



SOLIDProof

Bring trust into your projects

**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY

Hoolies

AUDIT

SECURITY ASSESSMENT

21. August, 2023

FOR



SolidProof_io



@solidproof_io



Introduction	3
Disclaimer	3
Project Overview	4
Summary	4
Social Medias	4
Audit Summary	5
File Overview	6
Imported packages	6
Components	7
Exposed Functions	7
Capabilities	8
Inheritance Graph	9
Audit Information	10
Vulnerability & Risk Level	10
Auditing Strategy and Techniques Applied	11
Methodology	11
Overall Security	12
Upgradeability	12
Ownership	13
Ownership Privileges	14
Minting tokens	14
Burning tokens	15
Blacklist addresses	16
Fees and Tax	17
Lock User Funds	18
Centralization Privileges	19
Audit Results	20

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	Hoolies
Website	https://www.hoolies.io/
About the project	A mind-blowing collection of hand-drawn goodness, packed with a gazillion rad visual traits living on the Ethereum blockchain. We're talking rare characters, sick costumes, funky gadgets, and a palette bursting with more colors than a unicorn's dreams. But it's not just NFTs. Hoolies brings a new term in web 3.0 space: NFTC - which means NFT+Coin. Yep, we initiate our own \$HOO coins in the Hoolieverse! Everyone who mint an NFT will get \$HOO coins into their wallets automatically. These coins can be exchanged on DEX and CEX, used in our p2e game or spent in our marketplace.
Chain	TBA
Language	Solidity
Codebase	TBA
Commit	N/A
Unit Tests	Not Provided

Social Medias

Telegram	N/A
Twitter	https://twitter.com/HoolieVerse
Facebook	N/A
Instagram	N/A
GitHub	N/A
Reddit	N/A
Medium	N/A
Discord	N/A
YouTube	N/A
TikTok	N/A
LinkedIn	N/A



Audit Summary

Version	Delivery Date	Change Log
v1.0	21. August 2023	<ul style="list-style-type: none"> · Layout Project · Automated/ Manual-Security Testing · Summary

Note - The following audit report presents a comprehensive security analysis of the smart contract utilized in the project. This analysis did not include functional testing (or unit testing) of the contract's logic. We cannot guarantee 100% logical correctness of the contract as it was not functionally tested by us.





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/HooliesTestNFT.sol	41548d1f64eb51f33a00396d98511a850b960319

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(local imports).

```

./extensions/ERC721AQueryable.sol
@openzeppelin/contracts/access/Ownable.sol
@openzeppelin/contracts/utis/cryptography/MerkleProof.sol
@openzeppelin/contracts/security/ReentrancyGuard.sol
./DefaultOperatorFilterer.sol
@openzeppelin/contracts/utis/Strings.sol
./IHooliesTokenTest.sol

```

Note for Investors: We only audited contracts mentioned in the scope above. All contracts related to the project apart from that are not a part of the audit, and we cannot comment on its security and are not responsible for it in any way. This contract imports some local contracts which are not audited by us, and we will not be responsible for any security cause for those contracts.





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 needed within visibility modifier, such as public, private or internal, which determines the access level of the variable.

Components

 Contracts	 Libraries	 Interfaces	 Abstract
1	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
20	2








External	Internal	Private	Pure	View
0	19	0	0	4

StateVariables

Total	 Public
17	16



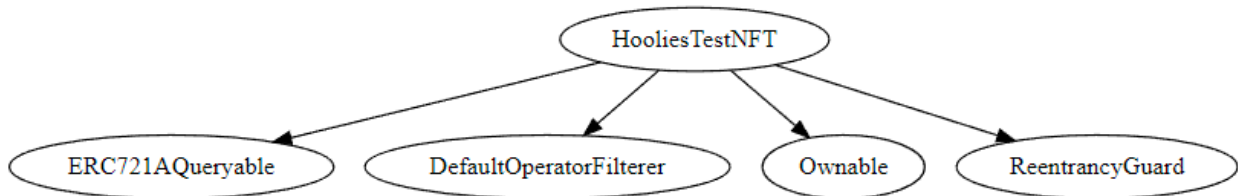
Capabilities

Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts	
0.8.17	-----	yes		-----	
 Transfers ETH	 Low-Level Calls	 Delegate Call	 Uses Hash Functions	 ECRrecover	 New/Create/Create2
-----		-----	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 the 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 as possible.
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 Upgradeability

Contract is not an upgradable



Deployer cannot update the contract with new functionalities.

Description	The contract is not an upgradeable contract. The Deployer is not able to change or add any functionalities to the contract after deploying.
Comment	N/A





Ownership

The ownership is not renounced

✗ The ownership is not renounced

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.

Example	N/A
Comment	N/A

Note – *The contract cannot be considered as renounced till it is not deployed or having some functionality that can change the state of the contract.*



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.

Contract owner cannot mint new tokens

 **The owner cannot mint new tokens**

Description	The owner is not able to mint new tokens once the contract is deployed.
Comment	The owner of the contract has the ability to mint the tokens to any particular address without paying any minting fees.



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 set fees more than 25%

Description

The owner is not able to set the fees above 25%.

Comment

N/A

Lock User Funds

In a smart contract, locking refers to the process of restricting access to certain tokens or assets for a specified period of time. When token 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.

Contract owner can lock functions.

✗ The owner can lock functions.

Description	The owner can lock the contract by any functions or update any variables.
Comment	This contract does contain some functionality that can lock the functions for an unlimited period. Moreover, the minting tokens can be locked by the owner for an indefinite period for the users that are not whitelisted. Also, the owner can set the max mint amount and if it is set to 0. Then no minting of tokens will be possible.

File/ Line(s): L276-278, L263-265
codebase: HolliesTestNFT.sol

```
ftrace | funcSig
function setPaused(bool _state!) public onlyOwner {
    paused = _state!;
}
```

```
ftrace | funcSig
function setMaxMintAmountPerTx(uint256 _maxMintAmountPerTx!) public onlyOwner {
    maxMintAmountPerTx = _maxMintAmountPerTx!;
}
```

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
HolliesTestNFT.sol	<ul style="list-style-type: none"> ➤ The owner can stop the sale only once. ➤ The owner can set any number in the mint start Timestamp. ➤ The owner can update any number in the daily prices for the token. ➤ The owner can update the Day token accrual rate to any number. ➤ The whitelist users can mint tokens even if the minting is paused. ➤ The owner can mint tokens for the particular account. ➤ The owner can set any number in the cost of tokens. ➤ The owner can set any number to the minimum mint amount per transaction. Also, the owner can add zero which will stop the minting of tokens. ➤ The owner can update the URI prefix and suffix for the token. ➤ The owner can pause the minting of tokens for the users who are not whitelisted. ➤ The owner can set any value to the Merkle root. ➤ The owner can update the getAddress which will receive the eth amount after minting tokens. ➤ The owner can enable/disable the minting for whitelist addresses. ➤ The owner can send the any amount of eth to the GETAddress account which will receive the eth after the minting is done. ➤ The owner can withdraw contract's balance.



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 Result

#1 | Logical error.

File	Severity	Location	Status
HolliesTestNFT.sol	Medium	L139, L295-300	Open

Description – There must be a certain threshold for adding the timestamp value, If the owner will set the value which is greater than the block.timestamp then currentDay value will be negative and then no operation will be possible. It is recommended to add the 'require' check that the mintStartTimeStamp must be less than the current block.timestamp. Also, on mintTransferOption function remove the 'onlyOwner' modifier as it will prevent minting the tokens from other users and the minting of tokens will fail. Add a private visibility so no one can call this function externally.

#2 | Owner can set arbitrary value in day prices.

File	Severity	Location	Status
HolliesTestNFT.sol	Medium	L122-124	Open

Description – There must be a certain threshold value that the owner cannot set more than that value, If the owner will set the prices so high then no minting of the token is possible.

#3 | Owner can set arbitrary value in the token quantity.

File	Severity	Location	Status
HolliesTestNFT.sol	Medium	L126-128	Open

Description – There must be a minimum threshold for the token quantity if the owner has set the token quantity to zero then minting will not be done to the account for the token and also the fees of the token will be spent even after there is no transfer of tokens to the user.

#4 | Owner can set arbitrary value to the cost of minting.

File	Severity	Location	Status
HolliesTestNFT.sol	Medium	L259-261	Open

Description – The owner has the ability to add any value to the cost of minting which is not recommended. There must be a certain threshold for the amount otherwise no user will be able to mint tokens.

#5 | Owner can lock minting by setting any number to the max mint amount.

File	Severity	Location	Status
HolliesTestNFT.sol	Medium	L263-265	Open

Description – Add a 'require' check that there must be a certain value that the owner cannot set. For example, if the owner has set it to zero then there will not be any minting possible from the users.

#6 | Owner can pause minting for unlimited period of time.

File	Severity	Location	Status
HolliesTestNFT.sol	Medium	L280-282	Open

Description – There must be some certain period that the minting will not be paused for an indefinite period of time for the users that are not whitelisted. Add a locking period for the pausing of mint functionality.

#7 | Access control.

File	Severity	Location	Status
HolliesTestNFT.sol	Medium	L280-282	Open

Description – Add 'onlyOwner' modifier so that no one can change the token address apart from the owner. As of now, anyone can change it to any address and then the transactions will fail.

#8 | Missing events.

File	Severity	Location	Status
HolliesTestNFT.sol	Low	L73, 83, 122, 127, 260, 264, 269, 273, 281, 277, 287, 292	Open

Description - Emit all the critical parameter changes.

#9 | Missing zero or dead address check.

File	Severity	Location	Status
HolliesTestNFT.sol	Low	L286-288, L64-66, L41-55	Open

Description – Add a ‘require’ check that the address should not be zero or dead.

#10 | Missing ‘require’ error message.

File	Severity	Location	Status
HolliesTestNFT.sol	Low	L304	Open

Description – Add a message to the require check that will be received if the transaction is failed.

#11 | Incorrect ‘require’ check.

File	Severity	Location	Status
HolliesTestNFT.sol	Low	L205	Open

Description – Add ‘==’ instead of ‘>=’ if the amount is greater than the cost then the user will not receive any ethers back if he pays more than the required cost.

#12 | Owner can change the Merkle root anytime.

File	Severity	Location	Status
HolliesTestNFT.sol	Informational	L280-282	Open

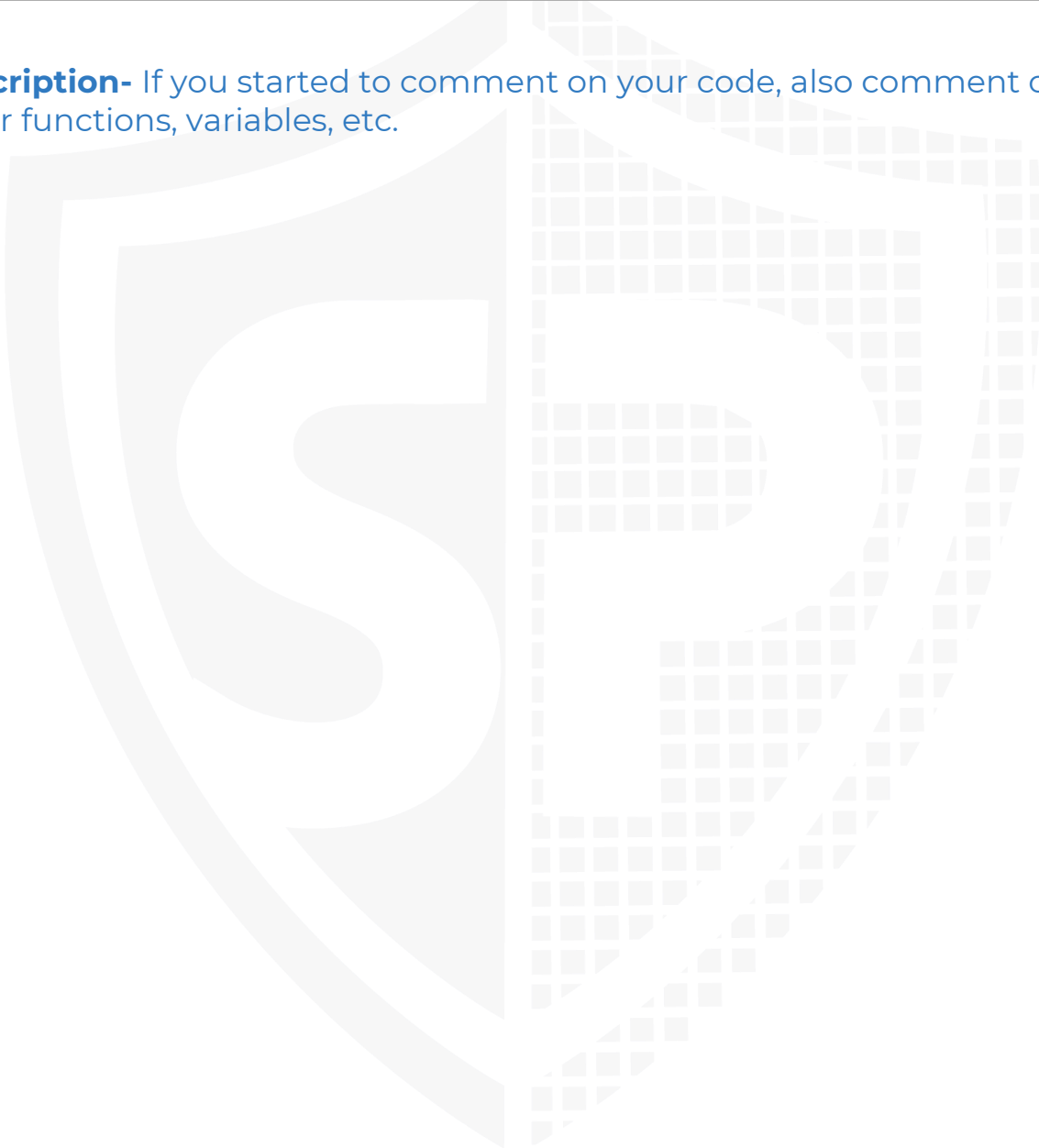


Description – It is recommended that the root should not be set after initial deployment.

#13 | NatSpec Documentation missing

File	Severity	Location	Status
HolliesTestNFT.sol	Informational	--	Open

Description- If you started to comment on your code, also comment on all other functions, variables, etc.





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.





**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY