CFG NINJA AUDITS

Security Assessment

EmiratiDoge2.0 Token

July 9, 2023

Audit Status: Pass

Audit Edition: Advance

POWERED BY BLADE POOL

Risk Analysis

Classifications of Manual Risk Results

Classification	Description
Critical	Danger or Potential Problems.
Major	Be Careful or Fail test.
Minor	Pass, Not-Detected or Safe Item.
Informational	Function Detected

Manual Code Review Risk Results

Contract Priviledge	Description
Buy Tax	5
Sale Tax	5
Cannot Sale	Pass
Cannot Sale	Pass
Max Tax	5
Modify Tax	Not Detected
Fee Check	Pass
Is Honeypot?	Not detected
Trading Cooldown	Not Detected
Can Pause Trade?	Pass
Pause Transfer?	Not Detected

Contract Priviledge	Description
Max Tx?	Pass
Is Anti Whale?	Not Detected
Is Anti Bot?	Not Detected
Is Blacklist?	Not Detected
Blacklist Check	Pass
is Whitelist?	Not Detected
Can Mint?	Pass
Is Proxy?	Not Detected
Can Take Ownership?	Not detected
Hidden Owner?	Not detected
Owner	0x80e2783058ec35ab8f8cd9e609bf61f06c4867dd
Self Destruct?	Not Detected
Other?	Not detected
Other?	Not detected
Holders	1
Auditor Confidence	High

The following quick summary it's added to the project overview; however, there are more details about the audit and its results. Please read every detail.

Project Overview

Token Summary

Parameter	Result
Address	0xcE88603855Cf22958ee4c48E50aFE643083cE89D
Name	EmiratiDoge2.0
Token Tracker	EmiratiDoge2.0 ($EmirDoge2.0)
Decimals	9
Supply	10,000,000,000,000,000
Platform	Binance Smart Chain
compiler	v0.8.19+commit.7dd6d404
Contract Name	EmiratiDoge2_0
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	https://bscscan.com/address/0xce88603855cf22958ee4c48e50afe643083ce89d#code
Payment Tx	0x9bbbb4b09ae2679cac42c470bad215c5e20f456addcb43fab11f347fd7268251

Main Contract Assessed Contract Name

Name	Contract	Live
EmiratiDoge2.0	0xcE88603855Cf22958ee4c48E50aFE643083cE89D	Yes

TestNet Contract Assessed Contract Name

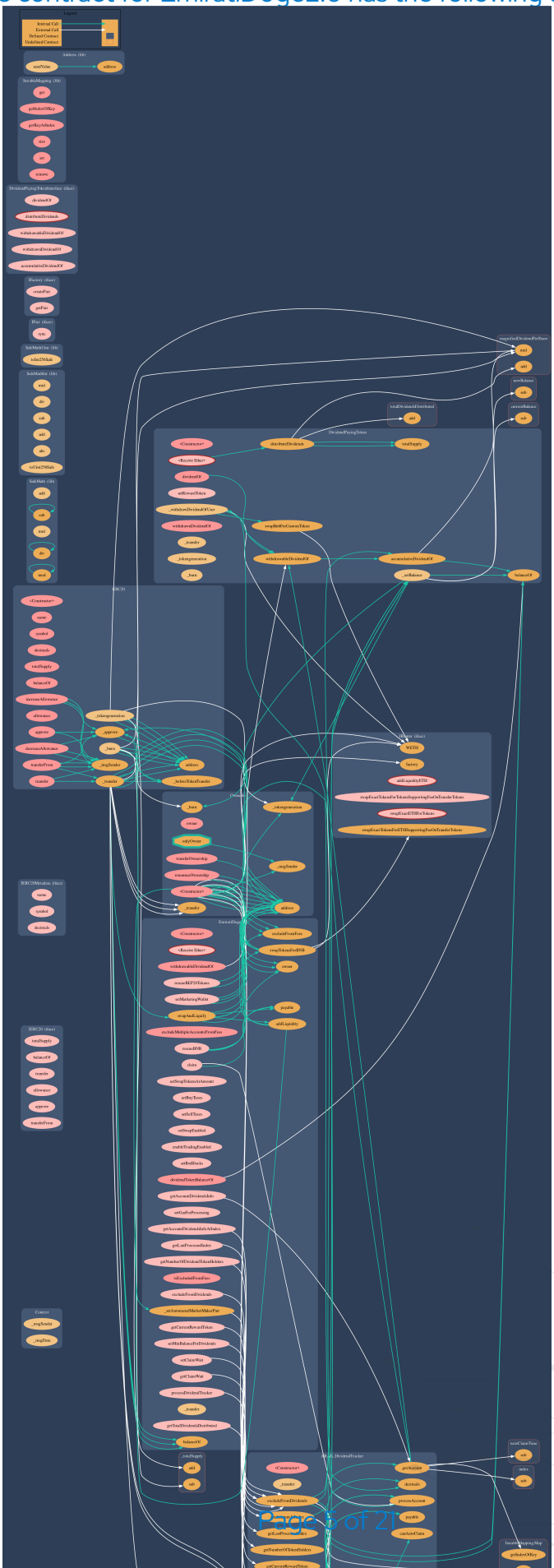
Name	Contract	Live
EmiratiDoge2.0	0x93a2836b28a06DB54feEAC1d3Be3567493caca2e	Yes

Solidity Code Provided

SolID	File Sha-1	FileName
EmiratiDoge2_0	2ee3328fd1efa5ab67cd289edba48a7c7c987000	EmiratiDoge2_0.sol
EmiratiDoge2_0		
EmiratiDoge2_0		
EmiratiDoge2_0		

Call Graph

The contract for EmiratiDoge2.0 has the following call graph structure.



Smart Contract Vulnerability Checks

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.

ID	Severity	Name	File	location
SWC-100	Pass	Function Default Visibility	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-103	Low	A floating pragma is set.	EmiratiDoge2_0.sol	L: 7 C: 0
SWC-104	Pass	Unchecked Call Return Value.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-105	Pass	Unprotected Ether Withdrawal.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-107	Pass	Read of persistent state following external call.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-108	Pass	State variable visibility is not set..	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-109	Pass	Uninitialized Storage Pointer.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-110	Pass	Assert Violation.	EmiratiDoge2_0.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-111	Pass	Use of Deprecated Solidity Functions.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-113	Pass	Multiple calls are executed in the same transaction.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-114	Pass	Transaction Order Dependence.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-115	Low	Authorization through tx.origin.	EmiratiDoge2_0.sol	L: 1075 C: 12, L: 1344 C: 20
SWC-116	Pass	A control flow decision is made based on The block.timestamp environment variable.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-117	Pass	Signature Malleability.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-118	Pass	Incorrect Constructor Name.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-120	Low	Potential use of block.number as source of randonmness.	EmiratiDoge2_0.sol	L: 1148 C: 28, L: 1304 C: 32
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-123	Pass	Requirement Violation.	EmiratiDoge2_0.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-124	Pass	Write to Arbitrary Storage Location.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-125	Pass	Incorrect Inheritance Order.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-126	Pass	Insufficient Gas Griefing.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-130	Pass	Right-To-Left-Override control character (U +202E).	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/Dead Code).	EmiratiDoge2_0.sol	L: 0 C: 0
SWC-136	Pass	Unencrypted Private Data On-Chain.	EmiratiDoge2_0.sol	L: 0 C: 0

We scan the contract for additional security issues using MYTHX and industry-standard security scanning tools.

Smart Contract Vulnerability Details

SWC-103 - Floating Pragma.

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

References:

Description:

Contracts should be deployed with the same compiler version and flags 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.

Smart Contract Vulnerability Details

SWC-115 - Authorization through tx.origin

CWE-477: Use of Obsolete Function

Description:

tx.origin is a global variable in Solidity which returns the address of the account that sent the transaction. Using the variable for authorization could make a contract vulnerable if an authorized account calls into a malicious contract. A call could be made to the vulnerable contract that passes the authorization check since tx.origin returns the original sender of the transaction which in this case is the authorized account.

Remediation:

tx.origin should not be used for authorization. Use msg.sender instead.

References:

Solidity Documentation - tx.origin

Ethereum Smart Contract Best Practices - Avoid using tx.origin

SigmaPrime - Visibility.

Smart Contract Vulnerability Details

SWC-120 - Weak Sources of Randomness from Chain Attributes

CWE-330: Use of Insufficiently Random Values

Description:

Solidity allows for ambiguous naming of state variables when inheritance is used. Contract A with a variable x could inherit contract B that also has a state variable x defined. This would result in two separate versions of x, one of them being accessed from contract A and the other one from contract B. In more complex contract systems this condition could go unnoticed and subsequently lead to security issues.

Shadowing state variables can also occur within a single contract when there are multiple definitions on the contract and function level.

Remediation:

Using commitment scheme, e.g. RANDAO. Using external sources of randomness via oracles, e.g. Oraclize. Note that this approach requires trusting in oracle, thus it may be reasonable to use multiple oracles. Using Bitcoin block hashes, as they are more expensive to mine.

References:

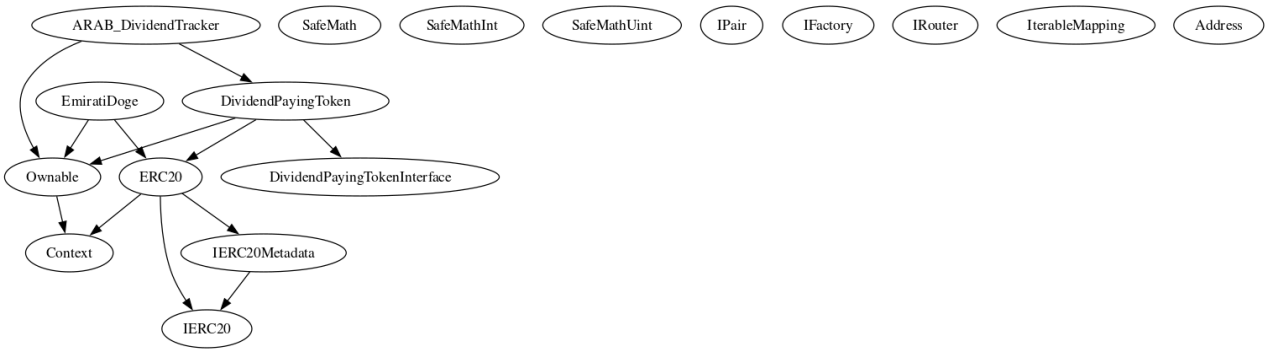
How can I securely generate a random number in my smart contract?)

When can BLOCKHASH be safely used for a random number? When would it be unsafe?

The Run smart contract.

Inheritance

The contract for EmiratiDoge2.0 has the following inheritance structure.



Smart Contract Advance Checks

ID	Severity	Name	Result	Status
$EmirDoge2.0-01	Minor	Potential Sandwich Attacks.	Pass	Not-Found
$EmirDoge2.0-02	Minor	Function Visibility Optimization	Pass	Detected
$EmirDoge2.0-03	Minor	Lack of Input Validation.	Pass	Not-Detected
$EmirDoge2.0-04	Major	Centralized Risk In addLiquidity.	Pass	Not-Detected
$EmirDoge2.0-05	Minor	Missing Event Emission.	Pass	Not-Detected
$EmirDoge2.0-06	Minor	Conformance with Solidity Naming Conventions.	Pass	Not-Detected
$EmirDoge2.0-07	Minor	State Variables could be Declared Constant.	Pass	Not-Found
$EmirDoge2.0-08	Minor	Dead Code Elimination.	Pass	Not-Found
$EmirDoge2.0-09	Major	Third Party Dependencies.	Pass	Not-Found
$EmirDoge2.0-10	Major	Initial Token Distribution.	Pass	Not-Found
$EmirDoge2.0-11	Minor	Multisend is present in code.	Pass	Detected
$EmirDoge2.0-12	Major	Centralization Risks In The X Role	Pass	Not-Found
$EmirDoge2.0-13	Informational	Extra Gas Cost For User..	Pass	Not-Found

ID	Severity	Name	Result	Status
$EmirDoge2.0-6	Medium	Unnecessary Use Of SafeMath	Fail	Detected
$EmirDoge2.0-15	Medium	Symbol Length Limitation due to Solidity Naming Standards.	Pass	Not-Found
$EmirDoge2.0-16	Medium	Invalid collection of Taxes during Transfer.	Pass	Not-Found
$EmirDoge2.0-17	Informational	Conformance to numeric notation best practice.	Pass	Not-Found
$EmirDoge2.0-18	Medium	Stop Transactions by using Enable Trade.	Pass	Not-Detected

$EmirDoge2.0-14 | Unnecessary Use Of SafeMath

Category	Severity	Location	Status
Logical Issue	Medium	EmiratiDoge2_0.sol: 7,9	Detected

Description

The SafeMath library is used unnecessarily. With Solidity compiler versions 0.8.0 or newer, arithmetic operations
will automatically revert in case of integer overflow or underflow.
library SafeMath {
An implementation of SafeMath library is found.
using SafeMath for uint256;
SafeMath library is used for uint256 type in contract.

Remediation

We advise removing the usage of SafeMath library and using the built-in arithmetic operations provided by the
Solidity programming language

Project Action

Technical Findings Summary

Classification of Risk

Severity	Description
Critical	Risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
Major	Risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
Medium	Risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform
Minor	Risks can be any of the above but on a smaller scale. They generally do not compromise the overall integrity of the Project, but they may be less efficient than other solutions.
Informational	Errors are often recommended to improve the code's style or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

Findings

Severity	Found	Pending	Resolved
Critical	0	0	0
Major	1	0	0
Medium	0	0	0
Minor	0	0	0
Informational	0	0	0
Total	1	0	0

Social Media Checks

Social Media	URL	Result
Twitter		Fail
Other		Fail
Website		Fail
Telegram	https://t.me/EmiratiDoge	Pass

We recommend to have 3 or more social media sources including a completed working websites.

Social Media Information Notes:

Auditor Notes: undefined

Project Owner Notes:

Assessment Results

Score Results

Review	Score
Overall Score	84/100
Auditor Score	87/100
Review by Section	**Score**
Manual Scan Score	36/53
SWC Scan Score	34 /37
Advance Check Score	14 /19

The Following Score System Has been Added to this page to help understand the value of the audit, the maximun 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 changed to 80 Points, if a project does not attain 80% is an automatic failure. Read our notes and final assessment below.

Audit Passed

Assessment Results

Important Notes:

- Owner can't set max tx amount.
- No high-risk Exploits/Vulnerabilities Were Found in the Source Code.

Auditor Score =87
Audit Passed

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that actagainst the nature of decentralization, such as explicit ownership or specialized access roles incombination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimalEVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on howblock.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functionsbeing invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that mayresult in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to makethe codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code,such as a constructor assignment imposing different require statements on the input variables than a setterfunction.

Coding Best Practices

ERC 20 Conding Standards are a set of rules that each developer should follow to ensure the code meet a set of creterias and is readable by all the developers.

Disclaimer

CFGNINJA has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocation for the Project, and users relying on this report should not consider this as having any merit for financial advice in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided 'as is, and CFGNINJA is under no covenant to audited completeness, accuracy, or solidity of the contracts. In no event will CFGNINJA or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment services provided by CFGNINJA are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The services may access, and depend upon, multiple layers of third parties.