



SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Audit

Security Assessment
26. August, 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	12
Inheritance Graph	12
Verify Claims	13
CallGraph	18
Source Units in Scope	19
Critical issues	20
High issues	20
Medium issues	20
Low issues	20
Informational issues	20
SWC Attacks	21

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	26. August 2021	<ul style="list-style-type: none">• Layout project• Automated- /Manual-Security Testing• Summary

Network

Binance Smart Chain (BEP20)

Website

<https://pokmonsters.com/>

Telegram

<https://t.me/PokMonsters>

Twitter

https://twitter.com/pokmonsters_nft



Description

NFT is a trendy topic these days, with an increasing amount of interest from all around the world. Many major players have begun their NFT projects, and a major of NFT projects have been produced, including collecting, farming for NFT, and so on... In reality, non-fungible tokens (NFTs) have become a hot topic among not just blockchain enthusiasts but also more mainstream individuals, particularly in the gaming industry. However, most individuals find NFT technology intimidating, prompting them to be hesitant to use it. Furthermore, many NFT initiatives have made no progress or innovation in educating users about the technology to increase their faith in it, instead focusing their efforts on developing complex systems to support their tokens and NFT marketplaces. Pokmonsters is one of the earliest attempts to offer NFT experiences to game enthusiasts. It is designed to be a relaxing, easy-to-play NFT gaming experience that everyone can enjoy.

Project Engagement

During the 24th of August, **Pokmonsters 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. **Pokmonsters Team** provided Solidproof.io with access to their code repository and whitepaper.

Logo



Contract Link

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
 - Address
 - Ownable
 - SafeMath
 - IERC20Metadata
 - IERC20
 - SafeERC20
 - IERC721
 - IERC721Locker
 - ReentrancyGuard
 - OwnableUpgradable
 - ContextUpgradeable
 - IERC165



Tested Contract Files

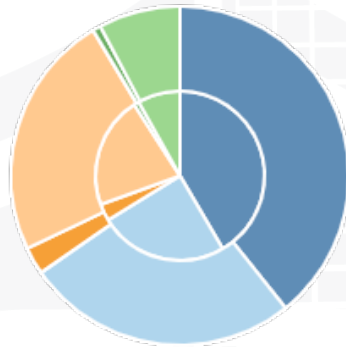
This audit covered the following files listed below with a SHA-1 Hash.

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.

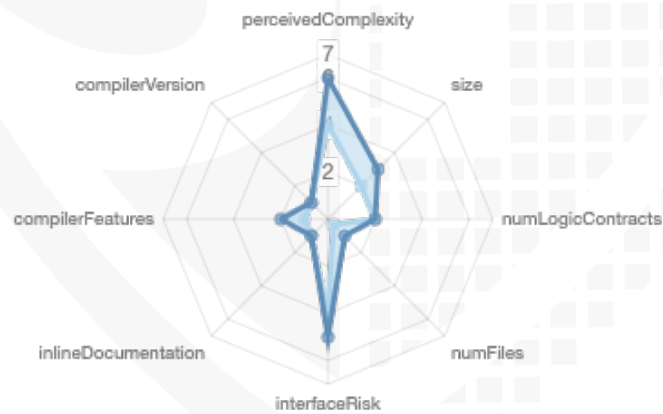
File Name	SHA-1 Hash
contracts/Timelock.sol	3d02eea60e8cbea646339c115cb7018ffcb7bab2
contracts/Chef.sol	f6f27c11e6cc5513e3513a45718e9d61faa25b61
contracts/pokmonsters.sol	414373ccb2dea1584bbe2ea52865e898e464e8d2

Metrics

Source Lines



Risk Level



Capabilities

Components

Contracts	Libraries	Interfaces	Abstract
5	4	9	6

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
109	2

External	Internal	Private	Pure	View
79	180	4	27	50

State Variables

Total	Public
54	38

Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
0.8.6		yes	yes (3 asm blocks)	

Transfers ETH	Low-Level Calls	Delegate Call	Uses Hash Functions	ECRecover	New/Create/Create2
		yes	yes		

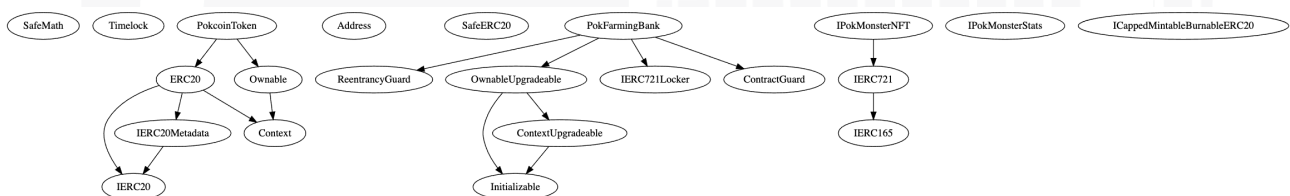
Scope of Work

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



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	✓	✓	✓

Optional implementations

Function	Description	Exist	Tested	Verified
renounceOwnership	Owner renounce ownership for more trust	✓	✓	—

Deployer cannot mint any new tokens

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

Max / Total Supply: 51.480.000 ether

Comment:

- Owner can set Minter
 - Minter can mint

```
constructor() ERC20("P.com", "SYMBOL") {  
    cap = 99000000 ether; // Max Supply: 99 million pokcoin  
    mint(_msgSender(), (cap * 52) / 100); //IDO + Add lq: 50% + 2% Airdrop  
}  
  
function mint(address _recipient, uint256 _amount) external onlyMinter returns (bool) {  
    uint256 balanceBefore = balanceOf(_recipient);  
    mint(_recipient, _amount);  
    return balanceOf(_recipient) > balanceBefore;  
}
```

Deployer cannot burn or lock user funds

```
function burn(uint256 _amount) external {  
    burn(msg.sender, _amount);  
}
```

```
function burnFrom(address _account, uint256 _amount) external {  
    approve(_account, msgSender(), allowance(_account, msgSender()).sub(_amount, "ERC20: burn amount exceeds allowance"));  
    burn(_account, _amount);  
}
```

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

Deployer cannot pause the contract

Name	Exist	Tested	Verified
Deployer cannot pause	✓	✓	✓



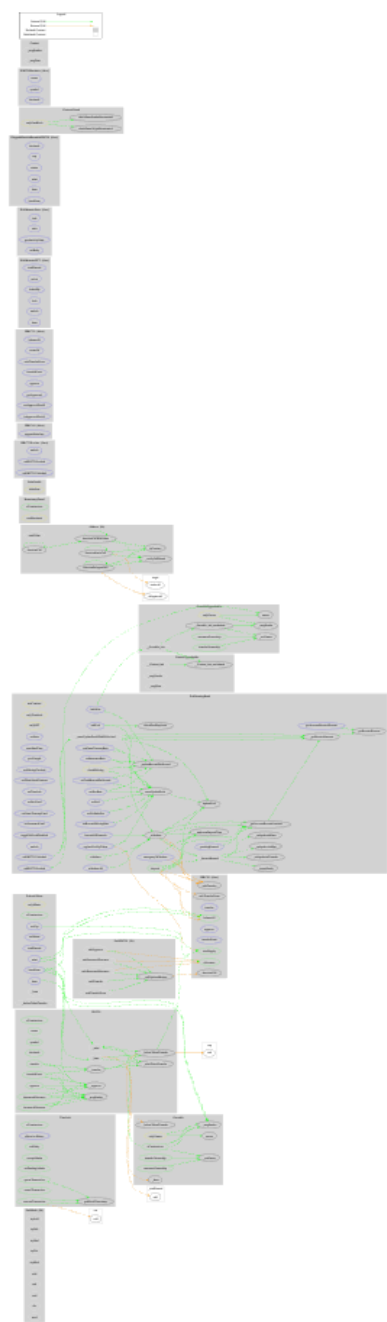
Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓









Legend

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

CallGraph



Source Units in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/Timelock.sol	2	—	339	339	145	145	91	
	contracts/Chef.sol	8	7	1811	1394	989	480	585	
	contracts/pokmonsters.sol	5	2	862	748	271	461	189	
	Totals	15	9	3012	2481	1405	1086	865	

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

Informational issues

Issue	File	Type	Line	Description
#1	Main	Missing Zero Address Validation (missing-zero-check)	1146	Check that the address is not zero
#2	Timelock	Missing Zero Address Validation (missing-zero-check)	308, 275	Check that the address is not zero

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	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 words "Solid Proofed" in a white, elegant script font. The word "Solid" is positioned above "Proofed". Behind the text is a faint, stylized shield emblem. The shield is divided into four quadrants: the top-left and bottom-right are solid blue, while the top-right and bottom-left contain a grid pattern. The entire design is set against a solid blue background.

Solid
Proofed

Blockchain Security | Smart Contract Audits | KYC

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

MADE IN GERMANY