CERTIK

Audit Report

Produced by CertiK

for

December 19, 2020

CertiK Audit Report For SotaNext

Request Date: 2020-12-18
Revision Date: 2020-12-19
Platform Name: Ethereum

Contents

Disclaimer

CertiK reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, CertiK's mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that projects are checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and assessments to each project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance's BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality deliverable. For more information: https://certik.io.

Executive Summary

This report has been prepared for SotaNext to discover issues and vulnerabilities in the source code of their Sota-platform smart contracts. A comprehensive examination has been performed, utilizing CertiK's Formal Verification Platform, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- 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.

Vulnerability Classification

CertiK categorizes issues into three buckets based on overall risk levels:

Critical

Code implementation does not match specification, which could result in the loss of funds for contract owner or users.

Medium

Code implementation does not match the specification under certain conditions, which could affect the security standard by loss of access control.

Low

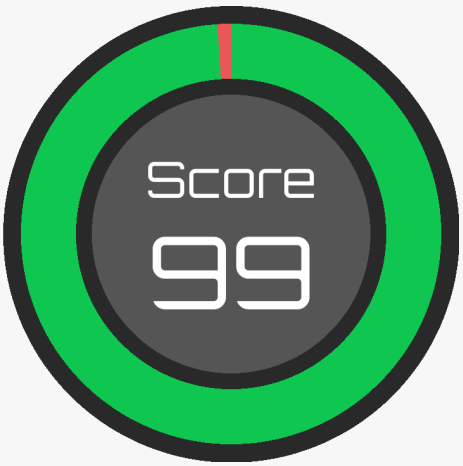
Code implementation does not follow best practices, or uses suboptimal design patterns, which could lead to security vulnerabilities further down the line.

Testing Summary

PASS

CERTIK *believes this smart contract passes security qualifications to be listed on digital asset exchanges.*

Dec 19, 2020



Type of Issues

CertiK's smart label engine applied 100% formal verification coverage on the source code. Our team of engineers has scanned the source code using proprietary static analysis tools and code-review methodologies. The following technical issues were found:

Title	Description	Issues	SWC ID
Integer Overflow/ Underflow	An overflow/underflow occurs when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function Incorrectness	Function implementation does not meet specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker can write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by miners to some degree.	3	SWC-116
Insecure Compiler Version	Using a fixed outdated compiler version or floating pragma can be problematic if there are publicly disclosed bugs and issues that affect the current compiler version used.	1	SWC-102 SWC-103
Insecure Randomness	Using block attributes to generate random numbers is unreliable, as they can be influenced by miners to some degree.	0	SWC-120
"tx.origin" for Authorization	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115

Title	Description	Issues	SWC ID
Delegatecall to Untrusted Callee	Calling untrusted contracts is very dangerous, so the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default, meaning a malicious user can make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized Variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used.	0	SWC-111
Unused Variables	Unused variables reduce code quality	0	SWC-131

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

Review Notes

Source Code SHA-256 Checksum

- **SotaToken.sol**[1]
  `33e66c14ca357006688271feb722374ed99954558b8524b9e00cd3313c46f512`

Summary

CertiK team is invited by Sota-platform team to audit the design and implementations of its to be released ERC20 based smart contract, and the source code has been analyzed under different perspectives and with different tools such as CertiK formal verification checkings as well as manual reviews by smart contract experts. That end-to-end process ensures proof of stability as well as a hands-on, engineering-focused process to close potential loopholes and recommend design changes in accordance with the best practices in the space. We have been actively interacting with client-side engineers when there was any potential loopholes or recommended design changes during the audit process, and Sota-platform team has been actively giving us updates for the source code and feedback about the business logics.

Meanwhile, it is recommended to have a more well-detailed document for the public to describe the source code specifications and implementations.

After manual review, we find this contract declares a allowable start time for each transfer, users can only transfer tokens after the start time. But the contract owner can add users to a whitelist so that these users can start transfer in advance. Besides, the owner can also change allowable start time forward.

Overall we found this erc20 contract follows good practices, with reasonable amount of features on top of the ERC20 related to administrive controls by the token issuer. With the final update of source code and delivery of the audit report, we conclude that the contract is not vulnerable to any classically known antipatterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend seeking multiple opinions, more test coverage and sandbox deployments before the mainnet release.

[1] <https://github.com/sota-platform/sota-token-contracts/blob/master/contracts/SotaToken.sol>

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 2 in File SotaToken.sol

```
2 pragma solidity ^0.7.0;
```

Only these compiler versions are safe to compile your code: 0.7.4

TIMESTAMP_DEPENDENCY

Line 69 in File SotaToken.sol

```
69        require(block.timestamp < allowTransferOn && _newTransferTime <
↪  allowTransferOn, "Invalid-time");
```

"block.timestamp" can be influenced by miners to some degree

TIMESTAMP_DEPENDENCY

Line 83 in File SotaToken.sol

```
83        require(block.timestamp > allowTransferOn ||
↪  whiteListTransfer[msg.sender], "Can-not-transfer");
```

"block.timestamp" can be influenced by miners to some degree

TIMESTAMP_DEPENDENCY

Line 98 in File SotaToken.sol

```
98        require(block.timestamp > allowTransferOn ||
↪  whiteListTransfer[msg.sender], "Can-not-transfer");
```

"block.timestamp" can be influenced by miners to some degree

Formal Verification Results

How to read

Detail for Request 1

transferFrom to same address

Verification date	20, Oct 2018
Verification timespan	395.38 ms

CERTIK *label location*	Line 30-34 in File howtoread.sol

CERTIK *label*

```
30    /*@CTK FAIL "transferFrom to same address"
31        @tag assume_completion
32        @pre from == to
33        @post __post.allowed[from][msg.sender] ==
34    */
```

Raw code location	Line 35-41 in File howtoread.sol

Raw code

```
35    function transferFrom(address from, address to
          ) {
36        balances[from] = balances[from].sub(tokens
37        allowed[from][msg.sender] = allowed[from][
38        balances[to] = balances[to].add(tokens);
39        emit Transfer(from, to, tokens);
40        return true;
41    }
```

Counterexample	This code violates the specification

Initial environment

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          from = 0x0
5          to = 0x0
6          tokens = 0x6c
7      }
8      This = 0
53             balance: 0x0
54         }
55     }
56
```

Post environment

```
57 After Execution:
58     Input = {
59         from = 0x0
60         to = 0x0
61         tokens = 0x6c
```

Formal Verification Request 1

SotaToken constructor

19, Dec 2020
474.82 ms

Line 18-28 in File SotaToken.sol

```
18      /*@CTK "SotaToken constructor"
19        @tag assume_completion
20        @post msg.sender != address(0)
21        @post __post._totalSupply == _totalSupply + cap
22        @post __post._balances[msg.sender] == _balances[msg.sender] + cap
23        @post __post._decimals == decimals
24        @post __post.whiteListTransfer[msg.sender] == true
25        @post __post._name == name
26        @post __post._symbol == symbol
27        @post __post._cap == cap
28      */
```

Line 29-38 in File SotaToken.sol

```
29      constructor (
30          string memory name,
31          string memory symbol,
32          uint8 decimals,
33          uint256 cap
34      ) public ERC20(name, symbol) ERC20Capped(cap) {
35          _setupDecimals(decimals);
36          _mint(_msgSender(), cap);
37          whiteListTransfer[_msgSender()] = true;
38      }
```

The code meets the specification.

Formal Verification Request 2

SotaToken adminWhiteList

19, Dec 2020
30.07 ms

Line 45-50 in File SotaToken.sol

```
45      /*@CTK "SotaToken adminWhiteList"
46        @tag assume_completion
47        @post msg.sender == _owner
48        @post __post.whiteListTransfer[_whitelistAddr] == _whiteList
```

```
49      @post __return == true
50    */
```

Line 51-54 in File SotaToken.sol

```
51    function adminWhiteList(address _whitelistAddr, bool _whiteList) public
      ↪ onlyOwner returns (bool) {
52        whiteListTransfer[_whitelistAddr] = _whiteList;
53        return true;
54    }
```

The code meets the specification.

Formal Verification Request 3

SotaToken adminSetTime

19, Dec 2020
38.6 ms

Line 60-67 in File SotaToken.sol

```
60    /*@CTK "SotaToken adminSetTime"
61      @tag assume_completion
62      @post msg.sender == _owner
63      @post block.timestamp < allowTransferOn
64      @post _newTransferTime < allowTransferOn
65      @post __post.allowTransferOn == _newTransferTime
66      @post __return == true
67    */
```

Line 68-72 in File SotaToken.sol

```
68    function adminSetTime(uint _newTransferTime) public onlyOwner returns
      ↪ (bool) {
69        require(block.timestamp < allowTransferOn && _newTransferTime <
      ↪ allowTransferOn, "Invalid-time");
70        allowTransferOn = _newTransferTime;
71        return true;
72    }
```

The code meets the specification.

Formal Verification Request 4

SotaToken transfer

19, Dec 2020
637.07 ms

Line 73-81 in File SotaToken.sol

```
73    /*@CTK "SotaToken transfer"
74      @tag assume_completion
75      @post (block.timestamp > allowTransferOn || whiteListTransfer[msg.sender])
76      @post msg.sender != address(0)
77      @post to != address(0)
78      @post msg.sender != to -> __post._balances[to] == _balances[to] + amount
79      @post msg.sender != to -> __post._balances[msg.sender] == _balances[msg.sender] - amount
80      @post msg.sender == to -> __post._balances[msg.sender] == _balances[msg.sender]
81      */
```

Line 82-85 in File SotaToken.sol

```
82    function transfer(address to, uint amount) public /*@IGNORE override(ERC20)@IGNORE*/ returns (bool) {
83        require(block.timestamp > allowTransferOn || whiteListTransfer[msg.sender], "Can-not-transfer");
84        return super.transfer(to, amount);
85    }
```

✅ The code meets the specification.

Formal Verification Request 5

SotaToken transferFrom

📅 19, Dec 2020
⏱ 857.96 ms

Line 86-96 in File SotaToken.sol

```
86    /*@CTK "SotaToken transferFrom"
87      @tag assume_completion
88      @post (block.timestamp > allowTransferOn || whiteListTransfer[msg.sender])
89      @post msg.sender != address(0)
90      @post from != address(0)
91      @post to != address(0)
92      @post from != to -> __post._balances[to] == _balances[to] + amount
93      @post from != to -> __post._balances[from] == _balances[from] - amount
94      @post from == to -> __post._balances[from] == _balances[from]
```

```
    @post __post._allowances[from][msg.sender] ==
↪ _allowances[from][msg.sender] - amount
    */
```

Line 97-100 in File SotaToken.sol

```
  function transferFrom(address from, address to, uint amount) public
↪ /*@IGNORE override(ERC20)@IGNORE*/ returns (bool) {
      require(block.timestamp > allowTransferOn ||
↪ whiteListTransfer[msg.sender], "Can-not-transfer");
      return super.transferFrom(from, to, amount);
  }
```

✅ The code meets the specification.

Formal Verification Request 6

SotaToken _beforeTokenTransfer

📅 19, Dec 2020
⏱ 8.37 ms

Line 104-108 in File SotaToken.sol

```
  /*@CTK "SotaToken _beforeTokenTransfer"
    @tag assume_completion
    @pre from == address(0)
    @post _totalSupply + amount <= _cap
    */
```

Line 109-111 in File SotaToken.sol

```
  function _beforeTokenTransfer(address from, address to, uint256 amount)
↪ internal /*@IGNORE virtual  override(ERC20, ERC20Capped) @IGNORE*/{
      super._beforeTokenTransfer(from, to, amount);
  }
```

✅ The code meets the specification.

Source Code with CertiK Labels

SotaToken.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.0;
import "../ERC20Capped.sol";
import "../ERC20Burnable.sol";
import "../ERC20.sol";
import "../Ownable.sol";

contract SotaToken is ERC20, ERC20Capped, ERC20Burnable, Ownable {

    uint public allowTransferOn = 1617123600; // 2021-03-31 0:00:00 GMT+7 timezone
    mapping (address => bool ) public whiteListTransfer;

    /**
     * @dev Constructor function of Sota Token
     * @dev set name, symbol and decimal of token
     * @dev mint totalSupply (cap) to deployer
     */
    /*@CTK "SotaToken constructor"
      @tag assume_completion
      @post msg.sender != address(0)
      @post __post._totalSupply == _totalSupply + cap
      @post __post._balances[msg.sender] == _balances[msg.sender] + cap
      @post __post._decimals == decimals
      @post __post.whiteListTransfer[msg.sender] == true
      @post __post._name == name
      @post __post._symbol == symbol
      @post __post._cap == cap
    */
    constructor (
        string memory name,
        string memory symbol,
        uint8 decimals,
        uint256 cap
    ) public ERC20(name, symbol) ERC20Capped(cap) {
        _setupDecimals(decimals);
        _mint(_msgSender(), cap);
        whiteListTransfer[_msgSender()] = true;
    }

    /**
     * @dev Admin whitelist/un-whitelist transfer
     * @dev to allow address transfer
```

```
     * @dev token before allowTransferOn
     */
    /*@CTK "SotaToken adminWhiteList"
      @tag assume_completion
      @post msg.sender == _owner
      @post __post.whiteListTransfer[_whitelistAddr] == _whiteList
      @post __return == true
    */
    function adminWhiteList(address _whitelistAddr, bool _whiteList) public onlyOwner returns (bool) {
        whiteListTransfer[_whitelistAddr] = _whiteList;
        return true;
    }

    /**
     * @dev Admin can set allowTransferOn to
     * @dev any time before 2021-03-31 0:00:00 GMT+7
     */
    /*@CTK "SotaToken adminSetTime"
      @tag assume_completion
      @post msg.sender == _owner
      @post block.timestamp < allowTransferOn
      @post _newTransferTime < allowTransferOn
      @post __post.allowTransferOn == _newTransferTime
      @post __return == true
    */
    function adminSetTime(uint _newTransferTime) public onlyOwner returns (bool) {
        require(block.timestamp < allowTransferOn && _newTransferTime < allowTransferOn, "Invalid-time");
        allowTransferOn = _newTransferTime;
        return true;
    }
    /*@CTK "SotaToken transfer"
      @tag assume_completion
      @post (block.timestamp > allowTransferOn || whiteListTransfer[msg.sender])
      @post msg.sender != address(0)
      @post to != address(0)
      @post msg.sender != to -> __post._balances[to] == _balances[to] + amount
      @post msg.sender != to -> __post._balances[msg.sender] == _balances[msg.sender] - amount
      @post msg.sender == to -> __post._balances[msg.sender] == _balances[msg.sender]
      */
        require(block.timestamp > allowTransferOn || whiteListTransfer[msg.sender], "Can-not-transfer");
```

```
        return super.transfer(to, amount);
    }
    /*@CTK "SotaToken transferFrom"
      @tag assume_completion
      @post (block.timestamp > allowTransferOn || whiteListTransfer[msg.sender])
      @post msg.sender != address(0)
      @post from != address(0)
      @post to != address(0)
      @post from != to -> __post._balances[to] == _balances[to] + amount
      @post from != to -> __post._balances[from] == _balances[from] - amount
      @post from == to -> __post._balances[from] == _balances[from]
      @post __post._allowances[from][msg.sender] == _allowances[from][msg.sender] - amount
      */
        require(block.timestamp > allowTransferOn || whiteListTransfer[msg.sender], "Can-not-transfer");
        return super.transferFrom(from, to, amount);
    }
     /**
     * @dev See {ERC20-_beforeTokenTransfer}.
     */
    /*@CTK "SotaToken _beforeTokenTransfer"
      @tag assume_completion
      @pre from == address(0)
      @post _totalSupply + amount <= _cap
      */
        super._beforeTokenTransfer(from, to, amount);
    }
}
```