QuillAudits

Audit Report
July, 2021

Contents

Scope of Audit

SCCNToken by Succession
Succession (SCCN) is a deployed ERC20 token associated with the Succession project. SCCN will be used as a cryptocurrency that will help facilitate functions associated with a dapp-wallet that will allow crypto owners to name beneficiaries.
Name: SUCCESSION
Symbol: SCCN
Decimals: 18
Total Supply: 10000 Million

Scope of Audit

The scope of this audit was to analyse SCCNToken smart contract's codebase for quality, security, and correctness.
Commit: 6d04e55dd8a0f40a59dbbc77d5e7eb7d790cc67b

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC20 transfer() does not return boolean
- ERC20 approve() race
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.
SmartCheck.

Static Analysis
Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step a series of automated tools are used to test security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

Gas Consumption
In this step we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

High severity issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.

Medium level severity issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.

Low level severity issues

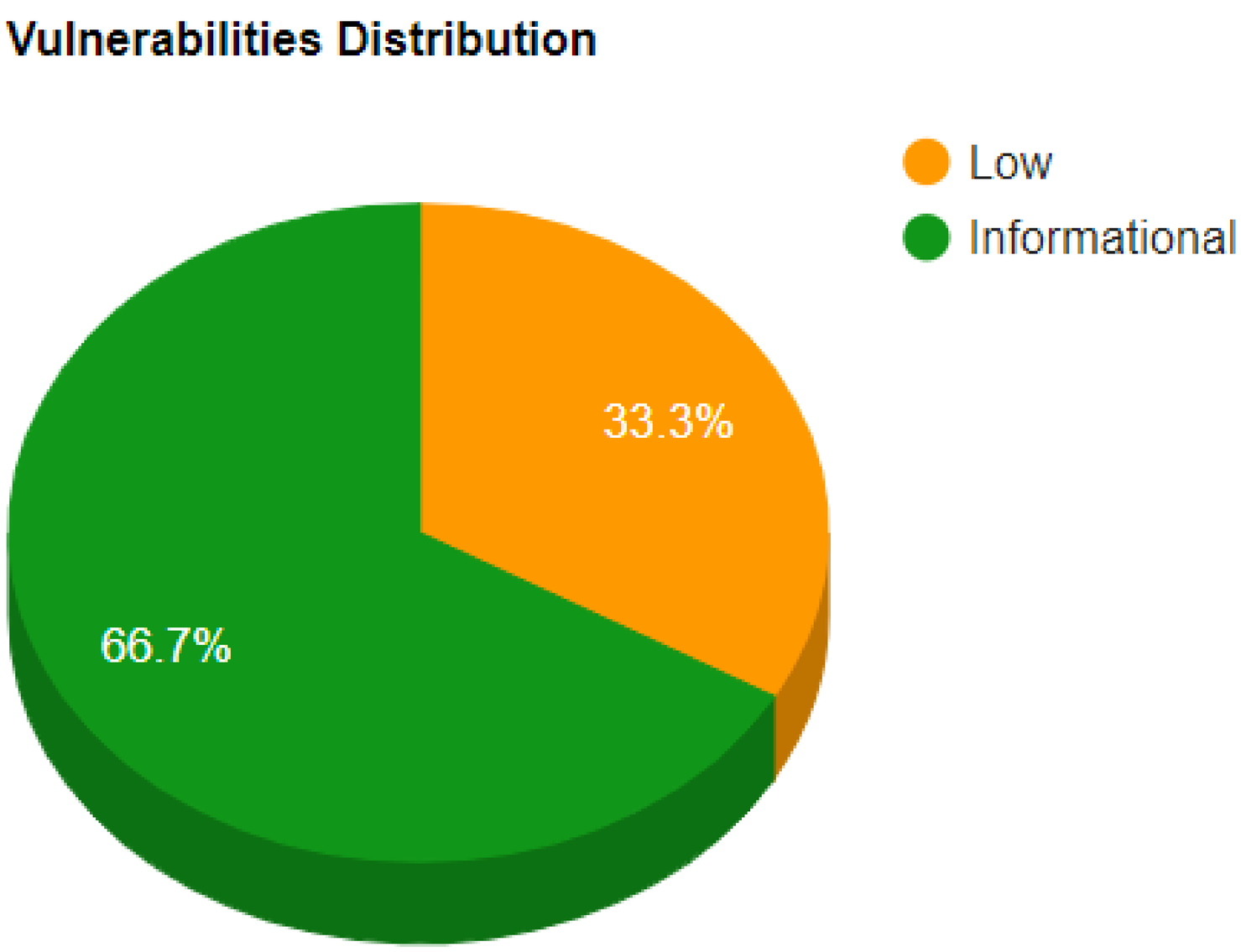
Low level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	2	4
Closed	0	0	0	0



Issues Found – Code Review / Manual Testing

High severity issues

No issues were found

Medium severity issues

No issues were found

Low level severity issues

1. An old compiler has been used. Use the latest compiler so as to avoid bugs introduced in older versions.

2. Multiple Pragma Directives have been used. Use one solidity compiler.

Informational

1. Contract SafeMath implements Public Functions to avoid overflows/ underflows. As their scope is limited to the contract itself, they can be made internal.

```
22  contract SafeMath {
23
24      function safeAdd(uint a, uint b) public pure returns (uint c) {
25          c = a + b;
26          require(c >= a);
27      }
28
29      function safeSub(uint a, uint b) public pure returns (uint c) {
30          require(b <= a);
31          c = a - b;
32      }
33
34      function safeMul(uint a, uint b) public pure returns (uint c) {
35          c = a * b;
36          require(a == 0 || c / a == b);
37      }
38
39      function safeDiv(uint a, uint b) public pure returns (uint c) {
40          require(b > 0);
41          c = a / b;
42      }
43  }
```

2. Add error messages for require statements, so as to easily track down the root cause of errors.

3. The complete totalSupply is added to the balance of the Owner Account (as mentioned in the contract), which can lead to a loss of tokens if account keys are lost.

```
87        constructor() public {
88            symbol = "SCCN";
89            name = "SUCCESSION";
90            decimals = 18;
91            _totalSupply = 100000000000000000000000000;
92            balances[0x6badBC00eB87Ac5c96A098A42491fB779e5aA785] = _totalSupply;
93            emit Transfer(address(0), 0x6badBC00eB87Ac5c96A098A42491fB779e5aA785, _totalSupply);
94        }
95
```

Recommendation
Auto distribute tokens during deployment of Token contract.

4. **approve() race**
The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender is able to spend their entire approval amount twice.

Reference:
https://eips.ethereum.org/EIPS/eip-20

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the code. Besides a security audit, please don’t consider this report as investment advice.

Closing Summary

Some issues of low severity have been reported during the audit. No high-level issues have been reported.

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com