Assure DeFi™

The Verification **Gold Standard ™**

Security Assessment

Groks Token

December 18, 2023

Audit Status: Fail

Audit Edition: Standard

ASSURE DEFI™

THE VERIFICATION **GOLD STANDARD**

Risk Analysis

Classifications of Manual Risk Results

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

Manual Code Review Risk Results

Contract Privilege	Description
Buy Tax	5%
Sale Tax	8%
Cannot Sale	Pass
Cannot Sale	Pass
Max Tax	5%
Modify Tax	Detected
Fee Check	Pass
Is Honeypot?	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	Pass
Pause Transfer?	Not Detected
Max Tx?	Pass
Is Anti Whale?	Not Detected
Is Anti Bot?	Not Detected

Contract Privilege	Description
Is Blacklist?	Not Detected
Blacklist Check	Pass
is Whitelist?	Detected
Can Mint?	Pass
Is Proxy?	Not Detected
Can Take Ownership?	Not detected
Hidden Owner?	Not detected
Owner	0x7980Af1494b801895C45B84533596F3F58df152F
Self Destruct?	Not Detected
External Call?	Not detected
Other?	Not detected
Holders	2
Auditor Confidence	Medium Risk
KYC Present	No
KYC URL	

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	0x07bF0cbcBb9230a16A5c134aFCC6b51eD3235D57
Name	Groks
Token Tracker	Groks (Groks)
Decimals	9
Supply	400,000,000,000,000,000
Platform	BNBCHAIN
compiler	v0.8.15+commit.e14f2714
Contract Name	GroksToken
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	https://etherscan.io/token/0x07bF0cbcBb9230a16A5c134aFCC6b51eD3235D57#code
Payment Tx	Corporate

Main Contract Assessed Contract Name

Name	Contract	Live
Groks	0x07bF0cbcBb9230a16A5c134aFCC6b51eD3235D57	Yes

TestNet Contract Assessed Contract Name

Name	Contract	Live
Groks	0xcf8A34306dd73200e35bCDCA2F926C42d1FEc019	Yes

Solidity Code Provided

SolID	File Sha-1	FileName
GROKS	f906c36027cdc845f2a33dca182d38816cbe7c0f	groks.sol
GROKS		
GROKS		
GROKS		

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	groks.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	groks.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	groks.sol	L: 0 C: 0
SWC-103	Low	A floating pragma is set.	groks.sol	L: 7 C: 0
SWC-104	Pass	Unchecked Call Return Value.	groks.sol	L: 0 C: 0
SWC-105	Pass	Unprotected Ether Withdrawal.	groks.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	groks.sol	L: 0 C: 0
SWC-107	Pass	Read of persistent state following external call.	groks.sol	L: 0 C: 0
SWC-108	Low	State variable visibility is not set..	groks.sol	L: 405 C: 9
SWC-109	Pass	Uninitialized Storage Pointer.	groks.sol	L: 0 C: 0
SWC-110	Pass	Assert Violation.	groks.sol	L: 0 C: 0
SWC-111	Pass	Use of Deprecated Solidity Functions.	groks.sol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	groks.sol	L: 0 C: 0
SWC-113	Pass	Multiple calls are executed in the same transaction.	groks.sol	L: 0 C: 0
SWC-114	Pass	Transaction Order Dependence.	groks.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-115	Pass	Authorization through tx.origin.	groks.sol	L: 0 C: 0
SWC-116	Pass	A control flow decision is made based on The block.timestamp environment variable.	groks.sol	L: 0 C: 0
SWC-117	Pass	Signature Malleability.	groks.sol	L: 0 C: 0
SWC-118	Pass	Incorrect Constructor Name.	groks.sol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	groks.sol	L: 0 C: 0
SWC-120	Pass	Potential use of block.number as source of randonmness.	groks.sol	L: 0 C: 0
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	groks.sol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	groks.sol	L: 0 C: 0
SWC-123	Pass	Requirement Violation.	groks.sol	L: 0 C: 0
SWC-124	Pass	Write to Arbitrary Storage Location.	groks.sol	L: 0 C: 0
SWC-125	Pass	Incorrect Inheritance Order.	groks.sol	L: 0 C: 0
SWC-126	Pass	Insufficient Gas Griefing.	groks.sol	L: 0 C: 0
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	groks.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	groks.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	groks.sol	L: 0 C: 0
SWC-130	Pass	Right-To-Left-Override control character (U +202E).	groks.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	groks.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	groks.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	groks.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	groks.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/Dead Code).	groks.sol	L: 0 C: 0
SWC-136	Pass	Unencrypted Private Data On-Chain.	groks.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 Groks has the following inheritance structure.

The Project has a Total Supply of 400,000,000,000,000,000

Privileged Functions (onlyOwner)

Please Note if the contract is Renounced none of this functions can be executed.

Function Name	Parameters	Visibility
excludeFromFee		public
IncludeInFee		public
setTaxFeePercent		external
changeNumTokensSellToFee		external
setSwapAndLiquifyEnabled		external
setMarketingAddress		external
setSellMarketingFeePercent		external
setBuyMarketingFeePercent		external
renounceOwnership		external
transferOwnership		external

Groks-01 | Potential Sandwich Attacks.

Category	Severity	Location	Status
Security	Medium	groks.sol: L:595 C:14	Detected

Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction being attacked) a transaction to purchase one of the assets and make profits by back running (after the transaction being attacked) a transaction to sell the asset. The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

- swapExactTokensForETHSupportingFeeOnTransferTokens()
- addLiquidityETH()

Remediation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the aforementioned functions.

Referrences:

What Are Sandwich Attacks in DeFi — and How Can You Avoid Them?.

Groks-02 | Function Visibility Optimization.

Category	Severity	Location	Status
Gas Optimization	Informational	groks.sol: L: 405 C: 14,L: 512 C: 14,L: 508 C: 14,L: 597 C: 14	Detected

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
inSwapAndLiquify		Internal
renounceOwnership		Internal
transferOwnership		Internal
excludeFromFee		Internal
includeInFee		Internal
setSwapAndLiquifyEnabled		Internal

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.

Groks-03 | Lack of Input Validation.

Category	Severity	Location	Status
Volatile Code	Low	groks.sol: L: 512 C: 14,L: 508 C: 14,L: 497 C: 14	Detected

Description

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

The given input is missing the check for the 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");
...
...
require(value X limitation, "Your not able to do this function");
...
```

We also recommend customer to review the following function that is missing a required validation. missing required function.

Groks-05 | Missing Event Emission.

Category	Severity	Location	Status
Volatile Code	Low	groks.sol: L: 512 C: 14,L: 508 C: 14,L: 497 C: 14,L: 502 C: 14,L: 490 C: 14,L: 484 C: 14,L: 478 C: 14	Detected

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.

Groks-10 | Initial Token Distribution.

Category	Severity	Location	Status
Centralization / Privilege	High	groks.sol: L: 444 C:14	Detected

Description

All of the Groks tokens are sent to the contract deployer when deploying the contract. This could be a centralization risk as the deployer can distribute tokens without obtaining the consensus of the community.

Remediation

We recommend the team to be transparent regarding the initial token distribution process, and the team shall make enough efforts to restrict the access of the private key.

Project Action

emit Transfer(address(0), _msgSender(), _totalSupply);

Groks-13 | Extra Gas Cost For User.

Category	Severity	Location	Status
Logical Issue	Informational	groks.sol: L: 535 C: 14	Detected

Description

The user may trigger a tax distribution during the transfer process, which will cost a lot of gas and it is unfair to let
a single user bear it.

Remediation

We advise the client to make the owner responsible for the gas costs of the tax distribution.

Project Action

Groks-14 | Unnecessary Use Of SafeMath

Category	Severity	Location	Status
Logical Issue	Medium	groks.sol: L: 19 C: 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.
High	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
Low	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
High	1	1	0
Medium	2	2	0
Low	2	2	0
Informational	2	2	0
Total	7	7	0

Social Media Checks

Social Media	URL	Result
Twitter	https://twitter.com/groks_bsc	Pass
Other		Fail
Website	https://groks.finance/	Pass
Telegram	https://t.me/groks_bsc	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	79/100
Auditor Score	79/100
Review by Section	**Score**
Manual Scan Score	22
SWC Scan Score	33
Advance Check Score	24

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 84 Points for a higher standard, if a project does not attain 85% is an automatic failure. Read our notes and final assessment below.

Audit Fail

Assessment Results

Important Notes:

- The contract is using the latest erc20 reverts.

Auditor Score =79
Audit Fail

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

Assure Defi 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 Assure Defi is under no covenant to audited completeness, accuracy, or solidity of the contracts. In no event will Assure Defi 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 Assure Defi 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.