SMART CONTRACTS AUDITS
KYC CERTIFICATIONS

WEB3 DAPPS DEVELOPMENT
BUSSINES ANALYSIS

SECURING THE FUTURE
TRUSTCODEX

SMART CONTRACT AUDIT

SQUIRTLE POKEMON

Final version – 28 June 2023

AUDIT PASSED

TABLE OF CONTENT

Disclaimer

Please be aware that these evaluations should not be regarded as an endorsement or critique of any particular project or team, nor do they provide any guarantees regarding the security of said project. Additionally, these assessments should not be considered indicative of the economic value or viability of any product, service, or asset created by a team. It is important to note that Trustcodex does not conduct testing or auditing of integrations with external contracts or services.

Furthermore, these assessments should not be interpreted as having any direct impact on the potential economic outcomes of a token or any other related product, service, or asset. Trustcodex audits do not offer any warranties or representations to third parties and should not be utilized as a basis for making investment decisions. Trustcodex do not provide investment advice and should not be relied upon as such. It is worth mentioning that Trustcodex holds no obligation towards any third party as a result of publishing these assessments.

The primary objective of Trustcodex assessments is to assist customers in enhancing the quality of their code, while concurrently minimizing the inherent risks associated with cryptographic tokens and blockchain technology. However, it is crucial to understand that each company and individual is ultimately responsible for conducting their own due diligence and maintaining the ongoing security of their technology. Trustcodex does not guarantee the security or functionality of the analyzed technology.

The assessments are exclusively created for clients and are published with their explicit consent. The scope of the review is limited to the Solidity code, which is still in the developmental phase and susceptible to unidentified risks and vulnerabilities. It is important to clarify that the review does not extend beyond Solidity to encompass the compiler layer or any other areas that may pose potential security risks.

Trustcodex may choose to make these assessments available to parties other than clients on their website or various social media platforms. However, clients bear the responsibility of determining how the content of the assessments may be utilized on other websites. Trustcodex assumes no liability for the accuracy or completeness of any outcomes generated by third-party software employed on the website.

1. Audit overview

1.1. Methodology

Our team of experienced testers and smart contract developers carried out a line-by-line review, meticulously documenting any problems encountered. During the evaluation process, utmost care is exercised to examine the repository for security concerns, code excellence, and compliance with specifications and established practices.

The auditing process follows a routine series of steps:

- **Preparation:** Attain a thorough comprehension of the smart contract's purpose, functionality, and requirements, encompassing its intended use case, business logic, and security prerequisites.
- **Documentation Review:** Scrutinize the smart contract documentation to ensure its accuracy in describing the contract's functionality and accessibility to all relevant parties.
- **Code Review:** Conduct a comprehensive evaluation of the smart contract's code, encompassing its logic, data structures, and security measures, utilizing a combination of automated and manual methods.
- **Functionality Testing:** Verify the smart contract's functionality by subjecting it to various scenarios to ensure its intended operation and appropriate handling of edge cases.
- **Security Assessment:** Assess the smart contract's security measures, including its resilience against attacks, while identifying any potential vulnerabilities such as unhandled exceptions, reentrancy issues, and contract denial-of-service (DoS) attacks.
- **Performance Optimization**: Analyze the smart contract's performance metrics, including gas usage and execution time, and provide recommendations for enhancing efficiency and reducing costs.
- **Reporting:** Prepare a comprehensive report documenting the findings and recommendations, including a detailed description of any identified security vulnerabilities and suggested remedial measures.

1.2. Vulnerability & Risk level

Risk Level is determined by assessing the likelihood of a specific source-threat exploiting a vulnerability and evaluating the potential impact of such an event on the organization or system. The calculation of Risk Level follows the guidelines of CVSS version 3.0.

LEVEL	VULNERABILITY	ACTIONS
CRITICAL	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	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	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	A vulnerability that does not have a significant impact on scenarios for the use of the contract and is subjective.	Implementation of certain corrective actions or accepting the risk.
INFORMATIONAL	A vulnerability that has informational character but is not affecting any of the code.	An observation that does not determine a level of risk

2.Scope of work

The purpose of this audit is to conduct a thorough assessment of the code and functionality of a smart contract in order to detect any potential security weaknesses, enhance performance, and verify that the contract functions as intended.

The project team has provided us with the necessary files, such as those on Github, Bscscan, Etherscan, etc., which require testing. Our focus will be on evaluating the overall performance of the provided contracts and identifying any vulnerabilities or discrepancies in owner privileges that could potentially impact users interacting with these contracts.

Version	Date	Scope
1.0	24.06.23	First audit release

2.1. Project description

Name: SQUIRTLE POKEMON

Website: https://squirtlepokemon.com/

Twitter: https://twitter.com/Squitleweb3nft

Telegram: N/A

KYC: Project has not doxed with Trustcodex

Network: Binance Smart Chain

Contract v1.0:

https://bscscan.com/token/0xe5bc71a670bd0b2476bf412a183e9404389b2672

Logo:

Description:

A blockchain project that brings a fresh, vibrant, and exciting experience to the world of the metaverse. It is a unique digital playground where your favorite Pokémon toys come to life, breathing with vitality and exploring the vast world of the metaverse.

2.2. Assessed contracts

Tested Contract Files

The audit encompassed the following files listed below, along with their corresponding SHA-1Hash. It has come to our attention that one of the files has been altered, intentionally or unintentionally, subsequent to the security review, resulting in a different SHA-1Hash.

V1.0

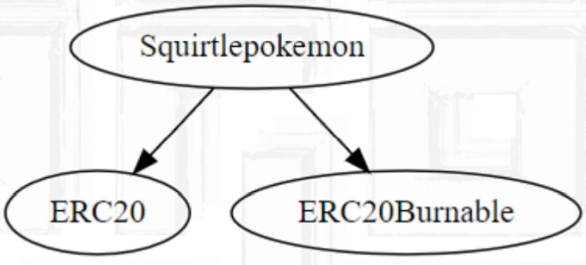
File name	SHA-1Hash
SquirtelPokemoon.sol	164daff747962beb9bf36b94173814d600697e80

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

Imported packages:

Dependency / import Path	Count
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol	1

Inheritance graph



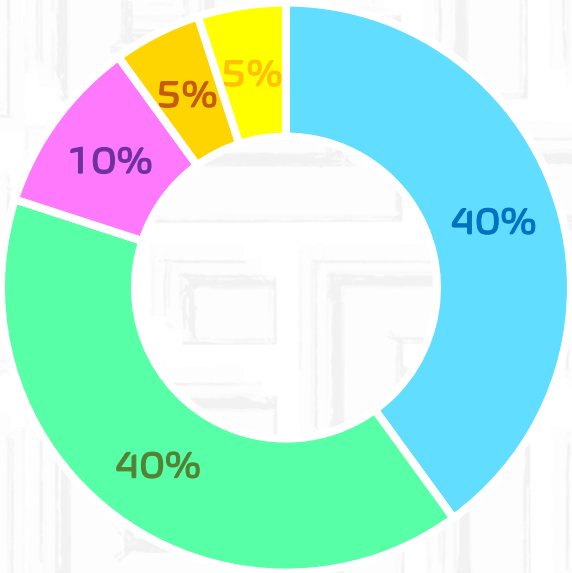
3. Findings

Throughout the review process, our team of highly skilled pentesters and smart contract developers conducted a comprehensive evaluation, which encompassed an overall project analysis. Great care was taken to assess the repository for security issues, code standard, and conformity to specifications and industry norms. A meticulous line-by-line examination was carried out, wherein any identified issues were recorded in detail.

3.1. Token

3.1.1. Distribution

- Burned supply
- Liquidity
- Marketing
- CEX listing
- TEAM



3.1.2. Liquidity status

The current audit block reveals a lack of liquidity for the token. If the liquidity becomes accessible, it opens the door for the notorious act referred to as a 'rug pull' by token developers.

As investors purchase tokens from the exchange, the liquidity pool gradually accumulates an increasing number of coins with established value, such as ETH, BNB, or Tether. At that point, developers can withdraw the liquidity from the exchange, convert it into cash, and make off with the funds.

To instill confidence in investors and prevent the token developers from absconding with the liquidity funds, the liquidity is secured by renouncing ownership of liquidity pool (LP) tokens for a predetermined duration. These tokens are sent to a time-lock smart contract, effectively preventing developers from reclaiming the liquidity pool funds. This practice has become a standard procedure followed by all token developers, and it serves as a clear distinction between a legitimate coin and a fraudulent one.

Furthermore, the team can enhance safety measures by permanently removing LP shares through burning, ensuring an even higher level of security (time lock without limit).

3.1.3. Privileges and roles

Correct implementation of Token standard

Function	Description	Exist	Status
Total supply	Provides information about the total token supply	YES	
BalanceOf	Provides account balance of the owner's account	YES	
Transfer	Executes transfers of a specified number of tokens to a specified address	YES	
TransferFrom	Executes transfers of a specified number of tokens from a specified address	YES	
Approve	Allow a spender to withdraw a set number of tokens from a specified account	YES	
Allowance	Returns a set number of tokens from a spender to the owner	YES	

Is contract an upgradeable

Description	Status	
Upgradeable contract	NO	

Owner cannot deploy a new version of the contract changing any setting providing the owners new privileges.

Deployer can interact with the contract

Function	Description	Exist	Status
RenounceOwnership	Owner renounce ownership	YES	

The contract ownership is not currently renounced. The ownership of the contract grants special powers to the protocol creators, making them the sole addresses that can call sensible ownable functions that may alter the state of the protocol.

Deployer can mint any new tokens

Function	Description	Exist	Status
Mint	Deployer can mint	NO	

The project owners do not have a mint function in the contract, owner cannot mint tokens after initial deploy. The project has a total supply and owners cannot mint any more than the max supply.

Deployer can burn or lock user funds

Function	Description	Exist	Status
Lock	Deployer can lock user funds	NO	
Burn	Deployer can burn funds	NO	

The project owners cannot manually burn LP funds or lock user funds.

Deployer can pause the contract

Function	Description	Exist	Status
Pause	Deployer can pause	NO	

The project owners cannot stop or pause trading. Investors can trade at any given time if owner disable swap

Deployer can change fees in the contract

Function	Description	Exist	Status
editFee	Deployer can edit fees	NO	⊗

The project owners have not the ability to set fees.

Deployer can blacklist addresses in the contract

Function	Description	Exist	Status
blacklist	Deployer can blacklist	NO	⊗

The project owners of do not have a blacklist function their contract. The contract does not allow owners to transfer their tokens without any restrictions. Token owner cannot blacklist the contract: Malicious or compromised owners can trap contracts relying on tokens with a blacklist.

Deployer can whitelist addresses in the contract

Function	Description	Exist	Status
whitelist	Deployer can whitelist	NO	⊗

The project owners do not have a whitelist function their contract. The contract does not allow owners to transfer their tokens without any fee or max tx requirements. Token owner normally is whitelisted to avoid paying fees or being blocked by other functions.

Deployer can set Max transfer in the contract

Function	Description	Exist	Status
MaxTx	Deployer can set max transfer	NO	⊗

The project owners cannot set max tx amount. The team do allow investors to swap, transfer or sell more than max tx amount.

Deployer can set Maxwallet in the contract

Function	Description	Exist	Status
Maxwallet	Deployer can set maxwallet	NO	⊗

The project owners cannot set max wallet amount. The team do allow investors to hold more than certain tokens to avoid centralization. Nevertheless, same user can have several max wallets.

Legend

Symbol	Meaning
	Passed
	Not passed
	Not available

Modifiers and public functions

Privileges and modifiers comments

N/A

SWC check

The following table contains an overview of the SWC registry. Each row consists of an SWC identifier (ID), weakness title, CWE parent and list of related code samples. For further information regarding SCW ID or CEW relationships go to https://swcregistry.io/

ID	Title	Relationships	Status
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	
SWC-116	Block values as a proxy for time	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	

SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	👍
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	👍
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	👍
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	👍
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	👍
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	👍
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	👍
SWC-126	Insufficient Gas Griefing	CWE-691: Insufficient Control Flow Management	👍
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	👍
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	👍
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	👍
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	👍
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	👍
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking	👍
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	👍
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	👍
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	👍
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	👍

Legend

Symbol	Meaning
👍	Passed
⚠	Not passed

4.Audit results

4.1. Critical issues

LEVEL	VULNERABILITY	ACTIONS
CRITICAL	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.

Issue	File	Line	Description	Status
-	-	-	NO CRITICAL ISSUES FOUND	-

4.2. High issues

LEVEL	VULNERABILITY	ACTIONS
HIGH	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.

Issue	File	Line	Description	Status
-	-	-	NO HIGH ISSUES FOUND	-

4.3. Medium issues

LEVEL	VULNERABILITY	ACTIONS
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.

Issue	File	Line	Description	Stauts
-	-	-	NO MEDIUM ISSUES FOUND	-

4.4. Low issues

LEVEL	VULNERABILITY	ACTIONS
LOW	A vulnerability that does not have a significant impact on scenarios for the use of the contract and is subjective.	Implementation of certain corrective actions or accepting the risk.

Issue	File	Line	Description	Status
#1	Main SC	2	It is recommended to add the constant version of solidity as this prevents the unintentional deployment of a contract with an outdated compiler that contains unresolved bugs	-

4.5. Informational issues

LEVEL	VULNERABILITY	ACTIONS
INFORMATIONAL	A vulnerability that has informational character but is not affecting any of the code.	An observation that does not determine a level of risk

Issue	File	Line	Description	Status
#1	Main SC	-	If you started to comment on your code, also comment on all other functions, variables, etc.	-

V1.0 Issues Comments

We recommend you to use NatSpec Format form (https://docs.soliditylang.org/en/latest/natspec-format.html) for your contracts to provide rich documentation for functions, return variables and more. This helps investors to make clear what that variable, functions etc. do.

4.6. Audit Score

Description	Score
Project score	85/100
Code structure score	87/100
Auditor score	82/100
SWC score	83/100

Final score	85/100

To enhance the evaluation process and determine the value of the audit, a scoring system has been implemented on this page. The scoring system ranges from 0 to 100, with 100 being the highest achievable score. However, in order to attain a perfect score, the project must meet all the necessary criteria and provide all the required data for assessment.

It is important to note that the passing score has been established at 75 points. If a project fails to reach a score of 75% or higher, it will be deemed an automatic failure. Please refer to the following notes and final assessment for further details.

AUDIT PASSED