	Risk Level	Description
	Critical	Critical vulnerabilities are usually straight-forward to exploit and can lead to asset loss or data manipulations
	Major	Major vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
	Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to asset loss or data manipulations
	Minor	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution
	Informational	Informational vulnerabilities are problems that do not pose a risk to security. These are merely improvements over the existing code



Checked Items

The provided smart contract has been checked for the following vulnerabilities. Here are a few parameters that have been considered.

Item	Type	Description	Status
Visibility Test	SWC-100 , SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow And Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the solidity compiler	Passed
Floating/Unlocked Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Failed
Unchecked Call Return Value	SWC-104	The Return Value of The Contract Should Be Checked	Not Relevant
Access Control & Authorization	CWE-184	There shouldn't be any possibility of illegal takeover of ownership	Passed
Self-Destruction Instruction	SWC-106	The contract should not be destroyed until it has funds belonging to users.	Not Relevant
Deprecated Solidity Functions	SWC-111	Avoid using deprecated built-in functions	Passed
DoS (Denial of Service)	SWC-113 , SWC-128	Contract execution must not be stopped due to a failing call	Passed
Race Conditions/Front-running	SWC-114	Unsynchronized Transactions, order dependency	Passed
Presence of unused variables	SWC-131	The contract should ideally not contain unused variables	Passed
Authorization Through TX Origin	SWC-115	Tx.origin shouldn't be used for Authorization	Passed
Unencrypted Private Data on Chain	SWC-136	No private data should be stored on a blockchain	Passed
Reentrancy Attack	SWC-107	State variables should be updated before performing a transaction	Passed
Improper/Uneven Code Style	Custom	The code should be evenly styled, with proper documentation and identifier casing	Failed
Incorrect Order of Inheritance	SWC-125	The contract must follow the correct order of Inheritance	Passed
Gas Griefing	SWC-126	Contract should be safe from gas griefing attacks	Passed
Oracle Manipulation	Custom	Contract must be safe from oracle manipulation attacks	Not Relevant

Project Overview

The project is titled NovaDAO, and the subject of our security audit is the contract called NovaGenesisPass. This contract is an implementation of the ERC-721 token standard, meaning it is a Non Fungible Token. It implements the ERC-721a standard instead of the regular OpenZeppelin implementation for gas cost savings.

NovaGenesisPass is the contract used for minting and managing the '**NovaDeus**' or **NovaGod NFT**. NFTs are minted for 0.1 eth each and the price can be adjusted by the owner. The contract by default allows every user to mint 5 NFTs at most with two whitelisted users who can mint an infinite amount of NFTs. Users can also mint multiple NFTs in a single transaction.

Requirements

Other than the requirements prescribed by ERC-721 standard, following are the requirements of NovaGenesisPass contract

- 1) Users - The contract is required to have two types of users; normal, and white-listed users. Normal users can mint only a limited amount of NFTs as described by the contract limits whereas white-listed users have no such boundation on the mints, the contract should have only two whitelisted at any given time.
- 2) NFT Mints - Each user is allowed to mint NFTs as long as they're within the ranges prescribed by the contract. These limits are only applicable on the non-whitelisted users and are described by the individual NFT mint limit as well as the global NFT threshold limit.
- 3) Owner Privileges - The contract owner should have the authority to change –
 - A. The base NFT price
 - B. Global NFT Threshold
 - C. NFT Limit Per User

Test Coverage

Total Functions : 20

Total Coverage : 0%

Tests Conducted

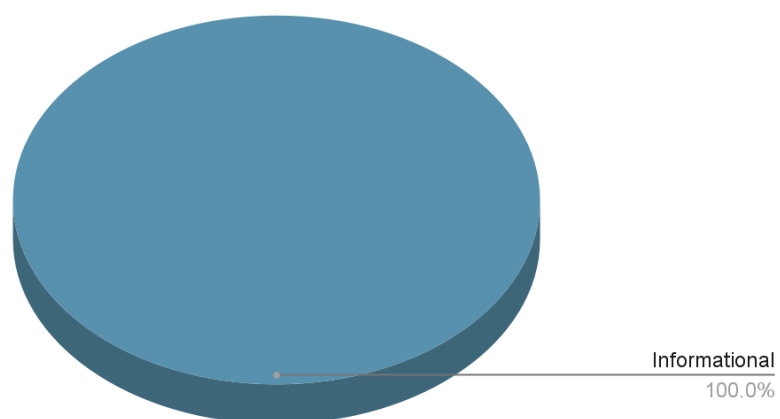
No Tests Have Been Conducted For The Most Recent Version of NovaGenesisPass.sol

Function Name	File Name	Status
contractURI()	NovaGenesisPass.sol	N/A
setContractURI(string memory contractURI_)	NovaGenesisPass.sol	N/A
_baseURI()	NovaGenesisPass.sol	N/A
setBaseURI(string memory _baseUri)	NovaGenesisPass.sol	N/A
setNftThreshold(uint16 _threshold)	NovaGenesisPass.sol	N/A
getNftThreshold()	NovaGenesisPass.sol	N/A
setNftprice(uint16 _price)	NovaGenesisPass.sol	N/A
getNftPrice()	NovaGenesisPass.sol	N/A
setNftPerUser(uint8 nftPerUser_)	NovaGenesisPass.sol	N/A
getNftPerUser()	NovaGenesisPass.sol	N/A
setWhitelistUser1(address whitelistAddress)	NovaGenesisPass.sol	N/A
getWhitelistUser1()	NovaGenesisPass.sol	N/A
setWhitelistUser2(address whitelistAddress)	NovaGenesisPass.sol	N/A
getWhitelistUser2()	NovaGenesisPass.sol	N/A
mint(uint8 _quantity)	NovaGenesisPass.sol	N/A
_whiteMintPass(uint8 _quantity)	NovaGenesisPass.sol	N/A
_mintPass(uint8 _quantity)	NovaGenesisPass.sol	N/A
getWalletAddress()	NovaGenesisPass.sol	N/A
setWalletAddress(address _wallet)	NovaGenesisPass.sol	N/A

To make the test coverage equal to 95%, the tests should be conducted for other functions as well. **A minimum of 19 functions need to be tested—for the test coverage to reach the desired goal.**

Vulnerability Report

Vulnerability Score



Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Mitigated	Resolved
Critical	0	0	0	0	0	0	0
Major	0	0	0	0	0	0	0
Medium	0	0	0	0	0	0	0
Minor	0	0	0	0	0	0	0
Informational	3	3	0	0	0	0	0

1 – Unlocked Pragma

Category : **Informational**

Location : <https://github.com/lapitstechnologies/audits/blob/main/NovaDAO/Contracts/NovaGenesisPass.sol>

Status : Pending

Description

The contract's pragma declares all solidity compiler versions above 0.8.4 valid for our contract. The contract however, has only been tested for v0.8.4. This should immediately be fixed as it can lead to unknown bugs and vulnerabilities for the unlocked compiler versions.

Recommendation

The pragma should be changed to `pragma solidity v0.8.4;` This ensures that the compiler stays locked at version 0.8.4;

2 – Use of Long, Lengthy Number Notations

Category : **Informational**

Location : <https://github.com/lapitstechnologies/audits/blob/main/NovaDAO/Contracts/NovaGenesisPass.sol>

Status : Pending

Description

The contract makes use of lengthy decimal notations for numbers. This is problematic for two reasons; one - it increases the contract size (even though minimally) and second, it makes the number hard to read.

Recommendation

It is suggested that we replace decimal notation with scientific or hexadecimal ones

Eg.

```
Use - uint256 private _nftPrice = 1e17;  
      Instead of  
uint256 private _nftPrice = 100000000000000000;
```

3 – Uneven Documentation/Code Style

Category : **Informational**

Location : <https://github.com/lapitstechnologies/audits/blob/main/NovaDAO/Contracts/NovaGenesisPass.sol>

Status : Pending

Description

The smart contract hasn't been properly documented, at some places it even breaks the **mixed-case styling** that is followed throughout the contract.

Recommendation

- 1) It is suggested that all the identifier names should be put in a mixed-case and should all be evenly styled, Eg.

```
function setNftprice(uint16 _price)
should instead be
function setNftPrice(uint16 _price)
```

- 2) All the important functions, and the variables should be properly documented.
Eg.
The Function setNftThreshold(uint _threshold) should be documented as follows

```
/**
 * @dev Changes The Global NFT Threshold
 *
 */
function setNftThreshold(uint16 _threshold) public onlyOwner {
    _nftThreshold = _threshold;
}
```

Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices against the most common cybersecurity vulnerabilities and issues in smart contract source code. The details of the diagnosis are disclosed in this report, the audit makes no statements or warranties on the security of the code. It does not guarantee the safety of this contract

It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report. It is important to note that we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts



Get Your Smart Contracts Approved, Tested and Secured By Professionals At [Lapits](https://lapits.com)

About Lapits

Cited as one of the best “**Blockchain Services Provider**” companies in India, Lapits is a Blockchain, Web Development and Mobile App development firm, with a long proven history of delivering quality software solutions. Lapits is one of the leaders in the Crypto and Web3 space and has been awarded by multiple national and international renowned bodies.

Lapits houses a powerful team of developers, business analysts and sales managers who know the industry in and out. Be it driving sales, developing apps, or launching an exchange; at lapits, we do it all. For more information on services provided by lapits, head over to [Lapits.com](https://lapits.com)

