



**SOLID**Proof  
*Bring trust into your projects*

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

MADE IN GERMANY

# Audit

**Security Assessment**  
**31. August, 2021**

**For**

**CRYPTOKAI**

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	19
Source Units in Scope	20
Critical issues	21
High issues	21
Medium issues	21
Low issues	21
Informational issues	21
Audit Comments	22
SWC Attacks	23

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

## Network

Binance Smart Chain (BEP20)

## Website

<https://cryptokai.io/>



## Description

CryptoKAI(Yōkai - 妖怪) are a class of supernatural entities and spirits in the digital world of cryptocurrency living in Binance Smart Chain (BSC) Platform.

CryptoKAI is inspired by the haunted, friendly, mysterious and malevolent beings in Japanese folklore. CryptoKAI brings its devious world of NFT gaming and earning in cryptocurrency.

Battle and farm \$KAI using these malevolent monsters in our Blockchain-based NFT gaming. Fight monsters or take on your fellow players!

## Project Engagement

During the 29th of August 2021, **CryptoKai 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**

<https://bscscan.com/address/0x37c12504d411de64dd74cbfbb51e6d9f25c38968#code>

# 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)
<b>Critical</b>	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.
<b>High</b>	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.
<b>Medium</b>	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.
<b>Low</b>	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.
<b>Informational</b>	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
  - Context
  - ReentrancyGuard
  - ERC20
  - IERC20
- Uniswap
  - UniswapV2Factory
  - UniswapV2Pair
  - UniswapV2Router01
  - UniswapV2Router02



## Tested Contract Files

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

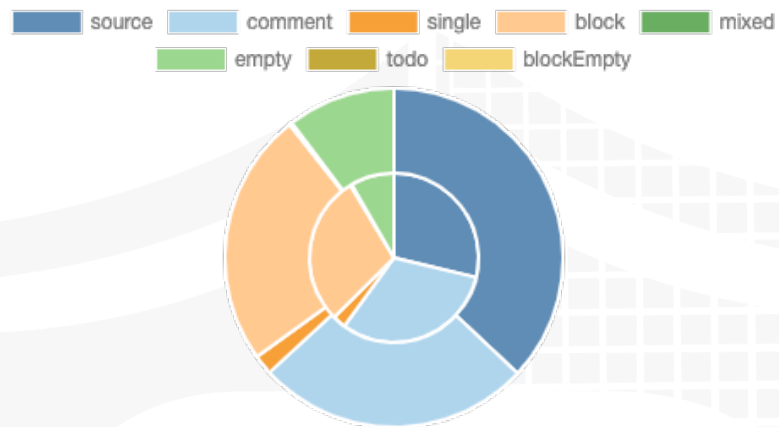
*A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.*

### v1.0

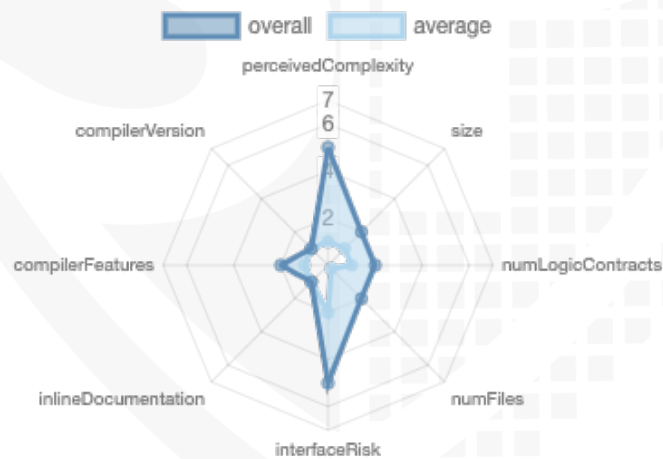
File Name	SHA-1 Hash
contracts/KaiERC20.sol	0f45bf1f7dbe85d7c6a121b68edf4a119bbacecd
contracts/CryptoKai.sol	f5d8f9c1a9257b68816658fcf8fac408c5a8ecd6
contracts/Context.sol	a34cc2179b2da819d60afa9d711d0094d5a72799
contracts/Uniswap.sol	570cda5051df3f905a74466062a24cec0ee4bb77
contracts/SafeMath.sol	252b3cae72fa4bde1cf723d04677a593bd82d36
contracts/Ownable.sol	9e39ce6c806d48065c2984fcb706045879309079
contracts/ERC20.sol	3164cd1b5dd73d692da25725a173e0f5377ac157
contracts/ReentrancyGuard.sol	15418ab2dc6d962dd376d991088e3577b36ddaaa
contracts/IERC20.sol	2d6eb8a102a8a92dcfef4d19c977a062e891c7cf
contracts/ManagerInterface.sol	8e52fa64cc8d0bb3946f836d978199d6d4c1a54e

# Metrics

## Source Lines v1.0



## Risk Level v1.0



## Capabilities

### Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	3	1	6	3

### 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	99	5

Version	External	Internal	Private	Pure	View
1.0	80	80	1	23	41

### State Variables

Version	Total	Public
1.0	27	14

### Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	<code>^0.7.6</code> <code>&gt;=0.6.0</code> <code>&lt;0.8.0</code>		yes	**** (0 asm blocks)	

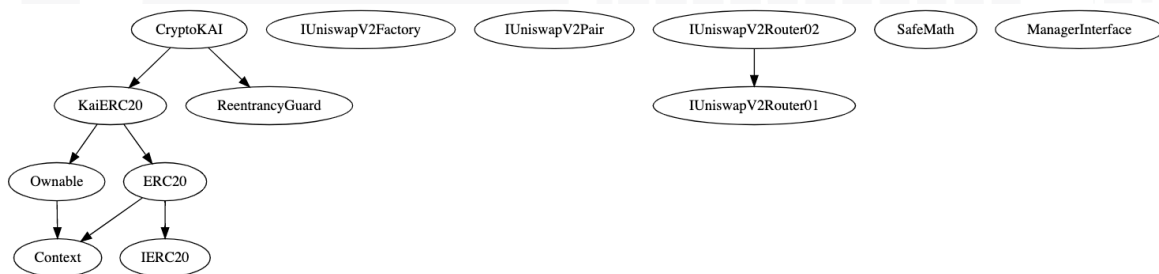
## Scope of Work

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	✓	✓	✓

### 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: 57.000.000

Comments:

### v1.0

- `_mint` is called in KaiERC20 file
  - Function win
  - Function farm

```
function win(address winner↑, uint256 reward↑) external onlyKAI {
    require(playToEarnReward != amountPlayToEarn, "Over cap farm");
    require(winner↑ != address(0), "0x is not accepted here");
    require(reward↑ > 0, "not accept 0 value");

    playToEarnReward = playToEarnReward.add(reward↑);
    if (playToEarnReward <= amountPlayToEarn) _mint(winner↑, reward↑);
    else {
        uint256 availableReward = playToEarnReward.sub(amountPlayToEarn);
        _mint(winner↑, availableReward);
        playToEarnReward = amountPlayToEarn;
    }
}
```

```
constructor(string memory name, string memory symbol) KaiERC20(name, symbol)
{
    _mint(_msgSender(), maxSupply.sub(amountFarm).sub(amountPlayToEarn));
    IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(
        0x10ED43C718714eb63d5aA57B78B54704E256024E
    );

    uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
        .createPair(address(this), _uniswapV2Router.WETH());

    uniswapV2Router = _uniswapV2Router;
    _approve(address(this), address(uniswapV2Router), ~uint256(0));
}
```

```
function _mint(address account, uint256 amount) internal virtual {  
    require(account != address(0), "ERC20: mint to the zero address");  
  
    _beforeTokenTransfer(address(0), account, amount);  
  
    _totalSupply = _totalSupply.add(amount);  
    _balances[account] = _balances[account].add(amount);  
    emit Transfer(address(0), account, amount);  
}
```



## Deployer cannot burn or lock user funds

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

1. antiBot	→
2. approve	→
3. burn	→
4. decreaseAllowance	→
5. farm	→
6. increaseAllowance	→
7. renounceOwnership	→
8. setAddressForBosses	→
9. setBots	→
10. setManager	→
11. setMinTokensBeforeSwap	→
12. setTransferFeeRate	→
13. sweepTokenForBosses	→
14. transfer	→
15. transferFrom	→
16. transferOwnership	→
17. win	→



## Deployer cannot pause the contract

Name	Exist	Tested	Verified
Deployer cannot pause	✓	✓	✓

Comments:

**v1.0**

.

1. antiBot	→
2. approve	→
3. burn	→
4. decreaseAllowance	→
5. farm	→
6. increaseAllowance	→
7. renounceOwnership	→
8. setAddressForBosses	→
9. setBots	→
10. setManager	→
11. setMinTokensBeforeSwap	→
12. setTransferFeeRate	→
13. sweepTokenForBosses	→
14. transfer	→
15. transferFrom	→
16. transferOwnership	→
17. win	→

## Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

### Legend



















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

# CallGraph



# Source Units in Scope

## v1.0

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/KaiERC20.sol	1	————	86	80	63	1	57	————
	contracts/CryptoKai.sol	1	————	118	114	86	7	86	
	contracts/Context.sol	1	————	24	24	10	12	1	
	contracts/Uniswap.sol	————	4	344	22	18	1	134	
	contracts/SafeMath.sol	1	————	214	214	61	139	16	
	contracts/Ownable.sol	1	————	68	68	27	33	24	————
	contracts/ERC20.sol	1	————	306	306	89	185	79	————
	contracts/ReentrancyGuard.sol	1	————	62	62	15	38	5	
	contracts/IERC20.sol	————	1	77	26	17	57	13	
	contracts/ManagerInterface.sol	————	1	25	6	3	1	21	————
	<b>Totals</b>	<b>7</b>	<b>6</b>	<b>1324</b>	<b>922</b>	<b>389</b>	<b>474</b>	<b>436</b>	

## 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	A floating pragma is set	2	The current pragma Solidity directive is „^0.7.6“.

## Informational issues

Issue	File	Type	Line	Description
#1	Main	Conformity to Solidity naming conventions (naming-convention)	104	

# Audit Comments

## 31. August 2021:

- There is still an owner (Owner still has not renounced ownership)
- Developer can set antibot amount only one time
- Developer can set address as bot address
- Developer can lock the contract once for 10 minutes
  - Following conditions should be true for locking
    - antiBotTime should be higher than block timestamp
    - Amount should be higher than antiBotAmount
    - Sender should be marked as bot in bots mapping as true with setBots function
- Developer can set sell and buy fee rate
- \_mint function is called in KaiERC20 file
  - Function win
  - Function farm
- KaiERC20 can set manager address with setManager function
  - That addresses which Developer can set in the contract is not part of this report

## SWC Attacks

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

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



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

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

The logo features the words "SolidProof" in a white, handwritten-style script. 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 in the center. The shield has a grid-like pattern on its right side and a solid blue area on its left side.

SolidProof

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

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

MADE IN GERMANY