



CFG NINJA AUDITS

Security Assessment

DUCK Token

May 15, 2023



Table of Contents

1 Assessment Summary

2 Technical Findings Summary

3 Project Overview

3.1 Main Contract Assessed

4 Smart Contract Risk Checks

5 Contract Ownership

7 KYC Check

8 Smart Contract Vulnerability Checks

8.1 Smart Contract Vulnerability Details

8.2 Smart Contract Inheritance Details

8.3 Smart Contract Privileged Functions

9 Assessment Results and Notes(Important)

10 Social Media Check(Informational)

11 Technical Findings Details

12 Disclaimer



Assessment Summary

This report has been prepared for DUCK Token on the Binance Smart Chain network. CFGNINJA provides both client-centered and user-centered examination of the smart contracts and their current status when applicable. This report represents the security assessment made to find issues and vulnerabilities on the source code along with the current liquidity and token holder statistics of the protocol.

A comprehensive examination has been performed, utilizing Cross Referencing, Static Analysis, In-House Security Tools, and line-by-line Manual Review.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Inspecting liquidity and holders statistics to inform the current status to both users and client when applicable.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Verifying contract functions that allow trusted and/or untrusted actors to mint, lock, pause, and transfer assets.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- Thorough line-by-line manual review of the entire codebase by industry experts.








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	0	0	0
 Medium	1	1	0
 Minor	0	0	0
 Informational	0	0	0
Total	3	1	0



Project Overview

Token Summary

Parameter	Result
Address	0x4E4e5086c1f121fc2D4C1b7126025B10f76543A3
Name	DUCK
Token Tracker	DUCK (DUCK)
Decimals	18
Supply	1,000,000,000,000,000
Platform	Binance Smart Chain
compiler	v0.8.19+commit.7dd6d404
Contract Name	Token
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	https://bscscan.com/address/0x4E4e5086c1f121fc2D4C1b7126025B10f76543A3#code
Payment Tx	0xe93ed4807524f5ca48daf430e69f139095783e4c7c6c855faf053578e950cf98



Project Overview

Risk Analysis Summary

Parameter	Result
Buy Tax	5%
Sale Tax	5%
Is honeypot?	Clean
Can edit tax?	Yes
Is anti whale?	No
Is blacklisted?	No
Is whitelisted?	No
Holders	3
Confidence Level	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.



Main Contract Assessed Contract Name

Name	Contract	Live
DUCK	0x4E4e5086c1f121fc2D4C1b7126025B10f76543A3	Yes

TestNet Contract Assessed Contract Name

Name	Contract	Live
DUCK	0x2f7EF8225Fc4b02C3Ce88CFC52AD67A3F529611B	Yes

Solidity Code Provided

SolID	File Sha-1	FileName
DUCK	96b49eec693ec37a443c5b86ee641f9db965df5d	DUCK.sol



Mint Check

The project owners of DUCK do not have a mint function in the contract, owner cannot mint tokens after initial deploy.

The Project has a Total Supply of 1,000,000,000,000,000 and cannot mint any more than the Max Supply.

Mint Notes:

Auditor Notes:

Project Owner Notes:



Owner can't mint new coins



Fees Check

The project owners of DUCK have the ability to set higher than 25%

We Recommend the team to create a new contract with fees restrictions to avoid any problems, as alternative the team can use multi signature wallet to ensure the project is safe from a potential fee increase.

Tax Fee Notes:

Auditor Notes: The contract currently has 5% buy and 5% sale taxes, and cannot be set higher than 25%.

Project Owner Notes:



**Fees can be
changed above
25%**



Blacklist Check

The project owners of DUCK do not have a blacklist function their contract.

The Project 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.

Blacklist Notes:

Auditor Notes:

Project Owner Notes: undefined



MaxTx Check

The Project Owners of DUCK cannot set max tx amount

The Team allows any investors to swap, transfer or sell their total amount if needed.

MaxTX Notes:

Auditor Notes:

Project Owner Notes:

Project Has No MaxTX



Pause Trade Check

The Project Owners of DUCK can stop or pause trading

We recommend the Team only allow Open Trade and never use Stop Trade, as this will be catastrophic for the Project and Investors.

We recommend the Team create a new contract without the stop trade function.

Pause Trade Notes:

Auditor Notes: There is an Open Trade so holders cant trade until is enable.

Project Owner Notes:

Owner can pause trading



Contract Ownership

The contract ownership of DUCK 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.

The current owner is the address
0xea95784312c9bf9318231a8bd840416842ff1526
which can be viewed:
[HERE](#)

The owner wallet has the power to call the functions displayed on the privileged functions chart below, if the owner's wallet is compromised, they could exploit these privileges.

We recommend the team renounce ownership at the right time, if possible, or gradually migrate to a timelock with governing functionalities regarding transparency and safety considerations.

We recommend the team use a Multisignature Wallet if the contract is not going to be renounced; this will give the team more control over the contract.



Liquidity Ownership

The token does not have liquidity at the moment of the audit, block 28222459

If liquidity is unlocked, then the token developers can do what is infamously known as 'rugpull'. 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). This is because investors are basically sending these tokens of value to the exchange, to get the new token. 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 time 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.

[Read More](#)



KYC Information

The Project Owners of DUCK is not KYC.

KYC Information Notes:

Auditor Notes:

Project Owner Notes:



Smart Contract Vulnerability Checks

ID	Severity	Name	File	location
SWC-100	Pass	Function Default Visibility	DUCK.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	DUCK.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	DUCK.sol	L: 0 C: 0
SWC-103	Low	A floating pragma is set.	DUCK.sol	L: 2 C: 1
SWC-104	Pass	Unchecked Call Return Value.	DUCK.sol	L: 0 C: 0
SWC-105	Pass	Unprotected Ether Withdrawal.	DUCK.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	DUCK.sol	L: 0 C: 0
SWC-107	Pass	Read of persistent state following external call.	DUCK.sol	L: 0 C: 0



ID	Severity	Name	File	location
SWC-108	Low	State variable visibility is not set..	DUCK.sol	L: 83 C: 97, L: 83 C: 118, L: 83 C: 139, L: 83 C: 160, L: 83 C: 7732, L: 83 C: 7754, L: 83 C: 8093, L: 83 C: 8146, L: 83 C: 8621, L: 83 C: 8721, L: 83 C: 8739, L: 83 C: 8764, L: 83 C: 8783, L: 83 C: 18121, L: 83 C: 18191, L: 83 C: 18225
SWC-109	Pass	Uninitialized Storage Pointer.	DUCK.sol	L: 0 C: 0
SWC-110	Pass	Assert Violation.	DUCK.sol	L: 0 C: 0
SWC-111	Pass	Use of Deprecated Solidity Functions.	DUCK.sol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	DUCK.sol	L: 0 C: 0
SWC-113	Pass	Multiple calls are executed in the same transaction.	DUCK.sol	L: 0 C: 0
SWC-114	Pass	Transaction Order Dependence.	DUCK.sol	L: 0 C: 0



ID	Severity	Name	File	location
SWC-115	Pass	Authorization through tx.origin.	DUCK.sol	L: 0 C: 0
SWC-116	Pass	A control flow decision is made based on The block.timestamp environment variable.	DUCK.sol	L: 0 C: 0
SWC-117	Pass	Signature Malleability.	DUCK.sol	L: 0 C: 0
SWC-118	Pass	Incorrect Constructor Name.	DUCK.sol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	DUCK.sol	L: 0 C: 0
SWC-120	Pass	Potential use of block.number as source of randomness.	DUCK.sol	L: 0 C: 0
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	DUCK.sol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	DUCK.sol	L: 0 C: 0
SWC-123	Pass	Requirement Violation.	DUCK.sol	L: 0 C: 0
SWC-124	Pass	Write to Arbitrary Storage Location.	DUCK.sol	L: 0 C: 0
SWC-125	Pass	Incorrect Inheritance Order.	DUCK.sol	L: 0 C: 0
SWC-126	Pass	Insufficient Gas Griefing.	DUCK.sol	L: 0 C: 0
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	DUCK.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	DUCK.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	DUCK.sol	L: 0 C: 0



ID	Severity	Name	File	location
SWC-130	Pass	Right-To-Left-Override control character (U+202E).	DUCK.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	DUCK.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	DUCK.sol	L: 0 C: 0
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	DUCK.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	DUCK.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/Dead Code).	DUCK.sol	L: 0 C: 0
SWC-136	Pass	Unencrypted Private Data On-Chain.	DUCK.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-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.

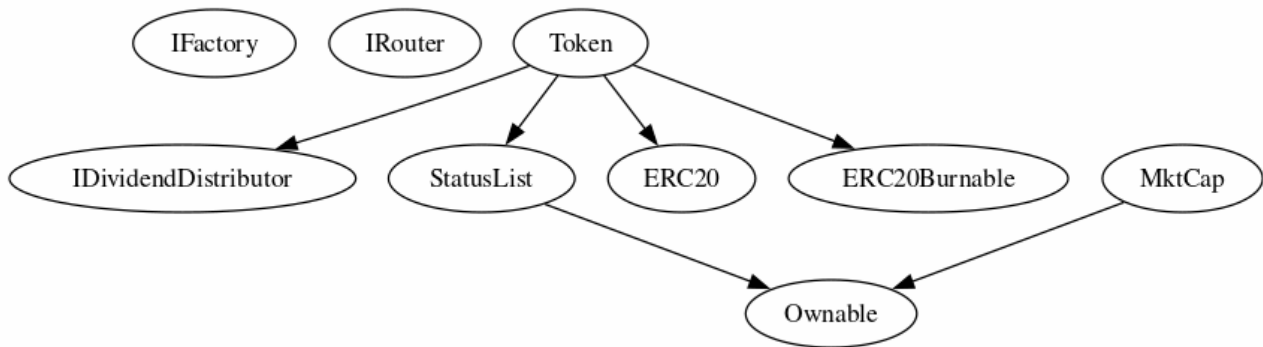
References:

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



Inheritance

The contract for DUCK has the following inheritance structure.



Privileged Functions (onlyOwner)

Function Name	Parameters	Visibility
setAll		public
setAutoSellConfig		public
setAllot		public
setBasePair		public
setMarketing		public
send		public
setStatus		public
setDistributionCriteria		external
setopenDividends		external
setFees		public
setExDividend		public
setinb		public





Smart Contract Advance Checks

ID	Severity	Name	Result	Status
DUCK-01	Minor	Potential Sandwich Attacks.	Pass	Not found
DUCK-02	Minor	Function Visibility Optimization	Fail	Not found
DUCK-03	Minor	Lack of Input Validation.	Fail	Not found
DUCK-04	Major	Centralized Risk In addLiquidity.	Pass	Not found
DUCK-05	Major	Missing Event Emission.	Fail	Not found
DUCK-06	Minor	Conformance with Solidity Naming Conventions.	Pass	Not Found
DUCK-07	Minor	State Variables could be Declared Constant.	Pass	Not found
DUCK-08	Major	Dead Code Elimination.	Pass	Not-Found
DUCK-09	Major	Third Party Dependencies.	Pass	Not Found
DUCK-10	Major	Initial Token Distribution.	Pass	Not found
DUCK-11	Critical	distributeTokensBetween Holders is a multisender of tokens from contract.	Fail	Not found
DUCK-12	Major	Centralization Risks In The X Role	Pass	Not Found
DUCK-13	Informational	Extra Gas Cost For User..	Pass	Not found
DUCK-14	Medium	Unnecessary Use Of SafeMath	Fail	Pending



DUCK-02 | Function Visibility Optimization.

Category	Severity	Location	Status
Gas Optimization	 Minor	DUCK.sol:	 Not found

Description

The following functions are declared as public and are not invoked in any of the contracts contained within the projects scope:

Function Name	Parameters	Visibility
updateDividendTracker		external
updateRouter		external
excludeFromFees		external
excludeMultipleAccountsFromFees		external
setSwapTokensAtAmount		external
setAutomatedMarketMakerPair		external
updateGasForProcessing		external

The functions that are never called internally within the contract should have external visibility

Remediation



We advise that the function's visibility specifiers are set to external, and the array-based arguments change their data location from memory to calldata, optimizing the gas cost of the function.

References:

external vs public best practices.



DUCK-03 | Lack of Input Validation.

Category	Severity	Location	Status
Volatile Code	 Minor	DUCK.sol: 69,9	 Not found

Description

The given input is missing the check for the non-zero address.

The given input is missing the check for the `setExcludedFromFeesStatus`, `setCollection` and `setThresholdAmount` is missing required function.

Remediation



We advise the client to add the check for the passed-in values to prevent unexpected errors as below:

```
...  
    require(receiver != address(0), "Receiver is the zero address");  
...
```

We also recommend customer to review the following function that is missing a required validation. `setExcludedFromFeesStatus`, `setCollection` and `setThresholdAmount` is missing required function.



DUCK-05 | Missing Event Emission.

Category	Severity	Location	Status
Volatile Code	 Major	DUCK.sol: 423,14	 Not found

Description



Detected missing events for critical arithmetic parameters. There are functions that have no event emitted, so it is difficult to track off-chain changes. The linked code does not create an event for the transfer.

Remediation

Emit an event for critical parameter changes. It is recommended emitting events for the sensitive functions that are controlled by centralization roles.



DUCK-11 | distributeTokensBetweenHolders is a multisender of tokens from contract..

Category	Severity	Location	Status
Security	 Critical	DUCK.sol: 103,14	 Not found

Description

Unnecessary use of SatusList to control transfers



Remediation

Remove Satus Library and use normal transfer function

Project Action



DUCK-14 | Unnecessary Use Of SafeMath

Category	Severity	Location	Status
Logical Issue	 Medium	DUCK.sol: 20, 11	 Pending

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.

_balances[recipient] = _balances[recipient].add(amount);

magnifiedDividendPerShare = magnifiedDividendPerShare.add(
(amount).mul(magnitude) / totalSupply()

);

Note: Only a sample of 2 SafeMath library usage in this contract (out of 14) are shown above.

Remediation

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

Solidity programming language

Project Action



Social Media Checks

Social Media	URL	Result
Twitter	https://twitter.com/kristinaolano?s=09	Pass
Other	https://duckcoin.world/DUCK.whitepaper.pdf	Pass
Website	https://duckcoin.world/	Pass
Telegram	https://t.me/China88DUCK	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:



Audit Result

Final Audit Score

Review	Score
Security Score	60
Auditor Score	60

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 must 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 Fail



Assessment Results

Important Notes:

- Contract has taxes up to 100%.
- High-risk Exploits/Vulnerabilities Were Found in the Source Code.

Auditor Score =60
Audit Fail



Appendix

Finding Categories

Centralization / Privilege

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

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM 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 how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being 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 may result in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the 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 requirements on the input variables than a setter function.

Coding Best Practices

ERC 20 Coding Standards are a set of rules that each developer should follow to ensure the code meets a set of criteria 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 advocacy 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.

