SECURING THE FUTURE

TRUSTCODEX

SMART CONTRACT AUDIT

DARUMA

Final version – 06 June 2023

AUDIT PASSED

https://github.com/trustcodex

twitter.com/trustcodex

TABLE OF CONTENT

Disclaimer

Trustcodex typically earns compensation from one or more clients for conducting the analysis found in their reports. It's important to 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.

Cryptographic tokens and blockchain technology carry a high degree of technical risk and uncertainty. 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. Trustcodex and the customer may indicate otherwise.

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 above-mentioned 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.

Find here the following main goals of the study:
- Contract upgradeability
- Correct implementation of Token standard
- Deployer cannot mint any new tokens
- Deployer cannot burn or lock user funds
- Deployer cannot pause the contract
- Overall smart contract audit

2.1. Project description

Name: DARUMA

Network: Binance smart chain

0xaC9bDa4578612dAc995b527CB3d62a80B8389eE6

Max supply: 1B

Holders: 1

Mcap: N/A

Website: www.darumabsc.fun

Twitter: www.twitter.com/darumabsc

Telegram: www.t.me/daruma_bsc

2.2. Assessed contracts

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	Tx
DARUMA.sol	0x2135b91638792ad240dc171050a6ea677a1d9f7fbcce3921295f0c72d0dd255a

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

Imported packages:

- OpenZeppelin
- Address
- Ownable
- ReentrancyGuard
- Uniswap
- UniswapV2Factory
- UniswapV2Pair
- UniswapV2Router01
- UniswapV2Router02
- SafeMath

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. Metrics

50% burned supply

40% liquidity PancakSwap (to be locked)

5% Marketing

5% CEX

3.1.2. Graph and Inheritance

Available with premium VIP audit

3.1.3. Privileges and roles

Correct implementation of Token standard

Function	Description	Exist	Tested	Verified
Total supply	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	👍	👍	👍

Deployer cannot interact with the contract.

Function	Description	Exist	Tested	Verified
RenounceOwnership	Owner renounce ownership for more trust	👍	👍	👍

Deployer cannot mint any new tokens.

Function	Description	Exist	Tested	Verified
Mint	Deployer cannot mint	👍	👍	👍

Deployer cannot burn or lock user funds.

Function	Description	Exist	Tested	Verified
Lock	Deployer cannot lock	👍	👍	👍
Burn	Deployer cannot burn	👍	👍	👍

Deployer cannot pause the contract.

Function	Description	Exist	Tested	Verified
Pause	Deployer cannot pause	👍	👍	👍

Legend

Symbol	Meaning
👍	Completed
🔍	Partially completed
⚠	Not completed

3.1.4. 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

Audit Results

AUDIT PASSED

Critical issues

No critical issues found.

High issues

No critical issues found.

Medium issues

No critical issues found.

Low issues

LEVEL	VULNERABILITY	ACTIONS
LOW	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.

Issue	File	Line	Description
1	Main SC	-	Contract doesn’t import npm packages from source (like OpenZeppelin etc.) We recommend to import all packages from npm directly without flatten the contract.
2	Main SC	12	A floating pragma is set. The current pragma Solidity directive is „“<=0.6.0 <0.8.0.””.
3	Main SC	1152	State variable visibility is not set. It is best practice to set the visibility of state variables explicitly

Informational issues

No critical issues found