Audit Report - 2

Ext. Report submitted on 24th April 2021

By Nodeberry Private Limited

This report was completed on 28-April-2021. Not an investment advice / a promotion to invest in Kleeka Inu.

Table Of Contents

Executive Summary

Nodeberry, a blockchain research & development company was engaged on April 21, 2021 to perform a comprehensive security review of the Kleeka Inu ERC20 Token smart contract. Our review was conducted over a period of 24 hours (as the client required it to be done quickly) by Sujith Somraaj, the chief executive of Nodeberry.

Nodeberry completed the task using manual, static and dynamic analysis techniques.

Then the audit report 1 was handed over to the development team for fixing the bugs and critical issues. The new code was received through chat communication on 26th of April 2021.

We again did our analysis for 24 hours and submitted our latest report by 27th April 2021.

Timeline

Commencement	22nd April 2021
1st Review	24th April 2021
Final Audit Report	27th April 2021

Engagement Goals

The primary scope of the engagement was to evaluate and establish overall security of the Kleeka Inu ERC20 token contract with a specific focus of vulnerability exploited during the initial days of operation of the contract.

Contract Specification

The contract was not delivered via git, which we usually prefer. We received the codes via bsc scan link :

First Code :

https://testnet.bscscan.com/address/0x241dc984ee69ec9db5792c5b1b6c635d297b3b14#code

Fixed Code:

https://bscscan.com/address/0xeff826e3c63526a683deed47b7246b5713173853#code

Overall Assessment

Nodeberry was engaged to evaluate and identify any potential security concerns within the codebase of the Kleeka Inu Token Contract.

The contract codes written on old compilers which are mostly nightly builds are not appreciated. The business logic was based on invalid keywords and needed to be fixed asap.

There are multiple compilation errors & warnings that need to be fixed before the launch of smart contract on mainnet & further round of audits.

Most of the mentioned issues were fixed by the developer team. The main issue was the usage of chainId() which is to be avoided and also making time based logical operation. But due to the essentiality of those two things, we are not able to avoid them.

There are several gas cost related issues where the functions were consuming large quantities of transactional cost. The issue was addressed to the team. As this engagement is to find the vulnerability within the smart contract, we don’t work too much on gas optimization.

General Recommendations

Unit Testing is suggested

The process of unit testing will enhance the motive of the smart contract. Unit testing will improve the expectations and accuracy of the smart contracts. There is no test folder found in the contract which means it is not unit tested and is highly risky.

Resolution : Due to less time left for launch the Dev Team cannot turn-around with unit testing

Usage of Linters are suggested

The use of linters like sol-hint and other available linters are highly suggested.

Resolution : Solved by the Dev team, in which they've used linters.

Reduce usage of keccak on unknown byte values

The overall dependence on keccak methods can be avoided and in the main contract such instances could be avoided as they are non mandatory.

Resolution : The business logic is based on the keccak function which cannot be avoided - Stated by the Dev Team; *Vulnerability - Less Severe - Medium Severity*

Don't use time based logics for randomness

In the key generation function, time based events are used. The block.timestamp can be altered by miners to some extent so try to find an alternative method to achieve randomness like using VRF.

Resolution : The business logic is based on the keccak function which cannot be avoided - Stated by the Dev Team; *Vulnerability - Less Severe - Medium Severity*

Specific Recommendations

Use of latest version of solidity compiler is recommended

Solidity compilers, follow semantic versioning and these include nightly versions which are under development or broken. So it's suggested to switch or migrate to the latest 0.8.4 solidity version.

Resolution : The compiler version is upgraded to 0.8.4

Reduce Length of Error Messages

In the view of reducing the gas consumption, we suggest using smaller error message in require() functions in #43, #138, #540, #541, #559

Resolution : Error Messages lengths were reduced partially and it doesn't affect the program security wise.

No Override Keyword Found for External Functions

Add the keyword for "external" functions in #337, #344, #351, #358, #365, #370, #380, #391, #402, #427

Resolution : Fixed the overriding and virtual function declaration issues.

Avoid Casting of uint96 to uint256

Avoid the explicit type conversion of int_const -1 to uint256. It has been deprecated in the latest version compilers.

Resolution : It is fixed by ceiling the value to 1 less than the maximum value. Ofc it is not suggested but it is not vulnerable.

Block.Timestamp should be used

The “now” keyword for fetching the current timestamp is deprecated. We suggest using block.timestamp in place of “now” in the contract#490

Resolution : Fixed by upgrading to block.timestamp

Avoid Usage of Inline Assembly

The usage of inline assembly is suggested only in case of absolute requirement. Also, use of ChainId in signature generation is not mandatory & also manipulative by the miners to some extent.

Resolution : Istanbul based compilers support this and this is a mandatory function for the operation of the contract - Dev Team

Avoid Usage of uint256(-1) and uint96(-1) for overflows

In a breaking change by ethereum in compilers after ^0.6.1 it is noted that this method or operation is invalid and no longer supported. It can be replaced by newer methods to contain the out-of-bounds issue.

Resolution : Fixed it by adding a fixed ceil value

Static Analysis Coverage

The contract repository underwent heavy scrutiny with multiple static analysis agents, including:

- Securify
- MAIAN
- Mythril
- Oyente
- Slither

Multiple concerns involving static analysis were identified and itemized above. Slither in particular found INFO:Slither:contracts/ analyzed (22 contracts with 72 detectors), 190 result(s) found. A number of these results were not detailed in the above findings, including the ignoring of return values, user set array length (as it has been mitigated by the existence of an administrative control), alongside numerous style-guide violations.

Compilation Warnings

In previous code there were multiple compilation warnings but in the final one there are no compilation warnings.

Compiler Version : ^ 0.8.4

evmVersion : Istanbul

Output :

```
Compiling your contracts...
===========================
> Compiling ./contracts/Inu.sol
> Artifacts written to /kleeka-inu/build/contracts
> Compiled successfully using:
   - solc: 0.8.4+commit.c7e474f2.Emscripten.clang
```

Consensys Surya (Listing all Methods of the Smart Contract)

+ [Lib] SafeMath
  - [Int] add
  - [Int] add
  - [Int] sub
  - [Int] sub
  - [Int] mul
  - [Int] div
  - [Int] div
  - [Int] mod
  - [Int] mod

+ Context
  - [Int] <Constructor> #
  - [Int] _msgSender
  - [Int] _msgData

+ Ownable (Context)
  - [Int] <Constructor> #
  - [Pub] owner
  - [Pub] renounceOwnership #
    - modifiers: onlyOwner
  - [Pub] transferOwnership #
    - modifiers: onlyOwner
  - [Int] _transferOwnership #

+ [Int] BEP20Interface
  - [Ext] totalSupply
  - [Ext] decimals
  - [Ext] symbol
  - [Ext] name
  - [Ext] getOwner
  - [Ext] balanceOf
  - [Ext] transfer #
  - [Ext] allowance
  - [Ext] approve #
  - [Ext] transferFrom #

+ Tokenlock (Ownable)
  - [Pub] freeze #
    - modifiers: onlyOwner
  - [Pub] unfreeze #
    - modifiers: onlyOwner

+ ApproveAndCallFallBack
  - [Pub] receiveApproval #

+ UserLock (Ownable)
  - [Pub] lockUser #
    - modifiers: onlyOwner
  - [Pub] unlockUser #
    - modifiers: onlyOwner

+ KLEEKA (BEP20Interface, Tokenlock, UserLock)
  - [Pub] <Constructor> #
  - [Ext] getOwner
  - [Ext] decimals
  - [Ext] symbol
  - [Ext] name
  - [Ext] totalSupply
  - [Ext] balanceOf
  - [Ext] transfer #
    - modifiers: validLock,permissionCheck
  - [Ext] allowance
  - [Ext] approve #
    - modifiers: validLock,permissionCheck
  - [Pub] approveAndCall #
    - modifiers: validLock,permissionCheck
  - [Ext] transferFrom #
    - modifiers: validLock,permissionCheck
  - [Pub] increaseAllowance #
    - modifiers: validLock,permissionCheck
  - [Pub] decreaseAllowance #
    - modifiers: validLock,permissionCheck
  - [Pub] burn #
    - modifiers: validLock,permissionCheck

- [Pub] delegate #
  - modifiers: validLock,permissionCheck
- [Pub] delegateBySig #
  - modifiers: validLock,permissionCheck
- [Ext] getCurrentVotes
- [Pub] getPriorVotes
- [Int] _transfer #
- [Int] _approve #
- [Int] _burn #
- [Int] _delegate #
- [Int] _moveDelegates #
- [Int] _writeCheckpoint #
- [Int] safe32
- [Int] ceil96
- [Int] getChainId

($) = payable function
= non-constant function

Disclaimer

This report was completed on 28-April-2021. Not an investment advice / a promotion to invest in Kleeka Inu.

As of the date of publication, the information provided in this report reflects the presently held, commercially reasonable understanding of Nodeberry's knowledge of security patterns as they relate to the Kleeka Inu Contract, with the understanding that distributed ledger technologies ("DLT") remain under frequent and continual development, and resultantly carry with them unknown technical risks and flaws. The scope of the review provided herein is limited solely to items denoted within "Scope of Engagement'' and contained within "Directory Structure". The report does NOT cover, review, or opine upon security considerations unique to the Solidity compiler, tools used in the development of the protocol, or distributed ledger technologies themselves, or to any other matters not specifically covered in this report.

The contents of this report must NOT be construed as investment advice or advise of any other kind. This report does NOT have any bearing upon the potential economics of the Kleeka Inu Token / Project / Protocol or any other relevant product, service or asset of Kleeka Inu or otherwise.

This report is not and should not be relied upon by Kleeka Inu or any reader of this report as any form of financial, tax, legal, regulatory, or other advice.

To the full extent permissible by applicable law, Nodeberry disclaims all warranties, express or implied. The information in this report is provided "as is" without warranty, representation, or guarantee of any kind, including the accuracy of the information provided. Nodeberry makes no warranties, representations, or guarantees about the Kleeka Inu Project.

Use of this report and/or any of the information provided herein is at the users sole risk, and Nodeberry hereby disclaims, and each user of this report hereby waives, releases, and holds Nodeberry harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.

Timeliness of Content

This report was completed on 28-April-2021. Not an investment advice / a promotion to invest in Kleeka Inu.

All content within this report is presented only as of the date published or indicated, to the commercially reasonable knowledge of Nodeberry as of such date, and may be superseded by subsequent events or for other reasons. The content contained within this report is subject to change without notice.

Nodeberry does not guarantee or warrant the accuracy or timeliness of any of the content contained within this report, whether accessed through digital means or otherwise.

Nodeberry is not responsible for setting individual browser cache settings nor can it ensure any parties beyond those individuals directly listed within this report are receiving the most recent content as reasonably understood by Nodeberry as of the date that a report is provided to such individuals.