CERTIK VERIFICATION REPORT FOR TOP NETWORK



Request Date: 2019-03-19 Revision Date: 2019-03-19



Disclaimer

This Report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and Top Network(the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the "Agreement"). This Report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This Report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiKs prior written consent.





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





Summary

This audit report summarises the smart contract verification service requested by Top Network. The goal of this security audit is to guarantee that the audited smart contracts are robust enough to avoid any potential security loopholes.

The result of this report is only a reflection of the source code that was determined in this scope, and of the source code at the time of the audit.

Type of Issues

CertiK smart label engine applied 100% coveraged formal verification labels on the source code, and scanned the code using our proprietary static analysis and formal verification engine to detect the follow type of issues.

Title	Description	Issues	SWC ID
Integer Overflow	An overflow/underflow happens when an arithmetic	0	SWC-101
and Underflow	operation reaches the maximum or minimum size of		
	a type.		
Function incor-	Function implementation does not meet the specifi-	0	
rectness	cation, leading to intentional or unintentional vul-		
	nerabilities.		
Buffer Overflow	An attacker is able to write to arbitrary storage lo-	0	SWC-124
	cations of a contract if array of out bound happens		
Reentrancy	A malicious contract can call back into the calling	0	SWC-107
	contract before the first invocation of the function is		
	finished.		
Transaction Or-	A race condition vulnerability occurs when code de-	0	SWC-114
der Dependence	pends on the order of the transactions submitted to		
	it.		
Timestamp De-	Timestamp can be influenced by minors to some de-	0	SWC-116
pendence	gree.		



Insecure Com-	Using an fixed outdated compiler version or float-	0	SWC-102
piler Version	ing pragma can be problematic, if there are publicly		SWC-103
	disclosed bugs and issues that affect the current com-		
	piler version used.		
Insecure Ran-	Block attributes are insecure to generate random	0	SWC-120
domness	numbers, as they can be influenced by minors to		
	some degree.		
tx.origin for au-	tx.origin should not be used for authorization. Use	0	SWC-115
thorization	msg.sender instead.		
Delegatecall to	Calling into untrusted contracts is very dangerous,	0	SWC-112
Untrusted Callee	the target and arguments provided must be sani-		
	tized.		
State Variable	Labeling the visibility explicitly makes it easier to	0	SWC-108
Default Visibility	catch incorrect assumptions about who can access		
	the variable.		
Function Default	Functions are public by default. A malicious user	0	SWC-100
Visibility	is able to make unauthorized or unintended state		
	changes if a developer forgot to set the visibility.		
Uninitialized	Uninitialized local storage variables can point to	0	SWC-109
variables	other unexpected storage variables in the contract.		
Assertion Failure	The assert() function is meant to assert invariants.	0	SWC-110
	Properly functioning code should never reach a fail-		
	ing assert statement.		
Deprecated	Several functions and operators in Solidity are dep-	0	SWC-111
Solidity Features	recated and should not be used as best practice.		
Unused variables	Unused variables reduce code quality	0	

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

For every issues found, CertiK categorizes them into 3 buckets based on its risk level:

- Critical: The code implementation does not match the specification, or it could result in loss of funds for contract owner or users.
- Medium: The code implementation does not match the specification at certain condition, or it could affect the security standard by lost of access control.



• Low: The code implementation is not a best practice, or use a suboptimal design pattern, which may lead to security vulnerability, but no concern found yet.



Source Code with CertiK Labels

File top.sol

```
1 pragma solidity ^0.5.6;
 2
 3 /**
 4 * @title ERC20Basic
 5 * @dev Simpler version of ERC20 interface
 6
   * @dev see https://github.com/ethereum/EIPs/issues/179
 7
 8
   contract ERC20Basic {
    uint256 public totalSupply = 2e28;
 9
    function balanceOf(address who) public view returns (uint256);
10
   function transfer(address to, uint256 value) public returns (bool);
11
     event Transfer(address indexed from, address indexed to, uint256 value);
13 }
14
15 contract TOPToken is ERC20Basic {
16
     bytes32 public name = "TOP Network";
17
     bytes32 public symbol = "TOP";
     uint256 public decimals = 18;
18
19
     address private owner = address(0);
20
     bool private active = false;
21
22
     mapping(address => uint256) private balances;
23
24
     event OwnershipTransferred(address indexed orgOwner, address indexed newOwner);
25
26
     //@CTK NO_OVERFLOW
27
     //@CTK NO_BUF_OVERFLOW
28
     //@CTK NO_ASF
29
     constructor() public {
30
       owner = msg.sender;
31
       balances[owner] = totalSupply;
32
       active = true;
33
     }
34
35
     modifier onlyOwner() {
36
       require(msg.sender == owner);
37
     }
38
39
40
41
     * Odev transfer token for a specified address
42
     * Oparam _to The address to transfer to.
43
     * @param _value The amount to be transferred.
44
     */
     //@CTK NO_OVERFLOW
45
     //@CTK NO_BUF_OVERFLOW
46
47
     //@CTK NO_ASF
48
     /*@CTK "transfer2_same"
49
       @post (_to) == (msg.sender) -> (__reverted) == (true)
50
51
     /*@CTK "transfer2"
52
       @tag assume_completion
53
       @pre (_to) != (msg.sender)
    @post (__post.balances[_to]) == ((balances[_to]) + (_value))
```



```
@post (__post.balances[msg.sender]) == ((balances[msg.sender]) - (_value))
55
56
        @post (__return) == (true)
      */
57
      function transfer(address _to, uint256 _value) public returns (bool) {
 58
59
        require(active);
60
        require(_to != address(0));
 61
        require(_to != msg.sender);
 62
        require(_value <= balances[msg.sender]);</pre>
63
64
        uint256 bal = balances[_to] + _value;
 65
        require(bal >= balances[_to]);
 66
 67
        balances[msg.sender] = balances[msg.sender] - _value;
        balances[_to] = bal;
 68
 69
 70
        emit Transfer(msg.sender, _to, _value);
71
        return true;
      }
72
73
74
75
      * @dev Gets the balance of the specified address.
      * Oparam _owner The address to query the the balance of.
 76
 77
      * @return An uint256 representing the amount owned by the passed address.
78
      */
79
      //@CTK NO_OVERFLOW
80
      //@CTK NO_BUF_OVERFLOW
81
      //@CTK NO_ASF
82
      /*@CTK "balanceOf"
83
        @pre (active) == (true)
        @post (__reverted) == (false)
 84
85
        @post (bal) == (balances[_owner])
86
      function balanceOf(address _owner) public view returns (uint256 bal) {
87
 88
        require(active);
 89
        return balances[_owner];
90
      }
91
92
      // Only owner can deactivate
93
      //@CTK NO_OVERFLOW
94
      //@CTK NO_BUF_OVERFLOW
95
      //@CTK NO_ASF
96
      /*@CTK "deactivate"
97
        @post (msg.sender) != (owner) -> (__reverted) == (true)
98
        @post (msg.sender) == (owner) -> (__post.active) == (false)
99
100
      function deactivate() public onlyOwner {
101
        active = false;
102
103
104
      // Only owner can activate
105
      //@CTK NO_OVERFLOW
106
      //@CTK NO_BUF_OVERFLOW
107
      //@CTK NO_ASF
108
      /*@CTK "activate"
109
        @post (msg.sender) != (owner) -> (__reverted) == (true)
110
        @post (msg.sender) == (owner) -> (__post.active) == (true)
111
112
      function activate() public onlyOwner {
```



```
113
      active = true;
114
115
      //@CTK NO_OVERFLOW
116
117
      //@CTK NO_BUF_OVERFLOW
118
      //@CTK NO_ASF
      /*@CTK "transferOwnership"
119
        @post (msg.sender) != (owner) -> (__reverted) == (true)
120
121
        @post ((msg.sender) == (owner) && (newOwner != 0x0))
122
             -> ((__post.owner) == (newOwner))
123
      function transferOwnership(address newOwner) public onlyOwner {
124
125
        require(newOwner != address(0));
126
        emit OwnershipTransferred(owner, newOwner);
127
        owner = newOwner;
128
129
130
      // Only owner can kill
131
      function kill() public onlyOwner {
132
        require(!active);
133
        selfdestruct(msg.sender);
134
      }
135 }
```



How to read

Detail for Request 1

transferFrom to same address

```
Verification\ date
                       20, Oct 2018
                        • 395.38 ms
 Verification timespan
CERTIK label location
                       Line 30-34 in File howtoread.sol
                   30
                           /*@CTK FAIL "transferFrom to same address"
                   31
                               @tag assume_completion
     □ERTIK label
                   32
                               @pre from == to
                   33
                               @post __post.allowed[from][msg.sender] ==
                   34
    Raw code location
                       Line 35-41 in File howtoread.sol
                   35
                           function transferFrom(address from, address to
                   36
                               balances[from] = balances[from].sub(tokens
                   37
                               allowed[from][msg.sender] = allowed[from][
         Raw\ code
                   38
                               balances[to] = balances[to].add(tokens);
                   39
                               emit Transfer(from, to, tokens);
                   40
                               return true;
     Counter example \\
                        This code violates the specification
                    1
                       Counter Example:
                       Before Execution:
                    3
                           Input = {
                    4
                               from = 0x0
                    5
                               to = 0x0
                    6
                               tokens = 0x6c
                    7
                           This = 0
  Initial environment
                                   balance: 0x0
                   54
                   55
                   56
                   57
                       After Execution:
                   58
                           Input = {
                               from = 0x0
                   59
    Post environment
                   60
                               to = 0x0
                   61
                               tokens = 0x6c
```



Static Analysis Request

$INSECURE_COMPILER_VERSION$

Line 1 in File top.sol

- 1 pragma solidity ^0.5.6;
 - 1 Only these compiler versions are safe to compile your code: 0.5.6



If method completes, integer overflow would not happen.

```
## 19, Mar 2019
© 24.86 ms
```

Line 26 in File top.sol

```
26  //@CTK NO_OVERFLOW
  Line 29-33 in File top.sol

29  constructor() public {
    owner = msg.sender;
    balances[owner] = totalSupply;
    active = true;
    33  }
```

The code meets the specification

Formal Verification Request 2

Buffer overflow / array index out of bound would never happen.

```
19, Mar 2019
0.49 ms
```

Line 27 in File top.sol

```
27  //@CTK NO_BUF_OVERFLOW
  Line 29-33 in File top.sol

29  constructor() public {
    owner = msg.sender;
    balances[owner] = totalSupply;
    active = true;
    33  }
```

The code meets the specification

Formal Verification Request 3

Method will not encounter an assertion failure.

```
19, Mar 2019

0.46 ms
```

Line 28 in File top.sol

```
28 //@CTK NO_ASF
```

Line 29-33 in File top.sol



```
29
    constructor() public {
30
       owner = msg.sender;
31
       balances[owner] = totalSupply;
32
       active = true;
33
     }
```

Formal Verification Request 4

If method completes, integer overflow would not happen.

```
## 19, Mar 2019
119.89 ms
```

Line 45 in File top.sol

```
//@CTK NO_OVERFLOW
```

Line 58-72 in File top.sol

```
58
     function transfer(address _to, uint256 _value) public returns (bool) {
59
       require(active);
60
       require(_to != address(0));
       require(_to != msg.sender);
61
62
       require(_value <= balances[msg.sender]);</pre>
63
64
       uint256 bal = balances[_to] + _value;
       require(bal >= balances[_to]);
65
66
67
       balances[msg.sender] = balances[msg.sender] - _value;
68
       balances[_to] = bal;
69
70
       emit Transfer(msg.sender, _to, _value);
71
       return true;
72
```

The code meets the specification

Formal Verification Request 5

Buffer overflow / array index out of bound would never happen.

```
## 19, Mar 2019
11.75 ms
```

Line 46 in File top.sol

```
46 //@CTK NO_BUF_OVERFLOW
   Line 58-72 in File top.sol
     function transfer(address _to, uint256 _value) public returns (bool) {
58
59
       require(active);
       require(_to != address(0));
```



```
61
       require(_to != msg.sender);
62
       require(_value <= balances[msg.sender]);</pre>
63
       uint256 bal = balances[_to] + _value;
64
65
       require(bal >= balances[_to]);
66
67
       balances[msg.sender] = balances[msg.sender] - _value;
       balances[_to] = bal;
68
69
70
       emit Transfer(msg.sender, _to, _value);
71
       return true;
72
```

Formal Verification Request 6

Method will not encounter an assertion failure.

```
## 19, Mar 2019
11.81 ms
```

Line 47 in File top.sol

```
47 //@CTK NO_ASF
```

Line 58-72 in File top.sol

```
58
     function transfer(address _to, uint256 _value) public returns (bool) {
59
       require(active);
       require(_to != address(0));
60
       require(_to != msg.sender);
61
62
       require(_value <= balances[msg.sender]);</pre>
63
       uint256 bal = balances[_to] + _value;
64
       require(bal >= balances[_to]);
65
66
67
       balances[msg.sender] = balances[msg.sender] - _value;
68
       balances[_to] = bal;
69
70
       emit Transfer(msg.sender, _to, _value);
71
       return true;
72
```

The code meets the specification

Formal Verification Request 7

```
transfer2\_same
```

```
19, Mar 2019
19.7 ms
```

Line 48-50 in File top.sol



```
/*@CTK "transfer2_same"
48
49
       @post (_to) == (msg.sender) -> (__reverted) == (true)
50
   Line 58-72 in File top.sol
     function transfer(address _to, uint256 _value) public returns (bool) {
58
59
       require(active);
60
       require(_to != address(0));
61
       require(_to != msg.sender);
       require(_value <= balances[msg.sender]);</pre>
62
63
64
       uint256 bal = balances[_to] + _value;
65
       require(bal >= balances[_to]);
66
       balances[msg.sender] = balances[msg.sender] - _value;
67
68
       balances[_to] = bal;
69
70
       emit Transfer(msg.sender, _to, _value);
71
       return true;
72
   The code meets the specification
```

```
transfer2
```

```
19, Mar 2019
100.2 ms
```

Line 51-57 in File top.sol

```
/*@CTK "transfer2"

0tag assume_completion

0pre (_to) != (msg.sender)

0post (__post.balances[_to]) == ((balances[_to]) + (_value))

0post (__post.balances[msg.sender]) == ((balances[msg.sender]) - (_value))

0post (__return) == (true)

*/
```

Line 58-72 in File top.sol

```
function transfer(address _to, uint256 _value) public returns (bool) {
58
59
       require(active);
60
       require(_to != address(0));
61
       require(_to != msg.sender);
       require(_value <= balances[msg.sender]);</pre>
62
63
64
       uint256 bal = balances[_to] + _value;
65
       require(bal >= balances[_to]);
66
67
       balances[msg.sender] = balances[msg.sender] - _value;
       balances[_to] = bal;
68
69
70
       emit Transfer(msg.sender, _to, _value);
71
       return true;
72
```



Formal Verification Request 9

If method completes, integer overflow would not happen.

```
## 19, Mar 2019
• 19.08 ms
```

Line 79 in File top.sol

```
79 //@CTK NO_OVERFLOW

Line 87-90 in File top.sol

87 function balanceOf(address _owner) public view returns (uint256 bal) {
88 require(active);
89 return balances[_owner];
90 }
```

The code meets the specification

Formal Verification Request 10

Buffer overflow / array index out of bound would never happen.

```
19, Mar 2019
0.54 ms
```

Line 80 in File top.sol

```
80  //@CTK NO_BUF_OVERFLOW
   Line 87-90 in File top.sol

87  function balanceOf(address _owner) public view returns (uint256 bal) {
    require(active);
    return balances[_owner];
   90  }
```

The code meets the specification

Formal Verification Request 11

Method will not encounter an assertion failure.

```
19, Mar 2019
0.64 ms
```

Line 81 in File top.sol

```
81 //@CTK NO_ASF
```

Line 87-90 in File top.sol



```
87  function balanceOf(address _owner) public view returns (uint256 bal) {
88    require(active);
89    return balances[_owner];
90  }
```

Formal Verification Request 12

balanceOf

```
19, Mar 2019
1.92 ms
```

Line 82-86 in File top.sol

Line 87-90 in File top.sol

```
function balanceOf(address _owner) public view returns (uint256 bal) {
require(active);
return balances[_owner];
}
```

The code meets the specification

Formal Verification Request 13

If method completes, integer overflow would not happen.

```
19, Mar 2019
20.86 ms
```

The code meets the specification



Buffer overflow / array index out of bound would never happen.

```
19, Mar 2019
0.51 ms
```

Line 94 in File top.sol

```
94 //@CTK NO_BUF_OVERFLOW
```

Line 100-102 in File top.sol

```
100 function deactivate() public onlyOwner {
101 active = false;
102 }
```

The code meets the specification

Formal Verification Request 15

Method will not encounter an assertion failure.

```
19, Mar 2019
0.5 ms
```

Line 95 in File top.sol

```
95 //@CTK NO_ASF
```

Line 100-102 in File top.sol

```
100  function deactivate() public onlyOwner {
101  active = false;
102 }
```

The code meets the specification

Formal Verification Request 16

deactivate

```
## 19, Mar 2019
1.83 ms
```

Line 96-99 in File top.sol

```
/*@CTK "deactivate"

@post (msg.sender) != (owner) -> (__reverted) == (true)

@post (msg.sender) == (owner) -> (__post.active) == (false)

*/
```

Line 100-102 in File top.sol

```
function deactivate() public onlyOwner {
   active = false;
}
```



Formal Verification Request 17

If method completes, integer overflow would not happen.

```
## 19, Mar 2019
19.43 ms
```

Line 105 in File top.sol

```
//@CTK NO_OVERFLOW
105
    Line 112-114 in File top.sol
      function activate() public onlyOwner {
112
113
        active = true;
114
```

The code meets the specification

Formal Verification Request 18

Buffer overflow / array index out of bound would never happen.

```
## 19, Mar 2019
```

 \odot 0.51 ms

Line 106 in File top.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 112-114 in File top.sol

```
function activate() public onlyOwner {
112
113
        active = true;
114
```

The code meets the specification

Formal Verification Request 19

Method will not encounter an assertion failure.

```
## 19, Mar 2019
```

0.5 ms

Line 107 in File top.sol

```
//@CTK NO_ASF
```

Line 112-114 in File top.sol



```
function activate() public onlyOwner {
   active = true;
}
```

Formal Verification Request 20

activate

```
19, Mar 2019
1.68 ms
```

Line 108-111 in File top.sol

Line 112-114 in File top.sol

```
function activate() public onlyOwner {
   active = true;
}
```

✓ The code meets the specification

Formal Verification Request 21

If method completes, integer overflow would not happen.

```
19, Mar 2019
28.19 ms
```

Line 116 in File top.sol

```
Line 124-128 in File top.sol

function transferOwnership(address newOwner) public onlyOwner {
 require(newOwner != address(0));
 emit OwnershipTransferred(owner, newOwner);
 owner = newOwner;
}
```

The code meets the specification



Buffer overflow / array index out of bound would never happen.

```
## 19, Mar 2019
0.62 \text{ ms}
```

Line 117 in File top.sol

```
117 //@CTK NO_BUF_OVERFLOW
    Line 124-128 in File top.sol
124
      function transferOwnership(address newOwner) public onlyOwner {
125
        require(newOwner != address(0));
126
        emit OwnershipTransferred(owner, newOwner);
127
        owner = newOwner;
128
      }
```

The code meets the specification

Formal Verification Request 23

Method will not encounter an assertion failure.

```
## 19, Mar 2019
\bullet 0.63 ms
```

127

128

Line 118 in File top.sol

owner = newOwner;

```
118 //@CTK NO_ASF
    Line 124-128 in File top.sol
124
      function transferOwnership(address newOwner) public onlyOwner {
125
        require(newOwner != address(0));
126
        emit OwnershipTransferred(owner, newOwner);
```

The code meets the specification

Formal Verification Request 24

transferOwnership

```
## 19, Mar 2019
\circ 3.17 ms
```

Line 119-123 in File top.sol

```
119
      /*@CTK "transferOwnership"
120
        @post (msg.sender) != (owner) -> (__reverted) == (true)
121
        @post ((msg.sender) == (owner) && (newOwner != 0x0))
122
             -> ((__post.owner) == (newOwner))
123
```



Line 124-128 in File top.sol

```
function transferOwnership(address newOwner) public onlyOwner {
    require(newOwner != address(0));
    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
}
```

 \bigcirc The code meets the specification