



SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Audit

Security Assessment

28. October, 2021

For



Disclaimer	3
Description	5
Project Engagement	5
Logo	5
Contract Link	5
Methodology	7
Used Code from other Frameworks/Smart Contracts (direct imports)	8
Tested Contract Files	9
Source Lines	10
Risk Level	10
Capabilities	11
Scope of Work	13
Inheritance Graph	13
Verify Claims	14
OnlyOwner functions	22
CallGraph	23
Source Units in Scope	24
Critical issues	25
High issues	25
Medium issues	25
Low issues	25
Informational issues	25
Audit Comments	26
SWC Attacks	27

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 security or functionality of the technology we agree to analyze.

Version	Date	Description
1.0	28. October 2021	<ul style="list-style-type: none">• Layout project• Automated- /Manual-Security Testing• Summary

Network

Binance Smart Chain (BEP20)

Website

<https://www.lucidlands.io/>

Telegram

<https://t.me/LucidLandsOffical>

Twitter

https://twitter.com/Lucid_Lands

Facebook

<https://www.facebook.com/gaming/LucidLandsOfficial>

Youtube

<https://www.lucidlands.io/#>

Instagram

https://www.instagram.com/official_lucidlands

Discord

<https://discord.gg/caXEYZb5Dw>

Description

Lucid Lands is the first decentralized NFT play-to-earn game on BSC network that integrates both 3D-animated gaming NFT and 2D unique computer-generated collective NFT marketplace.

Each unique NFT Heroes will hold its intrinsic value complimenting the rarity, which can be traded in the marketplace.

Furthermore, these Collectibles are utilized in the game which will aggregate the floor price according to its attributes and activity.

Project Engagement

During the 25th of October 2021, **Lucidlands Team** engaged Solidproof.io to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. They provided Solidproof.io with access to their code repository and whitepaper.

Logo



Contract Link

v1.0

TBA

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. 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 evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
 - i) Review of the specifications, sources, and instructions provided to SolidProof to make sure we understand the size, scope, and functionality of the smart contract.
 - ii) Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
 - iii) Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to SolidProof describe.
2. Testing and automated analysis that includes the following:
 - i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
 - ii) Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

Used Code from other Frameworks/Smart Contracts (direct imports)

Imported packages:

- OpenZeppelin
- ERC721



Tested Contract Files

This audit covered the following files listed below with a SHA-1 Hash.

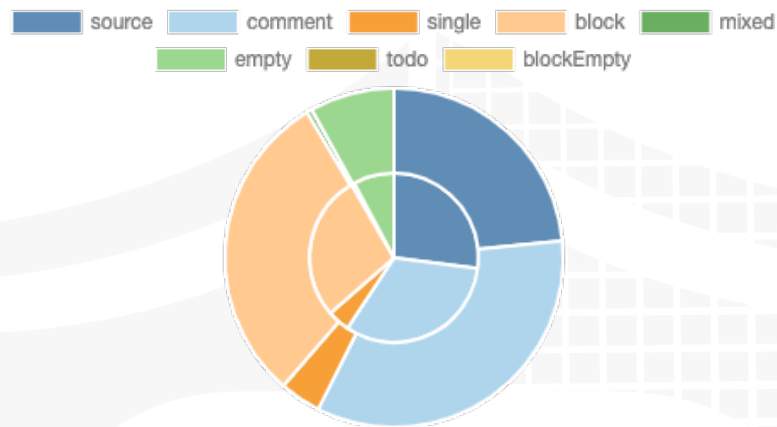
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

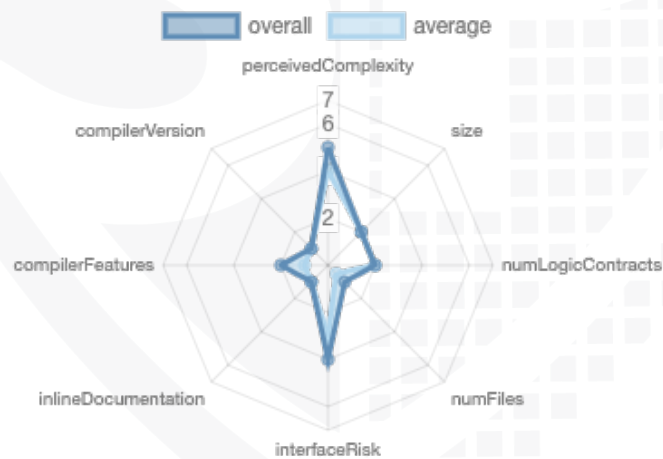
File Name	SHA-1 Hash
contracts/ShroomieS.sol	4baf4c4c9a48b02471653d2c8880359edf465b4d
contracts/ERC721.sol	d3b323cf4be4dda1343f8738af38c301e2060933

Metrics

Source Lines v1.0



Risk Level v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	2	5	5	2

Exposed Functions

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

Version	Public	Payable
1.0	46	2

Version	External	Internal	Private	Pure	View
1.0	17	110	15	15	61

State Variables

Version	Total	Public
1.0	22	7

Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	<div><div>>=0.4.21</div><div><0.8.0</div><div>>=0.6.0</div><div><0.8.0</div><div>>=0.6.2</div><div><0.8.0</div></div>		yes	yes (2 asm blocks)	

Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/ Create/ Create2
1.0	yes		yes			



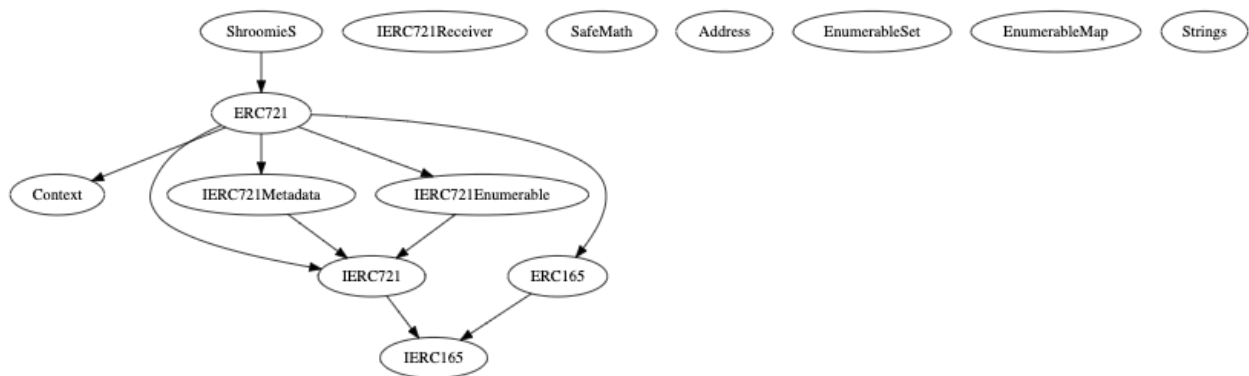
Scope of Work

The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

We will verify the following claims:

1. Correct implementation of Token standard
2. Deployer cannot mint any new tokens
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. Overall checkup (Smart Contract Security)

Inheritance Graph v1.0



Verify Claims

Correct implementation of Token standard

Tested	Verified
✓	✓

Function	Description	Exist	Tested	Verified
TotalSupply	provides information about the total token supply	✓	✓	✓
BalanceOf	provides account balance of the owner's account	✓	✓	✓
Transfer	executes transfers of a specified number of tokens to a specified address	✓	✓	✓
TransferFrom	executes transfers of a specified number of tokens from a specified address	✓	✓	✓
Approve	allow a spender to withdraw a set number of tokens from a specified account	✓	✓	✓
Allowance	returns a set number of tokens from a spender to the owner	✓	✓	✓

Write functions of contract

approve

buyToken

changeTokenP...

mintShroomY

safeTransferFr...

safeTransferFr...

setApprovalFo...

setMintPrice

toggleForSale

transferFrom

Deployer cannot mint any new tokens

Name	Exist	Tested	Verified	File
Deployer cannot mint	✓	✓	✗	Main
Comment	Line: -			

Max / Total Supply: 0

Comments:

v1.0

- Everybody can mint tokens but you have to pay for it
 - Paid amount goes to the owner

Deployer cannot burn or lock user funds

Name	Exist	Tested	Verified
Deployer cannot lock	✓	✓	✓
Deployer cannot burn	✓	✓	✓

Comments:



v1.0

```
function mintShroomY(  
    string memory _name↑,  
    string memory _tokenURI↑,  
    uint256 _price↑  
) public payable {  
    require(msg.sender != address(0));  
    shroomYCounter++;  
    require(!_exists(shroomYCounter));  
  
    require(!_tokenURIExists[_tokenURI↑]);  
    require(!_tokenNameExists[_name↑]);  
}
```

- Requirement conditions for minting Shroom
- Can only mint when token is not existing

```
function _mint(address to↑, uint256 tokenId↑) internal virtual {  
    require(to↑ != address(0), "ERC721: mint to the zero address");  
    require(!_exists(tokenId↑), "ERC721: token already minted");  
}
```

```
function buyToken(uint256 _tokenId↑) public payable {  
    require(msg.sender != address(0));  
    require(_exists(_tokenId↑));  
    address tokenOwner = ownerOf(_tokenId↑);  
    require(tokenOwner != address(0));  
    require(tokenOwner != msg.sender);  
    ShroomY memory shroomy = allShroomies[_tokenId↑];  
    require(msg.value >= shroomy.price);  
    require(shroomy.forSale);  
}
```

- Requirement conditions for buyingToken

```
function changeTokenPrice(uint256 _tokenId↑, uint256 _newPrice↑) public {  
    require(msg.sender != address(0));  
    require(_exists(_tokenId↑));  
    address tokenOwner = ownerOf(_tokenId↑);  
    require(tokenOwner == msg.sender);  
}
```

- Requirement conditions for changing token price

```
function toggleForSale(uint256 _tokenId↑) public {
    require(msg.sender != address(0));
    require(_exists(_tokenId↑));
    address tokenOwner = ownerOf(_tokenId↑);
    require(tokenOwner == msg.sender);
}
```

- Requirement conditions for changing toggling sale

```
function transferFrom(address from↑, address to↑, uint256 tokenId↑) public virtual override {
    //solhint-disable-next-line max-line-length
    require(_isApprovedOrOwner(msgSender(), tokenId↑), "ERC721: transfer caller is not owner nor approved");
    _transfer(from↑, to↑, tokenId↑);
}

function _isApprovedOrOwner(address spender↑, uint256 tokenId↑) internal view virtual returns (bool) {
    require(_exists(tokenId↑), "ERC721: operator query for nonexistent token");
    address owner = ERC721.ownerOf(tokenId↑);
    return (spender↑ == owner || getApproved(tokenId↑) == spender↑ || ERC721.isApprovedForAll(owner, spender↑));
}
```

Deployer cannot pause the contract

Name	Exist	Tested	Verified
Deployer cannot pause	✓	✓	✓



Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

Legend

Attribute	Symbol
Verified / Checked	✓
Partly Verified	⚠
Unverified / Not checked	✗
Not available	—







OnlyOwner functions

`setMintPrice`



Source Units in Scope

v1.0

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/ShroomieS.sol	1	—	160	148	125	1	86	
	contracts/ERC721.sol	8	5	1776	1601	585	1012	371	
	Totals	9	5	1936	1749	710	1013	457	

Legend

Attribute	Description
Lines	total lines of the source unit
nLines	normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
nSLOC	normalized source lines of code (only source-code lines; no comments, no blank lines)
Comment Lines	lines containing single or block comments
Complexity Score	a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

Audit Results

AUDIT PASSED

Critical issues

- no critical issues found -

High issues

- no high issues found -

Medium issues

- no medium issues found -

Low issues

Issue	File	Type	Line	Description
#1	Main	Contract doesn't import npm packages from source (like OpenZeppelin etc.)	-	We recommend to import all packages from npm directly without flatten the contract. Functions could be modified or can be susceptible to vulnerabilities
#2	Main	A floating pragma is set	4	The current pragma Solidity directive is „>=0.6.0 <0.8.0 ""
#3	Main	State variable visibility is not set	12	It is best practice to set the visibility of state variables explicitly
#4	Main	Missing Events Arithmetic	82	Emit an event for critical parameter changes

Informational issues

- no informational issues found -

Audit Comments

28. October 2021:

- `mintPrice` state variable has no functionality



SWC Attacks

ID	Title	Relationships	Status
SW C-13 6	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SW C-13 5	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SW C-13 4	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SW C-13 3	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SW C-13 2	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SW C-13 1	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SW C-13 0	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SW C-12 9	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SW C-12 8	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED

SW C-12 7	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
SW C-12 5	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
SW C-12 4	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
SW C-12 3	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SW C-12 2	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
SW C-12 1	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SW C-12 0	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	PASSED
SW C-11 9	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	PASSED
SW C-11 8	Incorrect Constructor Name	CWE-665: Improper Initialization	PASSED
SW C-11 7	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	PASSED

SW C-11 6	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-11 5	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SW C-11 4	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SW C-11 3	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SW C-11 2	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SW C-11 0	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SW C-10 9	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SW C-10 8	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	NOT PASSED
SW C-10 7	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SW C-10 6	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED

SW C-10 5	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SW C-10 4	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SW C-10 3	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	NOT PASSED
SW C-10 2	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SW C-10 1	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	PASSED
SW C-10 0	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED

The logo features the word "SolidProofed" in a white, elegant script font. The "P" is particularly large and stylized, with a long horizontal stroke that extends to the left. The background is a solid blue color with a faint, large shield emblem. The shield has a grid-like pattern on its right side and a solid blue area on its left side.

SolidProofed

Blockchain Security | Smart Contract Audits | KYC

A small horizontal bar representing the German flag, with black, red, and gold stripes.

MADE IN GERMANY