SECURING THE FUTURE

TRUSTCODEX

SMART CONTRACT AUDIT

KANNAGI FINANCE

Final version – 25 June 2023

AUDIT PASSED

TABLE OF CONTENT

Disclaimer

Note that these reports do not serve as an endorsement or criticism of any project or team and do not guarantee the security of the project. The reports should not be considered as a reflection of the economics or value of any product, service, or asset created by a team. Trustcodex does not include testing or auditing of integrations with external contracts or services.

These reports should not be interpreted as having any impact on the potential economics of a token or any other product, service, or asset. Trustcodex audits do not provide any warranty or representation to any third party and should not be used to make investment decisions.

The reports do not provide investment advice and should not be used as such. Trustcodex has no obligation to any third party by publishing these reports.

The goal of Trustcodex reports is to help customers enhance the quality of their code while reducing the prominent level of risk associated with cryptographic tokens and blockchain technology. Each company and individual are responsible for their own due diligence and ongoing security, and Trustcodex does not guarantee the security or functionality of the technology analyzed.

The reports are created for clients and are published with their consent. The scope of the review is limited to the Solidity code, which is still under development and subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas beyond Solidity that may pose security risks.

Trustcodex may make the reports available to parties other than clients on their website or social media platforms. The clients are responsible for determining how the content of the reports may be used on other websites. Trustcodex do not assume responsibility for the use of third-party software on the website and are not liable for the accuracy or completeness of any outcomes generated by such software.

It's important to note that the content of the reports is current as of the date appearing on the report and is subject to change.

1. Audit overview

1.1. Methodology

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

The auditing process follows a routine series of steps:

- **Preparation:** Obtain a clear understanding of the purpose, functionality, and requirements of the smart contract, including its intended use case, business logic, and security requirements.
- **Documentation Review:** Review the smart contract documentation to ensure it accurately describes the contract's functionality and is accessible to all parties.
- **Code Review:** Thoroughly review the smart contract code, including its logic, data structures, and security measures, using automated and manual methods.
- **Functionality Testing:** Test the smart contract's functionality in various scenarios to verify that it operates as intended and handles edge cases appropriately.
- **Security Assessment:** Evaluate the smart contract's security measures, including its ability to withstand attacks, and identify any potential security vulnerabilities, such as unhandled exceptions, reentrancy issues, and contract denial-of-service (DoS) attacks.
- **Performance Optimization**: Analyze the smart contract's performance, including its gas usage and execution time, and recommend improvements to optimize its efficiency and reduce costs.
- **Reporting:** Prepare a comprehensive report of the findings and recommendations, including a clear and detailed description of any potential security vulnerabilities and recommendations for remediation.

1.2. 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	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 scope of work of this audit is a comprehensive evaluation of the code and functionality of a smart contract to identify potential security vulnerabilities, improve performance, and ensure the contract operates as intended.

The project team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). Therefore, we will verify the overall performance of the contracts provided and we will focus on possibles vulnerabilities and/or different owner privileges that might affect the users that might interacts with the contracts provided.

Version	Date	Scope
1.0	25.06.23	First audit release

2.1. Project description

Name: BOLT INU

Network: zkSync Era Network(era)

Contract v1.0:
https://explorer.zksync.io/address/0x26aC1D9945f65392B8E4E6b895969b5c01A7B414#contract

Website: https://www.kannagi.finance

Twitter: https://twitter.com/Kannagi_Zksync

Telegram: https://t.me/kannagifinanceINT

KYC: Project has not doxed with Trustcodex.

Logo:

Description:

Kannagi Finance maximizes yield by utilizing different strategies such as lending, yield farming, and compounding. These strategies are implemented on zkSync, a layer 2 solution that ensures low transaction fees and fast processing times. The platform also offers aggregation, allowing users to access the best available yields across different protocols. With its multi-strategy optimization value system, users are assured of the highest possible returns, while the KANA token rewards them for their trading activity. Additionally, Kannagi minimizes risks associated with low liquidity issues by spreading investments across various protocols.

2.2. Assessed contracts

Tested Contract Files

This audit covered the following files listed below with their SHA-1Hash. A file with a different SHA-1Hash has been modified, intentionally or otherwise, after the security review.

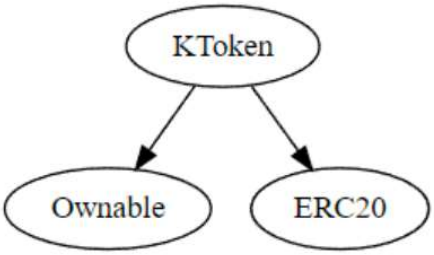
V1.0

File name	SHA-1Hash
Ktoken.sol	75f5ab4215428651aa0e9a24e81dfb17200744bd

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

```
@openzeppelin/contracts/token/ERC20/ERC20.sol
@openzeppelin/contracts/access/Ownable.sol
```

Inheritance graph



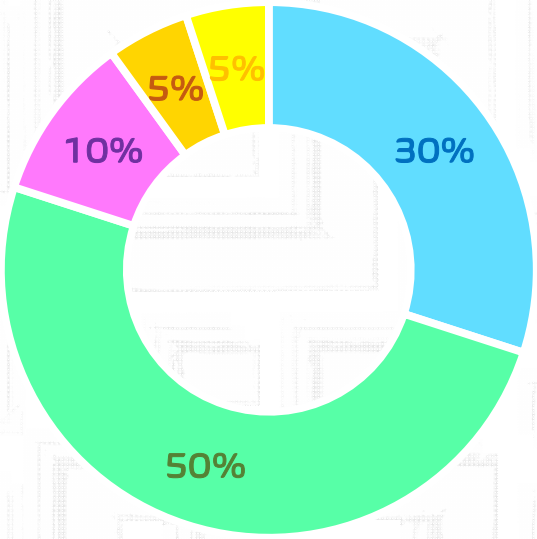
3. Findings

In the course of the review, great care was taken to assess the repository for security issues, code standard, and conformity to specifications and industry norms. This was accomplished through a meticulous line-by-line examination by our team of highly skilled pentesters and smart contract developers, who recorded any identified issues as they arose.

3.1. Token

3.1.1. Distribution

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



3.1.2. Liquidity status

The token does not have liquidity at the moment of the audit block. If liquidity is unlocked, then the token developers can do what is infamously known as 'rug pull'.

Once investors start buying token from the exchange, the liquidity pool will accumulate more and more coins of established value (e.g., ETH or BNB or Tether). Developers can withdraw this liquidity from the exchange, cash in all the value and run off with it.

Liquidity is locked by renouncing the ownership of liquidity pool (LP) tokens for a fixed period, by sending them to a time-lock smart contract. Without ownership of LP tokens, developers cannot get liquidity pool funds back. This provides confidence to the investors that the token developers will not run away with the liquidity money. It is now a standard practice that all token developers follow, and this is what really differentiates a scam coin from a real one.

LP shares can also be burned by the team to ensure even more safety (infinite time lock)

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 can not 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 cannot 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. The team may have fees defined; however, they may not be able to configure them above 25%.

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 project does not allows 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 have a whitelist function their contract. The project does allows 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 not allow investors to swap, transfer or sell more than max tx amount.

Deployer can set Maxwallet in the contract

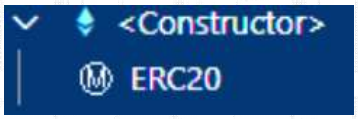
Function	Description	Exist	Status
Maxwallet	Deployer can set maxwallet	NO	

The project owners cannot set max wallet amount. The team do not 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

- No specific owner privileges

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

⚠ SWC-103 Floating Pragma

CWE-664: Improper Control of a Resource Through its Lifetime.

Description:

Contracts should be deployed with the same compiler version and fags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation:

Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen. Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

References:

Ethereum Smart Contract Best Practices - Lock pragmas to specific compiler version.

⚠ SWC-108 State Variable Default Visibility

CWE-710: Improper Adherence to Coding Standards

Description:

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Remediation:

Variables can be specified as being public, internal, or private. Explicitly define visibility for all state variables.

References:

Ethereum Smart Contract Best Practices - Explicitly mark visibility in functions and state variables.

4.Audit results

4.1. Audit Score

Description	Score
Overall score	85/100
Code structure score	86/100
Auditor score	83/100
SWC score	83/100

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however, to attain that value the project most pass and provide all the data needed for the assessment.

Our Passing Score has been set to 75 Points if a project does not attain 75% is an automatic failure. Read our notes and final assessment below.

4.2. 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	-

4.3. 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	-

4.4. 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	-

4.5. 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
-	-	-	NO MEDIUM ISSUES FOUND	-

4.6. 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.	Ackn

V1.0 Audit Comments

We recommend the use of 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.

AUDIT PASSED