CFG NINJA AUDITS

Security Assessment

MERA Contract

April 22, 2023

Audit Status: Pass

Audit Edition: Pinksale

Table of Contents

Assessment Summary

This report has been prepared for MERA Contract 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.

Project Overview

Token Summary

Parameter	Result
Address	0x46693225c6edf6298b22Ad2117Cc68249e204Ed4
Name	MERA
Token Tracker	MERA (MERA)
Decimals	18
Supply	1,000,000,000
Platform	Binance Smart Chain
compiler	v0.8.4+commit.c7e474f2
Contract Name	StandardToken
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	https://bscscan.com/address/0x46693225c6edf6298b22Ad2117Cc68249e204Ed4#code
Payment Tx	0x7c01448176f2e9c940183803a4b648038cc99df24f0b796575b3ddf549ebe89f

Project Overview

Simulation Summary

Parameter	Result
Transfer From Owner	Pass
Transfer From Holder	Pass
Add Liquidity	Pass
Buy from Owner	Pass
Buy from Holder	Pass
Remove Liquidity	Pass
SwapAndLiquify	Pass
RemoveLiquidity	Pass
LaunchPad	PinkSale

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
MERA	0x46693225c6edf6298b22Ad2117Cc68249e204Ed4	Yes

TestNet Contract was Not Assessed

Solidity Code Provided

SolID	File Sha-1	FileName
Web3FarmingBNB	92d2a10e4bb51a3bf8b45bf99c14de81566a52f8	web3farmingbnb.sol
Web3FarmingBNB		
Web3FarmingBNB		
Web3FarmingBNB		
Web3FarmingBNB		
Web3FarmingBNB		

Mint Check

The project owners of MERA 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 and cannot mint any more than the Max Supply.

Mint Notes:

Auditor Notes:

Project Owner Notes:

Fees Check

The project owners of MERA do not have the ability to set fees higher than 25% .

The team May have fees defined; however, they can't set those fees higher than 25% or may not be able to configure the same.

Tax Fee Notes:

Auditor Notes: The contract currently has 10% buy and 10% sale taxes, and cannot be set.

Project Owner Notes:

Fees Can Be Changed up to a maximum of 25%

Blacklist Check

The project owners of MERA 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:

MaxTx Check

The Project Owners of MERA 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 MERA don't have the ability to stop or pause trading.

The Team has done a great job to avoid stop trading, and investors has the ability to trade at any given time without any problems

Pause Trade Notes:

Auditor Notes:

Project Owner Notes: .

Owner can't pause trading

Contract Ownership

The contract ownership of MERA 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 0x56a82a44871559c77c4ababbb5aad728b2b8d074 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

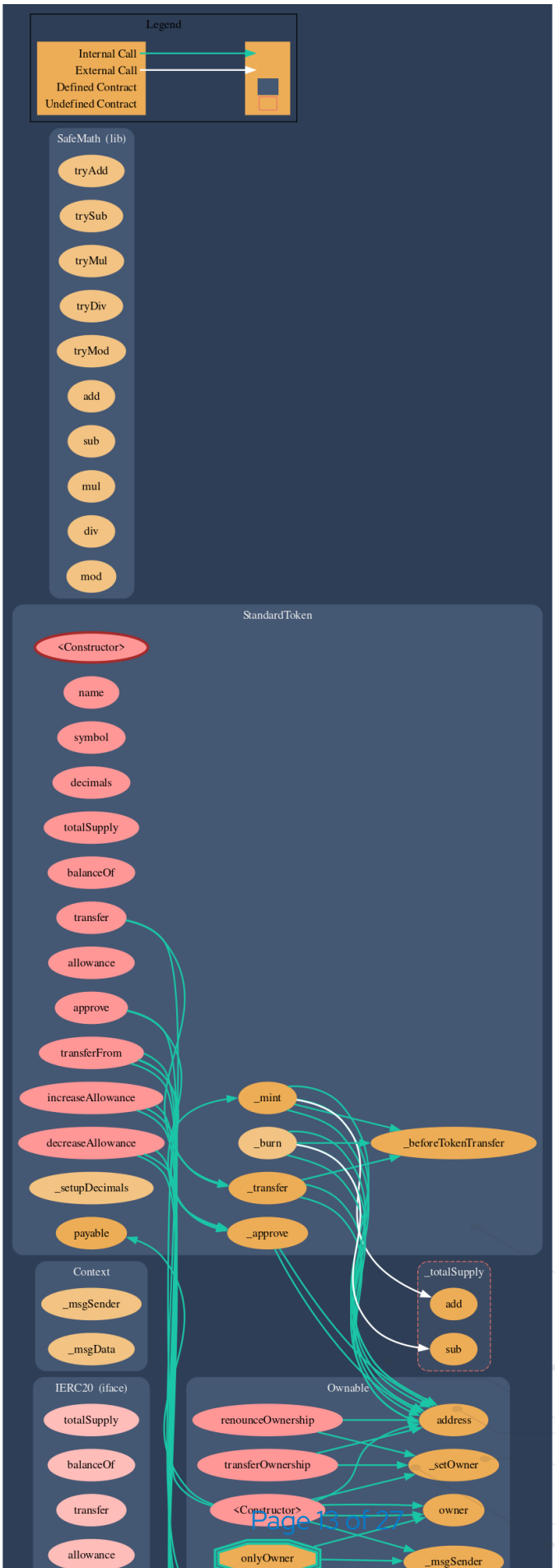
The token does not have liquidity at the moment of the audit, block 27492896

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

Call Graph

The contract for MERA has the following call graph structure.



KYC Information

The Project Owners of MERA is not KYC.

KYC Information Notes:

Auditor Notes: KYC to be completed by PinkSale, project will be a SAFU Project.

Project Owner Notes:

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	StandardToken.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	StandardToken.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	StandardToken.sol	L: 0 C: 0
SWC-103	Low	A floating pragma is set.	StandardToken.sol	L: 10 C: 0
SWC-104	Pass	Unchecked Call Return Value.	StandardToken.sol	L: 0 C: 0
SWC-105	Pass	Unprotected Ether Withdrawal.	StandardToken.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	StandardToken.sol	L: 0 C: 0
SWC-107	Low	Read of persistent state following external call.	StandardToken.sol	L: 1274 C: 26
SWC-108	Pass	State variable visibility is not set..	StandardToken.sol	L: 0 C: 0
SWC-109	Pass	Uninitialized Storage Pointer.	StandardToken.sol	L: 0 C: 0
SWC-110	Pass	Assert Violation.	StandardToken.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-111	Pass	Use of Deprecated Solidity Functions.	StandardToken.sol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	StandardToken.sol	L: 0 C: 0
SWC-113	Pass	Multiple calls are executed in the same transaction.	StandardToken.sol	L: 0 C: 0
SWC-114	Pass	Transaction Order Dependence.	StandardToken.sol	L: 0 C: 0
SWC-115	Pass	Authorization through tx.origin.	StandardToken.sol	L: 0 C: 0
SWC-116	Low	A control flow decision is made based on The block.timestamp environment variable.	StandardToken.sol	L: 1234 C: 12
SWC-117	Pass	Signature Malleability.	StandardToken.sol	L: 0 C: 0
SWC-118	Pass	Incorrect Constructor Name.	StandardToken.sol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	StandardToken.sol	L: 0 C: 0
SWC-120	Pass	Potential use of block.number as source of randonmness.	StandardToken.sol	L: 0 C: 0
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	StandardToken.sol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	StandardToken.sol	L: 0 C: 0
SWC-123	Low	Requirement Violation.	StandardToken.sol	L: 1089 C: 8
SWC-124	Pass	Write to Arbitrary Storage Location.	StandardToken.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-125	Pass	Incorrect Inheritance Order.	StandardToken.sol	L: 0 C: 0
SWC-126	Pass	Insufficient Gas Griefing.	StandardToken.sol	L: 0 C: 0
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	StandardToken.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	StandardToken.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	StandardToken.sol	L: 0 C: 0
SWC-130	Pass	Right-To-Left-Override control character (U +202E).	StandardToken.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	StandardToken.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	StandardToken.sol	L: 0 C: 0
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	StandardToken.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	StandardToken.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/Dead Code).	StandardToken.sol	L: 0 C: 0
SWC-136	Pass	Unencrypted Private Data On-Chain.	StandardToken.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-107 - Reentrancy.

CWE-841: Improper Enforcement of Behavioral Workflow.

Description:

One of the major dangers of calling external contracts is that they can take over the control flow. In the reentrancy attack (a.k.a. recursive call attack), a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways.

Remediation:

The best practices to avoid Reentrancy weaknesses are: Make sure all internal state changes are performed before the call is executed. This is known as the Checks-Effects-Interactions pattern Use a reentrancy lock.

References:

Ethereum Smart Contract Best Practices - Reentrancy

Smart Contract Vulnerability Details

SWC-116 - Block values as a proxy for time

CWE-829: Inclusion of Functionality from Untrusted Control Sphere

Description:

Contracts often need access to time values to perform certain types of functionality. Values such as block.timestamp, and block.number can give you a sense of the current time or a time delta, however, they are not safe to use for most purposes.

Remediation:

Developers should write smart contracts with the notion that block values are not precise, and the use of them can lead to unexpected effects. Alternatively, they may make use oracles..

References:

Safety: Timestamp dependence

Ethereum Smart Contract Best Practices - Timestamp Dependence

How do Ethereum mining nodes maintain a time consistent with the network?.

Solidity: Timestamp dependency, is it possible to do safely?.

Avoid using block.number as a timestamp

Smart Contract Vulnerability Details

SWC-123 - Requirement Violation

CWE-573: Improper Following of Specification by Caller

Description:

The Solidity require() construct is meant to validate external inputs of a function. In most cases, such external inputs are provided by callers, but they may also be returned by callees. In the former case, we refer to them as precondition violations. Violations of a requirement can indicate one of two possible issues:

A bug exists in the contract that provided the external input.
The condition used to express the requirement is too strong.

Remediation:

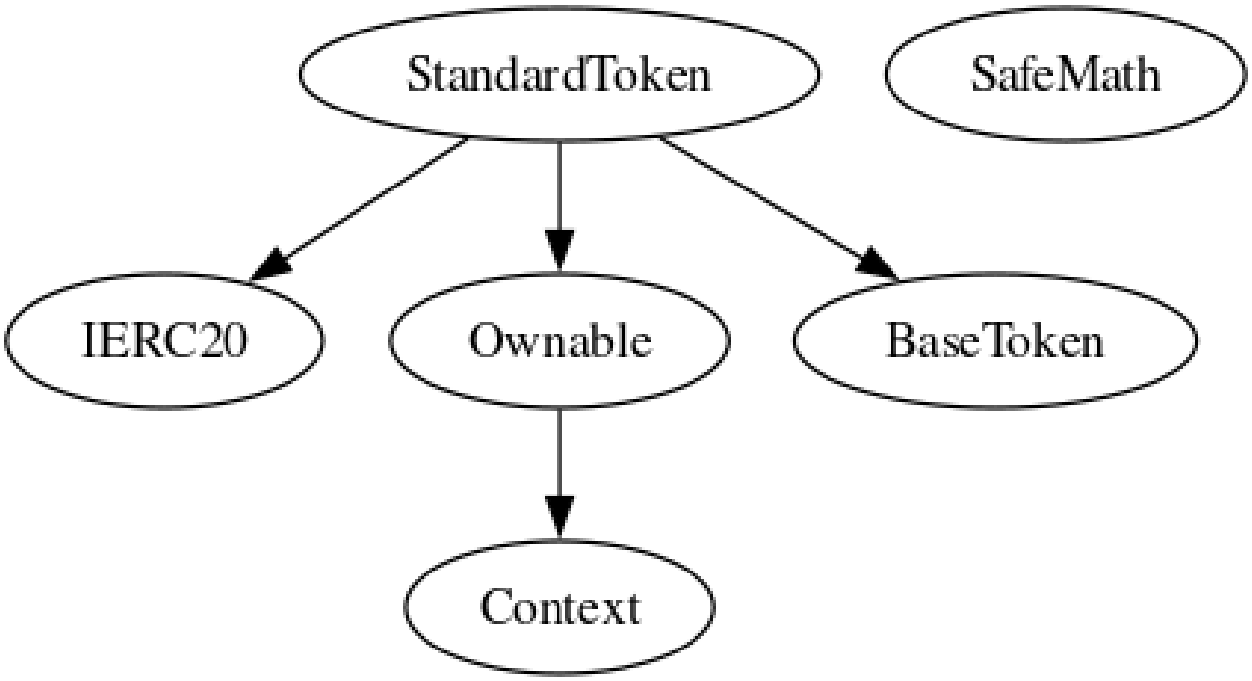
If the required logical condition is too strong, it should be weakened to allow all valid external inputs.Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.

References:

The use of revert(), assert(), and require() in Solidity, and the new REVERT opcode in the EVM

Inheritance

The contract for MERA has the following inheritance structure.



Social Media Checks

Social Media	URL	Result
Twitter	https://twitter.com/merainvest	Pass
Other	http://instagram.com/merainvest/	Pass
Website	https://metaverse.merainvest.com/	Pass
Telegram	https://t.me/merainvestnews	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:

Aduit Result

Final Audit Score

Review	Score
Security Score	85
Auditor Score	85

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:

- No issues or vulnerabilities were found.
- This is a Pinksale Generated Standard token.
- Please DYOR on the project.

Auditor Score =85
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.