verichains

SECURITY AUDIT OF

NAGA TOKEN AND TOKENVESTING SMART CONTRACTS

Public Report

Mar 23, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Mar 23, 2022. We would like to thank the Naga for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Naga Token and TokenVesting Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified one vulnerable issue in the smart contracts code. Naga team has resolved and updated all the recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Naga Token and TokenVesting Smart Contracts

Naga is the spirit of the nature, protecting streams, wells and rivers, Naga snake is a symbol of prosperity, god of the crop protection, bringing water to irrigate fields. The Naga snake is also a symbol of the connection between the human realm and nirvana. In Khmer legend, the Naga snake represents the god Shiva.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of Naga Token and TokenVesting Smart Contracts. It was conducted on commit 9e5dc282576bb3c2ad5b0f4dc057eb465602c7e8 from git repository *https://github.com/bnwgithub/smart-contract*.

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The following files were made available in the course of the review: TokenVesting.sol and CoinToken.sol.

2.2. Contract code

The Naga Token and TokenVesting Smart Contracts was written in Solidity language, with the required version to be ^0.8.9. The source code was written based on OpenZeppelin's library.

2.2.1. CoinToken contract

CoinToken contract is an ERC20 token contract. The contract inherits the AccessControl, Pausable, BEP20Snapshot, and Blacklist contracts. AccessControl allows the contract to implement role-based access control mechanisms. There are 4 roles: DEFAULT_ADMIN_ROLE, ADMIN_ROLE, BURNER_ROLE and PAUSER_ROLE. The user has PAUSER_ROLE may use the pause/unpause function from Pausable to control the activities of the contract. Users can only transfer tokens when the contract is not paused. BEP20Snapshot is a contract reference to ERC20Snapshot which help ADMIN_ROLE user take a snapshot of the balances and total supply at a time for later access. The Blacklist contract implements a bunch of functions which help ADMIN_ROLE to limit some suspected users. The contract also uses a BP contract to avoid Whale trading to control the price of tokens in the IDO time.

2.2.2. TokenVesting contract

The Naga team use this contract to release the tokens that investors bought in the past. The tokens of investors will be released in 2 phases. In the first phase in the TGE time, the investors receive the number of tokens that the Naga team set. In the second phase, tokens will be released following the linear logic every secondsPerSlice seconds.

The contract implements withdraw public function which allows the owner of the contract may transfer all tokens in the contract to any address wallet. The contract also inherits the Blacklist contract which allows the owner of the contract may insert the suspected users into the blacklist and avoid them from releasing the tokens.

2.3. Findings

During the audit process, the audit team found one vulnerability in the given version of Naga Token and TokenVesting Smart Contracts.

2.3.1. TokenVesting.sol - Missing check the received users in _computeReleasableAmount function CRITICAL

There are 2 flows in the _computeReleasableAmount function missing check the received tokens. During the cliff time and the first duration, the user can call the release function to trigger this function which returns the tokens without checking whether the user has claimed or not. Therefore, the users can receive the tokens many times with the amount of the token larger than the vestingAmount value.

```
340  function _computeReleasableAmount(VestingSchedule memory vestingSche…
     dule)
341      internal
342      view
343      returns(uint256){
344          uint256 currentTime = getCurrentTime();
345          if (vestingSchedule.revoked == true) {
346              return 0;
347          } else if (currentTime < vestingSchedule.cliff) {
348              if (vestingSchedule.tge > 0 &&  currentTime >= vestingSc…
     hedule.start) {
349                  return vestingSchedule.tge;
350              }
351              return 0;
352          } else if (currentTime >= vestingSchedule.start.add(vestingS…
     chedule.duration)) {
353              return vestingSchedule.amountTotal.sub(vestingSchedule.r…
     eleased);
354          } else if (currentTime >= vestingSchedule.cliff.add(vestingS…
     chedule.firstDuration)
355              && currentTime < vestingSchedule.cliff.add(vestingSchedu…
     le.firstDuration).add(vestingSchedule.slicePeriodSeconds)) {
356              return vestingSchedule.first;
357          } else {
358              uint256 newStart = vestingSchedule.cliff.add(vestingSche…
     dule.firstDuration);
359              uint256 totalWithoutTge = vestingSchedule.amountTotal.su…
     b(vestingSchedule.tge).sub(vestingSchedule.first);
360              uint256 timeFromStart = currentTime.sub(newStart);
361              uint secondsPerSlice = vestingSchedule.slicePeriodSecond…
     s;
362              uint256 vestedSlicePeriods = timeFromStart.div(secondsPe…
```

```
    rSlice);
363             uint256 vestedSeconds = vestedSlicePeriods.mul(secondsPe…
    rSlice);
364             uint256 newDuration = vestingSchedule.start.add(vestingS…
    chedule.duration).sub(newStart);
365             uint256 vestedAmount = totalWithoutTge.mul(vestedSeconds…
    ).div(newDuration) + vestingSchedule.tge + vestingSchedule.firs…
    t;
366             vestedAmount = vestedAmount.sub(vestingSchedule.released…
    );
367             return vestedAmount;
368         }
369     }
```

Snippet 1. TokenVesting.sol Missing check the received users in `_computeReleasableAmount` function

RECOMMENDATION

We suggest updating the function like the below code:

```
function _computeReleasableAmount(VestingSchedule memory vestingSchedule)
    internal
    view
    returns(uint256){
        uint256 currentTime = getCurrentTime();
        if (vestingSchedule.revoked == true) {
            return 0;
        } else if (currentTime < vestingSchedule.cliff) {
            if (vestingSchedule.tge > 0 &&  currentTime >= vestingSchedul…
  e.start) {
                return vestingSchedule.tge - vestingSchedule.released ;
            }
            return 0;
        } else if (currentTime >= vestingSchedule.start.add(vestingSchedu…
  le.duration)) {
            return vestingSchedule.amountTotal.sub(vestingSchedule.releas…
  ed);
        } else if (currentTime >= vestingSchedule.cliff.add(vestingSchedu…
  le.firstDuration)
            && currentTime < vestingSchedule.cliff.add(vestingSchedule.fi…
  rstDuration).add(vestingSchedule.slicePeriodSeconds)) {
            return vestingSchedule.first + vestingSchedule.tge - vestingS…
```

```
chedule.released;
      } else {
          uint256 newStart = vestingSchedule.cliff.add(vestingSchedule.…
firstDuration);
          uint256 totalWithoutTge = vestingSchedule.amountTotal.sub(ves…
tingSchedule.tge).sub(vestingSchedule.first);
          uint256 timeFromStart = currentTime.sub(newStart);
          uint secondsPerSlice = vestingSchedule.slicePeriodSeconds;
          uint256 vestedSlicePeriods = timeFromStart.div(secondsPerSlic…
e);
          uint256 vestedSeconds = vestedSlicePeriods.mul(secondsPerSlic…
e);
          uint256 newDuration = vestingSchedule.start.add(vestingSchedu…
le.duration).sub(newStart);
          uint256 vestedAmount = totalWithoutTge.mul(vestedSeconds).div…
(newDuration) + vestingSchedule.tge + vestingSchedule.first;
          vestedAmount = vestedAmount.sub(vestingSchedule.released);
          return vestedAmount;
      }
  }
```

Snippet 2. TokenVesting.sol Recommend fixing in `_computeReleasableAmount` function

UPDATES

- *Feb 28,2022*: This issue has been acknowledged and fixed by the Naga team.

2.4. Additional notes and recommendations

2.4.1. Unnecessary usage of SafeMath library in Solidity 0.8.0+ INFORMATIVE

All safe math usage in the contract are for overflow checking, solidity 0.8.0+ already do that by default, the only usage of safemath now is to have a custom revert message which isn't the case in the auditing contracts. We suggest using normal operators for readability and gas saving.

RECOMMENDATION

We suggest changing all methods from SafeMath library to the normal arithmetic operator in the contract and removing the SafeMath import statement.

UPDATES

- *Feb 28,2022*: This issue has been acknowledged and fixed by the Naga team.

2.4.2. CoinToken.sol - BPContract function INFORMATIVE

Since we do not control the logic of the BPContract, there is no guarantee that BPContract will not contain any security related issues. With the current context, in case the BPContract is compromised, there is not yet a way to exploit the Naga Token and TokenVesting Smart Contracts, but we still note that here as a warning for avoiding any related issue in the future.

By the way, if having any issue, the BPContract function can be easily disabled anytime by the contract owner using the setBpEnabled function. In addition, BPContract is only used in a short time in token public sale IDO then the contract owner will disable it forever by the setBotProtectionDisableForever function.

UPDATES

- *Feb 28,2022*: This issue has been acknowledged by the Naga team.

APPENDIX

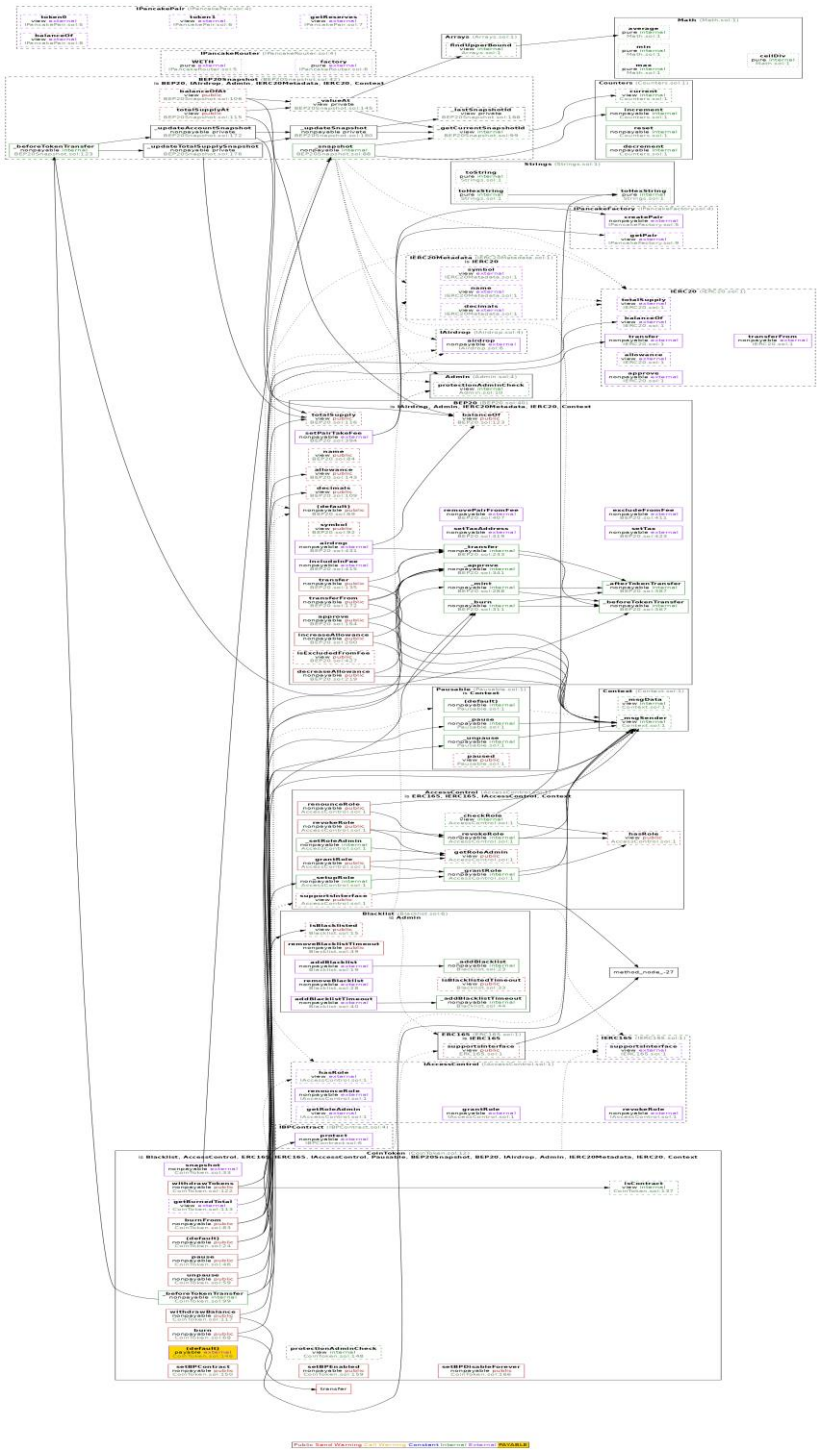

Image 1. Naga Token Smart contract call graph

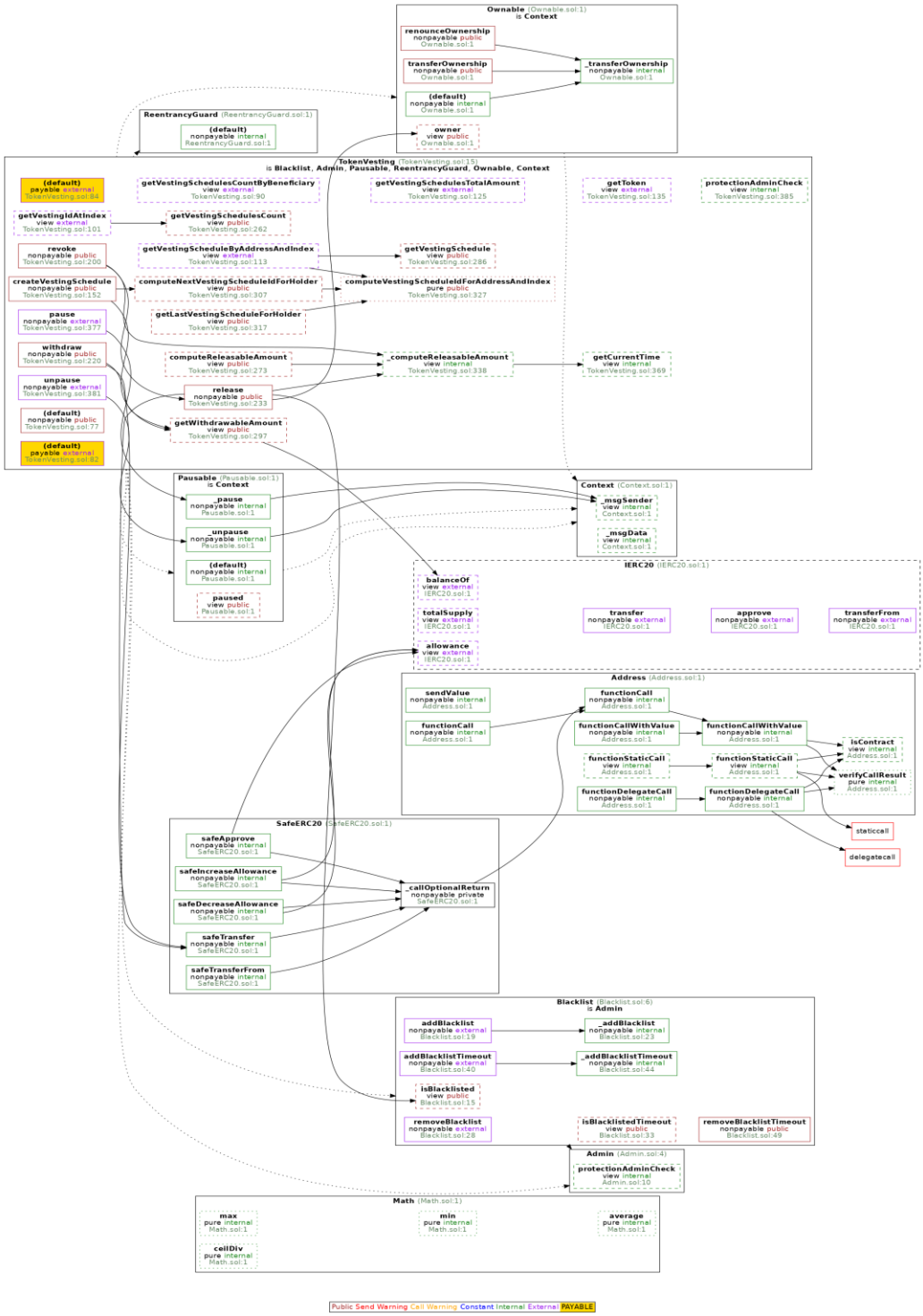

Image 2. Naga TokenVesting Smart contract call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Feb 23, 2022*	Private Report	Verichains Lab
1.1	*Feb 28, 2022*	Public Report	Verichains Lab
1.2	*Mar 23, 2022*	Public Report	Verichains Lab

Table 2. Report versions history