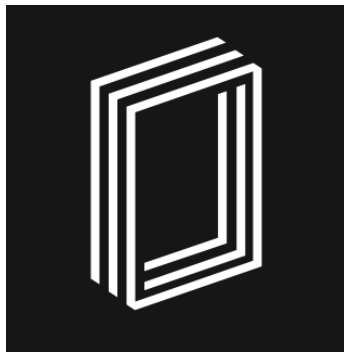


CERTIK AUDIT REPORT FOR VANTA



Request Date: 2019-09-05
Revision Date: 2019-09-15
Platform Name: Ethereum





Contents

Disclaimer	1
About CertiK	2
Executive Summary	3
Vulnerability Classification	3
Testing Summary	4
Audit Score	4
Type of Issues	4
Vulnerability Details	5
Manual Review Notes	6
Static Analysis Results	7
Formal Verification Results	8
How to read	8
Source Code with CertiK Labels	48



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 Vanta (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 CertiK’s prior written consent.



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, has developed a proprietary Formal Verification technology to apply rigorous and complete mathematical reasoning against code. This process ensures algorithms, protocols, and business functionalities are secured and working as intended across all platforms.

CertiK differs from traditional testing approaches by employing Formal Verification to mathematically prove blockchain ecosystem and smart contracts are hacker-resistant and bug-free. CertiK uses this industry-leading technology together with standardized test suites, static analysis, and expert manual review to create a full-stack solution for our partners across the blockchain world to secure 6.2B in assets.

For more information: <https://certik.org/>



Executive Summary

This report has been prepared for Vanta to discover issues and vulnerabilities in the source code of their VantaToken smart contract. 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 practice 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 3 buckets based on overall risk levels:

Critical

The code implementation does not match the specification, or it could result in the loss of funds for contract owner or users.

Medium

The code implementation does not match the specification under certain conditions, or it could affect the security standard by lost of access control.

Low

The code implementation does not follow best practices, or use suboptimal design patterns, which may 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.

Sep 14, 2019



Type of Issues

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

Title	Description	Issues	SWC ID
Integer Overflow and Underflow	An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function incorrectness	Function implementation does not meet the specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker is able to 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 minors to some degree.	0	SWC-116
Insecure Compiler Version	Using an 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.	0	SWC-102 SWC-103
Insecure Randomness	Block attributes are insecure to generate random numbers, as they can be influenced by minors 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
Delegatecall to Untrusted Callee	Calling into untrusted contracts is very dangerous, 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. A malicious user is able to 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 as best practice.	0	SWC-111
Unused variables	Unused variables reduce code quality	0	

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.



Manual Review Notes

Review Details

Source Code SHA-256 Checksum

- **VantaToken.sol** `0xfdf574766ba1a96a553e175aef fa85ad78063f0b32e6fbd91930ff58af5e39735dc6052b4d0f696f67eb76506c0d4de0fe9319ff`

Summary

CertiK was chosen by Vanta to audit the design and implementation of its new **VantaToken** smart contract. To ensure comprehensive protection, the source code has been analyzed by the proprietary CertiK formal verification engine and manually reviewed by our smart contract experts and engineers. 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.****

Overall we found the smart contracts to follow good practices. With the final update of source code and delivery of the audit report, we conclude that the contract is structurally sound and not vulnerable to any classically known anti-patterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend to seek multiple opinions, keep improving the codebase, and more test coverage and sandbox deployments before the mainnet release.

Recommendations

Items in this section are low impact to the overall aspects of the smart contracts, thus will let client to decide whether to have those reflected in the final deployed version of source codes.

VantaToken.sol

- **INFO** `_transfer()`: Recommend checking if `to == address(0)` to prevent value loss and be consistent with `_approve()`.
- **INFO** `_burnFrom()`: Internal method unused.
- **INFO** `require()`: Recommend supplying error message.
- **INFO** `transferOwnership()`: Recommend using the pull model:

```
function transferOwnership(address payable _newOwner) public onlyOwner {
    newOwner = _newOwner;
}
function acceptOwnership() public {
    require(msg.sender == newOwner);
    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
    newOwner = address(0);
}
```




Static Analysis Results

INSECURE_COMPILER_VERSION

Line 21 in File VantaToken.sol

21 `pragma solidity ^0.5.2;`

 Only these compiler versions are safe to compile your code: 0.5.10

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	<pre> 30 /*@CTK FAIL "transferFrom to same address" 31 @tag assume_completion 32 @pre from == to 33 @post __post.allowed[from][msg.sender] == 34 */ </pre>
Raw code location	Line 35-41 in File howtoread.sol
Raw code	<pre> 35 function transferFrom(address from, address to 36) { 37 balances[from] = balances[from].sub(tokens 38 allowed[from][msg.sender] = allowed[from][39 balances[to] = balances[to].add(tokens); 40 emit Transfer(from, to, tokens); 41 return true; </pre>
Counterexample	<div>  This code violates the specification </div>
Initial environment	<pre> 1 Counter Example: 2 Before Execution: 3 Input = { 4 from = 0x0 5 to = 0x0 6 tokens = 0x6c 7 } 8 This = 0 </pre>
Post environment	<pre> 52 } 53 balance: 0x0 54 } 55 } 56 57 After Execution: 58 Input = { 59 from = 0x0 60 to = 0x0 61 tokens = 0x6c </pre>



Formal Verification Request 1

SafeMath mul zero

 14, Sep 2019

 14.43 ms

Line 53-58 in File VantaToken.sol

```
53  /*@CTK "SafeMath mul zero"
54      @tag spec
55      @tag is_pure
56      @pre (a == 0)
57      @post __return == 0
58  */
```

Line 69-81 in File VantaToken.sol


```
69  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
70      // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
71      // benefit is lost if 'b' is also tested.
72      // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
73      if (a == 0) {
74          return 0;
75      }
76
77      uint256 c = a * b;
78      require(c / a == b);
79
80      return c;
81  }
```

 The code meets the specification.

Formal Verification Request 2

SafeMath mul nonzero

 14, Sep 2019

 292.75 ms

Line 59-68 in File VantaToken.sol

```
59  /*@CTK "SafeMath mul nonzero"
60      @tag spec
61      @tag is_pure
62      @pre (a != 0)
63      @post (a * b / a != b) == __reverted
64      @post !__reverted -> __return == a * b
65      @post !__reverted -> !__has_overflow
66      @post !__reverted -> !__has_assertion_failure
67      @post !(__has_buf_overflow)
68  */
```

Line 69-81 in File VantaToken.sol

```
69  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
70      // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
71      // benefit is lost if 'b' is also tested.
```

```

72      // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
73      if (a == 0) {
74          return 0;
75      }
76
77      uint256 c = a * b;
78      require(c / a == b);
79
80      return c;
81  }

```

✓ The code meets the specification.

Formal Verification Request 3

SafeMath div



14, Sep 2019



12.07 ms

Line 86-94 in File VantaToken.sol

```

86  /*@CTK "SafeMath div"
87      @tag spec
88      @tag is_pure
89      @post (b == 0) == __reverted
90      @post !__reverted -> __return == a / b
91      @post !__reverted -> !__has_overflow
92      @post !__reverted -> !__has_assertion_failure
93      @post !(__has_buf_overflow)
94  */

```

Line 95-102 in File VantaToken.sol

```

95  function div(uint256 a, uint256 b) internal pure returns (uint256) {
96      // Solidity only automatically asserts when dividing by 0
97      require(b > 0);
98      uint256 c = a / b;
99      // assert(a == b * c + a % b); // There is no case in which this doesn't hold
100
101      return c;
102  }

```

✓ The code meets the specification.

Formal Verification Request 4

SafeMath sub



14, Sep 2019



11.02 ms

Line 107-115 in File VantaToken.sol

```

107  /*@CTK "SafeMath sub"
108      @tag spec

```



```

109     @tag is_pure
110     @post (b > a) == __reverted
111     @post !__reverted -> __return == a - b
112     @post !__reverted -> !__has_overflow
113     @post !__reverted -> !__has_assertion_failure
114     @post !(__has_buf_overflow)
115     */

```

Line 116-121 in File VantaToken.sol

```

116     function sub(uint256 a, uint256 b) internal pure returns (uint256) {
117         require(b <= a);
118         uint256 c = a - b;
119
120         return c;
121     }

```

✓ The code meets the specification.

Formal Verification Request 5

SafeMath add



14, Sep 2019



13.53 ms

Line 126-134 in File VantaToken.sol

```

126     /*@CTK "SafeMath add"
127     @tag spec
128     @tag is_pure
129     @post (a + b < a || a + b < b) == __reverted
130     @post !__reverted -> __return == a + b
131     @post !__reverted -> !__has_overflow
132     @post !__reverted -> !__has_assertion_failure
133     @post !(__has_buf_overflow)
134     */

```

Line 135-140 in File VantaToken.sol

```

135     function add(uint256 a, uint256 b) internal pure returns (uint256) {
136         uint256 c = a + b;
137         require(c >= a);
138
139         return c;
140     }

```

✓ The code meets the specification.

Formal Verification Request 6

If method completes, integer overflow would not happen.



14, Sep 2019



4.72 ms

Line 176 in File VantaToken.sol



176 `//@CTK NO_OVERFLOW`

Line 183-185 in File VantaToken.sol

```
183 function totalSupply() public view returns (uint256) {  
184     return _totalSupply;  
185 }
```

✓ The code meets the specification.

Formal Verification Request 7

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

📅 14, Sep 2019

🕒 0.33 ms

Line 177 in File VantaToken.sol

177 `//@CTK NO_BUF_OVERFLOW`

Line 183-185 in File VantaToken.sol

```
183 function totalSupply() public view returns (uint256) {  
184     return _totalSupply;  
185 }
```

✓ The code meets the specification.

Formal Verification Request 8

Method will not encounter an assertion failure.

📅 14, Sep 2019

🕒 0.33 ms

Line 178 in File VantaToken.sol

178 `//@CTK NO_ASF`

Line 183-185 in File VantaToken.sol

```
183 function totalSupply() public view returns (uint256) {  
184     return _totalSupply;  
185 }
```

✓ The code meets the specification.

Formal Verification Request 9

totalSupply

📅 14, Sep 2019

🕒 0.34 ms

Line 179-182 in File VantaToken.sol



```
179  /*@CTK totalSupply
180      @tag assume_completion
181      @post (__return) == (_totalSupply)
182  */
```

Line 183-185 in File VantaToken.sol

```
183  function totalSupply() public view returns (uint256) {
184      return _totalSupply;
185  }
```

✓ The code meets the specification.

Formal Verification Request 10

If method completes, integer overflow would not happen.

📅 14, Sep 2019

🕒 4.66 ms

Line 192 in File VantaToken.sol

```
192  //@CTK NO_OVERFLOW
```

Line 198-200 in File VantaToken.sol

```
198  function balanceOf(address owner) public view returns (uint256) {
199      return _balances[owner];
200  }
```

✓ The code meets the specification.

Formal Verification Request 11

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

📅 14, Sep 2019

🕒 0.35 ms

Line 193 in File VantaToken.sol

```
193  //@CTK NO_BUF_OVERFLOW
```

Line 198-200 in File VantaToken.sol


```
198  function balanceOf(address owner) public view returns (uint256) {
199      return _balances[owner];
200  }
```

✓ The code meets the specification.

Formal Verification Request 12

Method will not encounter an assertion failure.

 14, Sep 2019

 0.3 ms

Line 194 in File VantaToken.sol

194 `//@CTK NO_ASF`

Line 198-200 in File VantaToken.sol


```
198     function balanceOf(address owner) public view returns (uint256) {
199         return _balances[owner];
200     }
```

 The code meets the specification.

Formal Verification Request 13

balanceOf

 14, Sep 2019

 0.33 ms

Line 195-197 in File VantaToken.sol

```
195     /*@CTK balanceOf
196         @post __return == __post._balances[owner]
197     */
```

Line 198-200 in File VantaToken.sol


```
198     function balanceOf(address owner) public view returns (uint256) {
199         return _balances[owner];
200     }
```

 The code meets the specification.

Formal Verification Request 14

If method completes, integer overflow would not happen.

 14, Sep 2019

 5.03 ms

Line 208 in File VantaToken.sol

208 `//@CTK NO_OVERFLOW`

Line 214-216 in File VantaToken.sol

```
214     function allowance(address owner, address spender) public view returns (uint256) {
215         return _allowed[owner][spender];
216     }
```


 The code meets the specification.



Formal Verification Request 15

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

 14, Sep 2019

 0.31 ms

Line 209 in File VantaToken.sol

209 `//@CTK NO_BUF_OVERFLOW`

Line 214-216 in File VantaToken.sol


```
214 function allowance(address owner, address spender) public view returns (uint256) {  
215     return _allowed[owner][spender];  
216 }
```

 The code meets the specification.

Formal Verification Request 16

Method will not encounter an assertion failure.

 14, Sep 2019

 0.31 ms

Line 210 in File VantaToken.sol

210 `//@CTK NO_ASF`

Line 214-216 in File VantaToken.sol


```
214 function allowance(address owner, address spender) public view returns (uint256) {  
215     return _allowed[owner][spender];  
216 }
```

 The code meets the specification.

Formal Verification Request 17

allowance correctness

 14, Sep 2019

 0.47 ms

Line 211-213 in File VantaToken.sol

```
211 /*@CTK "allowance correctness"  
212     @post __return == _allowed[owner][spender]  
213 */
```

Line 214-216 in File VantaToken.sol

```
214 function allowance(address owner, address spender) public view returns (uint256) {  
215     return _allowed[owner][spender];  
216 }
```


 The code meets the specification.



Formal Verification Request 18

If method completes, integer overflow would not happen.

 14, Sep 2019

 60.58 ms

Line 223 in File VantaToken.sol

223 `//@CTK NO_OVERFLOW`

Line 233-236 in File VantaToken.sol


```
233     function transfer(address to, uint256 value) public returns (bool) {
234         _transfer(msg.sender, to, value);
235         return true;
236     }
```

 The code meets the specification.

Formal Verification Request 19

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

 14, Sep 2019

 0.69 ms

Line 224 in File VantaToken.sol

224 `//@CTK NO_BUF_OVERFLOW`

Line 233-236 in File VantaToken.sol


```
233     function transfer(address to, uint256 value) public returns (bool) {
234         _transfer(msg.sender, to, value);
235         return true;
236     }
```

 The code meets the specification.

Formal Verification Request 20

Method will not encounter an assertion failure.

 14, Sep 2019

 24.28 ms

Line 225 in File VantaToken.sol

225 `//@CTK FAIL NO_ASF`

Line 233-236 in File VantaToken.sol

```
233     function transfer(address to, uint256 value) public returns (bool) {
234         _transfer(msg.sender, to, value);
235         return true;
236     }
```

 This code violates the specification.



```

1 Counter Example:
2 Before Execution:
3   Input = {
4     to = 2
5     value = 241
6   }
7   This = 0
8   Internal = {
9     __has_assertion_failure = false
10    __has_buf_overflow = false
11    __has_overflow = false
12    __has_returned = false
13    __reverted = false
14    msg = {
15      "gas": 0,
16      "sender": 0,
17      "value": 0
18    }
19  }
20  Other = {
21    __return = false
22    block = {
23      "number": 0,
24      "timestamp": 0
25    }
26  }
27  Address_Map = [
28    {
29      "key": 0,
30      "value": {
31        "contract_name": "ERC20",
32        "balance": 0,
33        "contract": {
34          "_balances": [
35            {
36              "key": 0,
37              "value": 0
38            },
39            {
40              "key": 2,
41              "value": 15
42            },
43            {
44              "key": 8,
45              "value": 0
46            },
47            {
48              "key": 64,
49              "value": 0
50            },
51            {
52              "key": "ALL_OTHERS",
53              "value": 2
54            }
55          ],
56          "_allowed": [
57            {
58              "key": 0,

```

```


59         "value": [
60             {
61                 "key": 0,
62                 "value": 64
63             },
64             {
65                 "key": "ALL_OTHERS",
66                 "value": 32
67             }
68         ],
69     },
70     {
71         "key": "ALL_OTHERS",
72         "value": [
73             {
74                 "key": "ALL_OTHERS",
75                 "value": 2
76             }
77         ]
78     }
79 ],
80 "_totalSupply": 0
81 }
82 }
83 },
84 {
85     "key": "ALL_OTHERS",
86     "value": "EmptyAddress"
87 }
88 ]
89
90 Function invocation is reverted.

```

Formal Verification Request 21

transfer

 14, Sep 2019

 22.76 ms

Line 226-232 in File VantaToken.sol

```

226  /*@CTK transfer
227      @tag assume_completion
228      @pre msg.sender != to
229      @post value <= _balances[msg.sender]
230      @post __post._balances[msg.sender] == _balances[msg.sender] - value
231      @post __post._balances[to] == _balances[to] + value
232  */

```

Line 233-236 in File VantaToken.sol

```

233  function transfer(address to, uint256 value) public returns (bool) {
234      _transfer(msg.sender, to, value);
235      return true;
236  }

```


 The code meets the specification.



Formal Verification Request 22

If method completes, integer overflow would not happen.

 14, Sep 2019

 52.2 ms

Line 247 in File VantaToken.sol

247 `//@CTK NO_OVERFLOW`

Line 254-257 in File VantaToken.sol


```
254     function approve(address spender, uint256 value) public returns (bool) {
255         _approve(msg.sender, spender, value);
256         return true;
257     }
```

 The code meets the specification.

Formal Verification Request 23

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

 14, Sep 2019

 0.59 ms

Line 248 in File VantaToken.sol

248 `//@CTK NO_BUF_OVERFLOW`

Line 254-257 in File VantaToken.sol


```
254     function approve(address spender, uint256 value) public returns (bool) {
255         _approve(msg.sender, spender, value);
256         return true;
257     }
```

 The code meets the specification.

Formal Verification Request 24

Method will not encounter an assertion failure.

 14, Sep 2019

 0.8 ms

Line 249 in File VantaToken.sol

249 `//@CTK NO_ASF`

Line 254-257 in File VantaToken.sol

```
254     function approve(address spender, uint256 value) public returns (bool) {
255         _approve(msg.sender, spender, value);
256         return true;
257     }
```


 The code meets the specification.



Formal Verification Request 25

approve

 14, Sep 2019

 4.97 ms

Line 250-253 in File VantaToken.sol

```
250  /*@CTK approve
251      @tag assume_completion
252      @post (__post._allowed[msg.sender][spender]) == (value)
253  */
```

Line 254-257 in File VantaToken.sol


```
254  function approve(address spender, uint256 value) public returns (bool) {
255      _approve(msg.sender, spender, value);
256      return true;
257  }
```

 The code meets the specification.

Formal Verification Request 26

If method completes, integer overflow would not happen.

 14, Sep 2019

 75.23 ms

Line 267 in File VantaToken.sol

```
267  //@CTK NO_OVERFLOW
```

Line 278-282 in File VantaToken.sol


```
278  function transferFrom(address from, address to, uint256 value) public returns (
279      bool) {
280      _transfer(from, to, value);
281      _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
282      return true;
283  }
```

 The code meets the specification.

Formal Verification Request 27

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

 14, Sep 2019

 9.2 ms

Line 268 in File VantaToken.sol

```
268  //@CTK NO_BUF_OVERFLOW
```

Line 278-282 in File VantaToken.sol

```

278 function transferFrom(address from, address to, uint256 value) public returns (
    bool) {
279     _transfer(from, to, value);
280     _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
281     return true;
282 }

```

✓ The code meets the specification.

Formal Verification Request 28

Method will not encounter an assertion failure.

📅 14, Sep 2019

🕒 88.7 ms

Line 269 in File VantaToken.sol

```

269 // @CTK FAIL NO_ASF

```

Line 278-282 in File VantaToken.sol

```

278 function transferFrom(address from, address to, uint256 value) public returns (
    bool) {
279     _transfer(from, to, value);
280     _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
281     return true;
282 }

```

✗ This code violates the specification.

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     from = 0
5     to = 16
6     value = 63
7   }
8   This = 0
9   Internal = {
10     __has_assertion_failure = false
11     __has_buf_overflow = false
12     __has_overflow = false
13     __has_returned = false
14     __reverted = false
15     msg = {
16       "gas": 0,
17       "sender": 0,
18       "value": 0
19     }
20   }
21   Other = {
22     __return = false
23     block = {
24       "number": 0,
25       "timestamp": 0
26     }
27   }

```

```

28 Address_Map = [
29     {
30         "key": 0,
31         "value": {
32             "contract_name": "ERC20",
33             "balance": 0,
34             "contract": {
35                 "_balances": [
36                     {
37                         "key": 128,
38                         "value": 0
39                     },
40                     {
41                         "key": 0,
42                         "value": 0
43                     },
44                     {
45                         "key": 9,
46                         "value": 1
47                     },
48                     {
49                         "key": 4,
50                         "value": 0
51                     },
52                     {
53                         "key": 64,
54                         "value": 0
55                     },
56                     {
57                         "key": 8,
58                         "value": 0
59                     },
60                     {
61                         "key": 32,
62                         "value": 0
63                     },
64                     {
65                         "key": 1,
66                         "value": 0
67                     },
68                     {
69                         "key": "ALL_OTHERS",
70                         "value": 16
71                     }
72                 ],
73                 "_allowed": [
74                     {
75                         "key": 0,
76                         "value": [
77                             {
78                                 "key": 0,
79                                 "value": 40
80                             },
81                             {
82                                 "key": 4,
83                                 "value": 4
84                             },
85                             {

```



```


86         "key": 1,
87         "value": 64
88     },
89     {
90         "key": "ALL_OTHERS",
91         "value": 16
92     }
93 ]
94 },
95 {
96     "key": 32,
97     "value": [
98         {
99             "key": 0,
100            "value": 16
101        },
102        {
103            "key": "ALL_OTHERS",
104            "value": 64
105        }
106    ]
107 },
108 {
109     "key": "ALL_OTHERS",
110     "value": [
111         {
112             "key": "ALL_OTHERS",
113             "value": 16
114         }
115     ]
116 }
117 ],
118 "_totalSupply": 0
119 }
120 }
121 },
122 {
123     "key": "ALL_OTHERS",
124     "value": "EmptyAddress"
125 }
126 ]
127
128 Function invocation is reverted.

```

Formal Verification Request 29

transferFrom correctness

 14, Sep 2019

 312.51 ms

Line 270-277 in File VantaToken.sol

```

270 /*@CTK "transferFrom correctness"
271     @tag assume_completion
272     @post value <= _balances[from] && value <= _allowed[from][msg.sender]
273     @post to != from -> __post._balances[from] == _balances[from] - value

```

```

274     @post to != from -> __post._balances[to] == _balances[to] + value
275     @post to == from -> __post._balances[from] == _balances[from]
276     @post __post._allowed[from][msg.sender] == _allowed[from][msg.sender] - value
277     */

```

Line 278-282 in File VantaToken.sol

```

278     function transferFrom(address from, address to, uint256 value) public returns (
        bool) {
279         _transfer(from, to, value);
280         _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
281         return true;
282     }

```

✓ The code meets the specification.

Formal Verification Request 30

If method completes, integer overflow would not happen.

📅 14, Sep 2019

🕒 33.63 ms

Line 294 in File VantaToken.sol

```

294     //@CTK NO_OVERFLOW

```

Line 301-304 in File VantaToken.sol

```

301     function increaseAllowance(address spender, uint256 addedValue) public returns (
        bool) {
302         _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
303         return true;
304     }

```

✓ The code meets the specification.

Formal Verification Request 31

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

📅 14, Sep 2019

🕒 0.6 ms

Line 295 in File VantaToken.sol

```

295     //@CTK NO_BUF_OVERFLOW

```

Line 301-304 in File VantaToken.sol

```

301     function increaseAllowance(address spender, uint256 addedValue) public returns (
        bool) {
302         _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
303         return true;
304     }


```

✓ The code meets the specification.

Formal Verification Request 32

Method will not encounter an assertion failure.

 14, Sep 2019

 14.85 ms

Line 296 in File VantaToken.sol

296 `//@CTK FAIL NO_ASF`

Line 301-304 in File VantaToken.sol

```
301     function increaseAllowance(address spender, uint256 addedValue) public returns (
302         bool) {
303         _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
304         return true;
305     }
```

 This code violates the specification.

```
1 Counter Example:
2 Before Execution:
3   Input = {
4     addedValue = 160
5     spender = 8
6   }
7   This = 0
8   Internal = {
9     __has_assertion_failure = false
10    __has_buf_overflow = false
11    __has_overflow = false
12    __has_returned = false
13    __reverted = false
14    msg = {
15      "gas": 0,
16      "sender": 128,
17      "value": 0
18    }
19  }
20  Other = {
21    __return = false
22    block = {
23      "number": 0,
24      "timestamp": 0
25    }
26  }
27  Address_Map = [
28    {
29      "key": 0,
30      "value": {
31        "contract_name": "ERC20",
32        "balance": 0,
33        "contract": {
34          "_balances": [
35            {
36              "key": 8,
37              "value": 16
38            },
39            {
```

```


40         "key": "ALL_OTHERS",
41         "value": 72
42     },
43 ],
44 "_allowed": [
45     {
46         "key": 128,
47         "value": [
48             {
49                 "key": 8,
50                 "value": 168
51             },
52             {
53                 "key": "ALL_OTHERS",
54                 "value": 160
55             }
56         ]
57     },
58     {
59         "key": "ALL_OTHERS",
60         "value": [
61             {
62                 "key": "ALL_OTHERS",
63                 "value": 72
64             }
65         ]
66     }
67 ],
68 "_totalSupply": 0
69 }
70 }
71 },
72 {
73     "key": "ALL_OTHERS",
74     "value": "EmptyAddress"
75 }
76 ]
77
78 Function invocation is reverted.

```

Formal Verification Request 33

increaseApproval correctness

 14, Sep 2019

 4.72 ms

Line 297-300 in File VantaToken.sol

```

297  /*@CTK "increaseApproval correctness"
298     @tag assume_completion
299     @post __post._allowed[msg.sender][spender] == _allowed[msg.sender][spender] +
        addedValue
300  */

```

Line 301-304 in File VantaToken.sol

```

301     function increaseAllowance(address spender, uint256 addedValue) public returns (
302         bool) {
303         _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
304         return true;
305     }


```

✓ The code meets the specification.

Formal Verification Request 34

If method completes, integer overflow would not happen.

 14, Sep 2019

 40.35 ms

Line 316 in File VantaToken.sol

```

316     //@CTK NO_OVERFLOW

```

Line 330-333 in File VantaToken.sol

```

330     function decreaseAllowance(address spender, uint256 subtractedValue) public
331         returns (bool) {
332         _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue));
333         return true;
334     }


```

✓ The code meets the specification.

Formal Verification Request 35

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

 14, Sep 2019

 0.8 ms

Line 317 in File VantaToken.sol

```

317     //@CTK NO_BUF_OVERFLOW

```

Line 330-333 in File VantaToken.sol

```

330     function decreaseAllowance(address spender, uint256 subtractedValue) public
331         returns (bool) {
332         _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue));
333         return true;
334     }


```

✓ The code meets the specification.

Formal Verification Request 36

Method will not encounter an assertion failure.

 14, Sep 2019

 14.74 ms

Line 318 in File VantaToken.sol

318 `//@CTK FAIL NO_ASF`

Line 330-333 in File VantaToken.sol

```
330     function decreaseAllowance(address spender, uint256 subtractedValue) public
331         returns (bool) {
332         _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue
333             ));
332         return true;
333     }
```

 This code violates the specification.

```
1 Counter Example:
2 Before Execution:
3   Input = {
4     spender = 8
5     subtractedValue = 219
6   }
7   This = 0
8   Internal = {
9     __has_assertion_failure = false
10    __has_buf_overflow = false
11    __has_overflow = false
12    __has_returned = false
13    __reverted = false
14    msg = {
15      "gas": 0,
16      "sender": 64,
17      "value": 0
18    }
19  }
20  Other = {
21    __return = false
22    block = {
23      "number": 0,
24      "timestamp": 0
25    }
26  }
27  Address_Map = [
28    {
29      "key": 0,
30      "value": {
31        "contract_name": "ERC20",
32        "balance": 0,
33        "contract": {
34          "_balances": [
35            {
36              "key": 0,
37              "value": 8
38            },
```

```

39     {
40         "key": "ALL_OTHERS",
41         "value": 219
42     }
43 ],
44 "_allowed": [
45     {
46         "key": 64,
47         "value": [
48             {
49                 "key": 0,
50                 "value": 0
51             },
52             {
53                 "key": 8,
54                 "value": 0
55             },
56             {
57                 "key": "ALL_OTHERS",
58                 "value": 219
59             }
60         ]
61     },
62     {
63         "key": 0,
64         "value": [
65             {
66                 "key": "ALL_OTHERS",
67                 "value": 219
68             }
69         ]
70     },
71     {
72         "key": "ALL_OTHERS",
73         "value": [
74             {
75                 "key": "ALL_OTHERS",
76                 "value": 219
77             }
78         ]
79     }
80 ],
81 "_totalSupply": 0
82 }
83 }
84 },
85 {
86     "key": "ALL_OTHERS",
87     "value": "EmptyAddress"
88 }
89 ]


```

91 Function invocation is reverted.

Formal Verification Request 37

decreaseApproval0

 14, Sep 2019

 21.82 ms

Line 319-323 in File VantaToken.sol

```
319  /*@CTK decreaseApproval0
320     @pre __return == true
321     @pre _allowed[msg.sender][spender] <= subtractedValue
322     @post __post._allowed[msg.sender][spender] == 0
323  */
```

Line 330-333 in File VantaToken.sol


```
330  function decreaseAllowance(address spender, uint256 subtractedValue) public
      returns (bool) {
331      _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue)
        );
332      return true;
333  }
```

 The code meets the specification.

Formal Verification Request 38

decreaseApproval

 14, Sep 2019

 4.88 ms

Line 324-329 in File VantaToken.sol

```
324  /*@CTK decreaseApproval
325     @pre __return == true
326     @pre _allowed[msg.sender][spender] > subtractedValue
327     @post __post._allowed[msg.sender][spender] ==
328           _allowed[msg.sender][spender] - subtractedValue
329  */
```

Line 330-333 in File VantaToken.sol


```
330  function decreaseAllowance(address spender, uint256 subtractedValue) public
      returns (bool) {
331      _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue)
        );
332      return true;
333  }
```

 The code meets the specification.

Formal Verification Request 39

If method completes, integer overflow would not happen.

 14, Sep 2019

 2.24 ms



Line 341 in File VantaToken.sol

341 `//@CTK NO_OVERFLOW`

Line 351-356 in File VantaToken.sol


```
351     function _transfer(address from, address to, uint256 value) internal {
352
353         _balances[from] = _balances[from].sub(value);
354         _balances[to] = _balances[to].add(value);
355         emit Transfer(from, to, value);
356     }
```

✓ The code meets the specification.

Formal Verification Request 40

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

 14, Sep 2019

 0.54 ms

Line 342 in File VantaToken.sol

342 `//@CTK NO_BUF_OVERFLOW`

Line 351-356 in File VantaToken.sol


```
351     function _transfer(address from, address to, uint256 value) internal {
352
353         _balances[from] = _balances[from].sub(value);
354         _balances[to] = _balances[to].add(value);
355         emit Transfer(from, to, value);
356     }
```

✓ The code meets the specification.

Formal Verification Request 41

Method will not encounter an assertion failure.

 14, Sep 2019

 27.66 ms

Line 343 in File VantaToken.sol

343 `//@CTK FAIL NO_ASF`

Line 351-356 in File VantaToken.sol

```
351     function _transfer(address from, address to, uint256 value) internal {
352
353         _balances[from] = _balances[from].sub(value);
354         _balances[to] = _balances[to].add(value);
355         emit Transfer(from, to, value);
356     }
```

✗ This code violates the specification.

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     from = 1
5     to = 0
6     value = 227
7   }
8   This = 0
9   Internal = {
10    __has_assertion_failure = false
11    __has_buf_overflow = false
12    __has_overflow = false
13    __has_returned = false
14    __reverted = false
15    msg = {
16      "gas": 0,
17      "sender": 0,
18      "value": 0
19    }
20  }
21  Other = {
22    block = {
23      "number": 0,
24      "timestamp": 0
25    }
26  }
27  Address_Map = [
28    {
29      "key": 0,
30      "value": {
31        "contract_name": "ERC20",
32        "balance": 0,
33        "contract": {
34          "_balances": [
35            {
36              "key": 32,
37              "value": 0
38            },
39            {
40              "key": 8,
41              "value": 0
42            },
43            {
44              "key": 0,
45              "value": 157
46            },
47            {
48              "key": 129,
49              "value": 0
50            },
51            {
52              "key": 128,
53              "value": 0
54            },
55            {
56              "key": 1,
57              "value": 0
58            },

```

```

59     {
60         "key": "ALL_OTHERS",
61         "value": 227
62     }
63 ],
64 "_allowed": [
65     {
66         "key": 0,
67         "value": [
68             {
69                 "key": 0,
70                 "value": 0
71             },
72             {
73                 "key": "ALL_OTHERS",
74                 "value": 64
75             }
76         ]
77     },
78     {
79         "key": "ALL_OTHERS",
80         "value": [
81             {
82                 "key": "ALL_OTHERS",
83                 "value": 227
84             }
85         ]
86     }
87 ],
88 "_totalSupply": 0
89 }
90 }
91 },
92 {
93     "key": "ALL_OTHERS",
94     "value": "EmptyAddress"
95 }
96 ]


```

97
98 Function invocation is reverted.

Formal Verification Request 42

__transfer

 14, Sep 2019

 25.0 ms

Line 344-350 in File VantaToken.sol

```

344  /*@CTK __transfer
345      @tag assume_completion
346      @pre from != to
347      @post value <= _balances[from]
348      @post __post._balances[from] == _balances[from] - value
349      @post __post._balances[to] == _balances[to] + value
350  */

```

Line 351-356 in File VantaToken.sol


```
351     function _transfer(address from, address to, uint256 value) internal {
352
353         _balances[from] = _balances[from].sub(value);
354         _balances[to] = _balances[to].add(value);
355         emit Transfer(from, to, value);
356     }
```

✓ The code meets the specification.

Formal Verification Request 43

If method completes, integer overflow would not happen.

 14, Sep 2019

 32.92 ms

Line 365 in File VantaToken.sol

```
365     //@CTK NO_OVERFLOW
```

Line 374-380 in File VantaToken.sol


```
374     function _mint(address account, uint256 value) internal {
375         require(account != address(0));
376
377         _totalSupply = _totalSupply.add(value);
378         _balances[account] = _balances[account].add(value);
379         emit Transfer(address(0), account, value);
380     }
```

✓ The code meets the specification.

Formal Verification Request 44

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

 14, Sep 2019

 6.19 ms

Line 366 in File VantaToken.sol

```
366     //@CTK NO_BUF_OVERFLOW
```

Line 374-380 in File VantaToken.sol


```
374     function _mint(address account, uint256 value) internal {
375         require(account != address(0));
376
377         _totalSupply = _totalSupply.add(value);
378         _balances[account] = _balances[account].add(value);
379         emit Transfer(address(0), account, value);
380     }
```

✓ The code meets the specification.

Formal Verification Request 45

Method will not encounter an assertion failure.

 14, Sep 2019

 24.9 ms

Line 367 in File VantaToken.sol

367 `//@CTK FAIL NO_ASF`

Line 374-380 in File VantaToken.sol

```
374 function _mint(address account, uint256 value) internal {
375     require(account != address(0));
376
377     _totalSupply = _totalSupply.add(value);
378     _balances[account] = _balances[account].add(value);
379     emit Transfer(address(0), account, value);
380 }
```

 This code violates the specification.

```
1 Counter Example:
2 Before Execution:
3   Input = {
4     account = 1
5     value = 111
6   }
7   This = 0
8   Internal = {
9     __has_assertion_failure = false
10    __has_buf_overflow = false
11    __has_overflow = false
12    __has_returned = false
13    __reverted = false
14    msg = {
15      "gas": 0,
16      "sender": 0,
17      "value": 0
18    }
19  }
20  Other = {
21    block = {
22      "number": 0,
23      "timestamp": 0
24    }
25  }
26  Address_Map = [
27    {
28      "key": 0,
29      "value": {
30        "contract_name": "ERC20",
31        "balance": 0,
32        "contract": {
33          "_balances": [
34            {
35              "key": 64,
36              "value": 2
37            },
```

```

38     {
39         "key": 2,
40         "value": 0
41     },
42     {
43         "key": 0,
44         "value": 32
45     },
46     {
47         "key": 32,
48         "value": 4
49     },
50     {
51         "key": "ALL_OTHERS",
52         "value": 1
53     }
54 ],
55 "_allowed": [
56     {
57         "key": 0,
58         "value": [
59             {
60                 "key": 0,
61                 "value": 128
62             },
63             {
64                 "key": "ALL_OTHERS",
65                 "value": 112
66             }
67         ]
68     },
69     {
70         "key": "ALL_OTHERS",
71         "value": [
72             {
73                 "key": "ALL_OTHERS",
74                 "value": 1
75             }
76         ]
77     }
78 ],
79 "_totalSupply": 193
80 }
81 }
82 },
83 {
84     "key": "ALL_OTHERS",
85     "value": "EmptyAddress"
86 }
87 ]

```

89 Function invocation is reverted.



Formal Verification Request 46

mint

📅 14, Sep 2019

🕒 24.51 ms

Line 368-373 in File VantaToken.sol

```
368  /*@CTK mint
369      @tag assume_completion
370      @post account != address(0)
371      @post (_post._totalSupply) == (_totalSupply + value)
372      @post (_post._balances[account]) == (_balances[account] + value)
373  */
```

Line 374-380 in File VantaToken.sol

```
374  function _mint(address account, uint256 value) internal {
375      require(account != address(0));
376
377      _totalSupply = _totalSupply.add(value);
378      _balances[account] = _balances[account].add(value);
379      emit Transfer(address(0), account, value);
380  }
```

✅ The code meets the specification.

Formal Verification Request 47

If method completes, integer overflow would not happen.

📅 14, Sep 2019

🕒 29.84 ms

Line 389 in File VantaToken.sol

```
389  //@CTK NO_OVERFLOW
```

Line 399-405 in File VantaToken.sol

```
399  function _burn(address account, uint256 value) internal {
400      require(account != address(0));
401
402      _totalSupply = _totalSupply.sub(value);
403      _balances[account] = _balances[account].sub(value);
404      emit Transfer(account, address(0), value);
405  }
```

✅ The code meets the specification.

Formal Verification Request 48

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

📅 14, Sep 2019

🕒 5.55 ms

Line 390 in File VantaToken.sol

```
390 // @CTK_NO_BUF_OVERFLOW
```

Line 399-405 in File VantaToken.sol

```
399 function _burn(address account, uint256 value) internal {
400     require(account != address(0));
401
402     _totalSupply = _totalSupply.sub(value);
403     _balances[account] = _balances[account].sub(value);
404     emit Transfer(account, address(0), value);
405 }
```

✓ The code meets the specification.

Formal Verification Request 49

Method will not encounter an assertion failure.

📅 14, Sep 2019

🕒 18.11 ms

Line 391 in File VantaToken.sol

```
391 // @CTK_FAIL_NO_ASF
```

Line 399-405 in File VantaToken.sol

```
399 function _burn(address account, uint256 value) internal {
400     require(account != address(0));
401
402     _totalSupply = _totalSupply.sub(value);
403     _balances[account] = _balances[account].sub(value);
404     emit Transfer(account, address(0), value);
405 }
```

✗ This code violates the specification.

```
1 Counter Example:
2 Before Execution:
3   Input = {
4     account = 8
5     value = 16
6   }
7   This = 0
8   Internal = {
9     __has_assertion_failure = false
10    __has_buf_overflow = false
11    __has_overflow = false
12    __has_returned = false
13    __reverted = false
14    msg = {
15      "gas": 0,
16      "sender": 0,
17      "value": 0
18    }
19  }
20  Other = {
```



```


21     block = {
22         "number": 0,
23         "timestamp": 0
24     }
25 }
26 Address_Map = [
27     {
28         "key": 0,
29         "value": {
30             "contract_name": "ERC20",
31             "balance": 0,
32             "contract": {
33                 "_balances": [
34                     {
35                         "key": 0,
36                         "value": 0
37                     },
38                     {
39                         "key": 8,
40                         "value": 0
41                     },
42                     {
43                         "key": "ALL_OTHERS",
44                         "value": 240
45                     }
46                 ],
47                 "_allowed": [
48                     {
49                         "key": "ALL_OTHERS",
50                         "value": [
51                             {
52                                 "key": "ALL_OTHERS",
53                                 "value": 240
54                             }
55                         ]
56                     }
57                 ],
58                 "_totalSupply": 128
59             }
60         },
61     ],
62     {
63         "key": "ALL_OTHERS",
64         "value": "EmptyAddress"
65     }
66 ]
67
68 Function invocation is reverted.

```

Formal Verification Request 50

burn

 14, Sep 2019

 35.87 ms



Line 392-398 in File VantaToken.sol

```
392  /*@CTK burn
393      @tag assume_completion
394      @post account != address(0)
395      @post (value <= _balances[account])
396      @post (__post._totalSupply) == (_totalSupply - value)
397      @post (__post._balances[account]) == (_balances[account] - value)
398  */
```

Line 399-405 in File VantaToken.sol

```
399  function _burn(address account, uint256 value) internal {
400      require(account != address(0));
401
402      _totalSupply = _totalSupply.sub(value);
403      _balances[account] = _balances[account].sub(value);
404      emit Transfer(account, address(0), value);
405  }
```

✓ The code meets the specification.

Formal Verification Request 51

If method completes, integer overflow would not happen.

📅 14, Sep 2019

🕒 0.52 ms

Line 414 in File VantaToken.sol

```
414  //@CTK NO_OVERFLOW
```

Line 423-429 in File VantaToken.sol

```
423  function _approve(address owner, address spender, uint256 value) internal {
424      require(spender != address(0));
425      require(owner != address(0));
426
427      _allowed[owner][spender] = value;
428      emit Approval(owner, spender, value);
429  }
```

✓ The code meets the specification.

Formal Verification Request 52

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

📅 14, Sep 2019

🕒 0.45 ms

Line 415 in File VantaToken.sol

```
415  //@CTK NO_BUF_OVERFLOW
```

Line 423-429 in File VantaToken.sol



```

423     function _approve(address owner, address spender, uint256 value) internal {
424         require(spender != address(0));
425         require(owner != address(0));
426
427         _allowed[owner][spender] = value;
428         emit Approval(owner, spender, value);
429     }

```

✓ The code meets the specification.

Formal Verification Request 53

Method will not encounter an assertion failure.

📅 14, Sep 2019

🕒 0.48 ms

Line 416 in File VantaToken.sol

```

416     //@CTK NO_ASF

```

Line 423-429 in File VantaToken.sol

```

423     function _approve(address owner, address spender, uint256 value) internal {
424         require(spender != address(0));
425         require(owner != address(0));
426
427         _allowed[owner][spender] = value;
428         emit Approval(owner, spender, value);
429     }

```

✓ The code meets the specification.

Formal Verification Request 54

__approve

📅 14, Sep 2019

🕒 2.62 ms

Line 417-422 in File VantaToken.sol

```

417     /*@CTK __approve
418         @tag assume_completion
419         @post spender != address(0)
420         @post owner != address(0)
421         @post __post._allowed[owner][spender] == value
422     */

```

Line 423-429 in File VantaToken.sol

```

423     function _approve(address owner, address spender, uint256 value) internal {
424         require(spender != address(0));
425         require(owner != address(0));
426
427         _allowed[owner][spender] = value;
428         emit Approval(owner, spender, value);
429     }

```



✓ The code meets the specification.

Formal Verification Request 55

If method completes, integer overflow would not happen.

📅 14, Sep 2019

🕒 68.39 ms

Line 439 in File VantaToken.sol

439 `//@CTK NO_OVERFLOW`

Line 450-453 in File VantaToken.sol

```
450 function _burnFrom(address account, uint256 value) internal {  
451     _burn(account, value);  
452     _approve(account, msg.sender, _allowed[account][msg.sender].sub(value));  
453 }
```

✓ The code meets the specification.

Formal Verification Request 56

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

📅 14, Sep 2019

🕒 9.4 ms

Line 440 in File VantaToken.sol

440 `//@CTK NO_BUF_OVERFLOW`

Line 450-453 in File VantaToken.sol

```
450 function _burnFrom(address account, uint256 value) internal {  
451     _burn(account, value);  
452     _approve(account, msg.sender, _allowed[account][msg.sender].sub(value));  
453 }
```

✓ The code meets the specification.

Formal Verification Request 57

Method will not encounter an assertion failure.

📅 14, Sep 2019

🕒 60.69 ms

Line 441 in File VantaToken.sol

441 `//@CTK FAIL NO_ASF`

Line 450-453 in File VantaToken.sol

```

450 function _burnFrom(address account, uint256 value) internal {
451     _burn(account, value);
452     _approve(account, msg.sender, _allowed[account][msg.sender].sub(value));
453 }

```

✗ This code violates the specification.

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     account = 64
5     value = 224
6   }
7   This = 0
8   Internal = {
9     __has_assertion_failure = false
10    __has_buf_overflow = false
11    __has_overflow = false
12    __has_returned = false
13    __reverted = false
14    msg = {
15      "gas": 0,
16      "sender": 0,
17      "value": 0
18    }
19  }
20  Other = {
21    block = {
22      "number": 0,
23      "timestamp": 0
24    }
25  }
26  Address_Map = [
27    {
28      "key": 0,
29      "value": {
30        "contract_name": "ERC20",
31        "balance": 0,
32        "contract": {
33          "_balances": [
34            {
35              "key": 8,
36              "value": 0
37            },
38            {
39              "key": 0,
40              "value": 0
41            },
42            {
43              "key": 1,
44              "value": 0
45            },
46            {
47              "key": 64,
48              "value": 0
49            },
50            {
51              "key": "ALL_OTHERS",
52              "value": 32

```

```


53     }
54   ],
55   "_allowed": [
56     {
57       "key": 0,
58       "value": [
59         {
60           "key": 0,
61           "value": 0
62         },
63         {
64           "key": "ALL_OTHERS",
65           "value": 32
66         }
67       ]
68     },
69     {
70       "key": "ALL_OTHERS",
71       "value": [
72         {
73           "key": "ALL_OTHERS",
74           "value": 32
75         }
76       ]
77     }
78   ],
79   "_totalSupply": 224
80 }
81 }
82 },
83 {
84   "key": "ALL_OTHERS",
85   "value": "EmptyAddress"
86 }
87 ]
88
89 Function invocation is reverted.

```

Formal Verification Request 58

burnFrom

 14, Sep 2019

 129.4 ms

Line 442-449 in File VantaToken.sol

```

442   /*@CTK burnFrom
443     @tag assume_completion
444     @post (value <= _allowed[account][msg.sender])
445     @post (value <= _balances[account])
446     @post (__post._allowed[account][msg.sender]) == (_allowed[account][msg.sender] -
         value)
447     @post (__post._balances[account]) == (_balances[account] - (value))
448     @post __post._totalSupply == (_totalSupply - value)
449   */

```

Line 450-453 in File VantaToken.sol


```
450     function _burnFrom(address account, uint256 value) internal {
451         _burn(account, value);
452         _approve(account, msg.sender, _allowed[account][msg.sender].sub(value));
453     }
```

✓ The code meets the specification.

Formal Verification Request 59

Ownable

 14, Sep 2019

 4.74 ms

Line 509-512 in File VantaToken.sol

```
509     /*@CTK Ownable
510         @tag assume_completion
511         @post __post._owner == msg.sender
512     */
```

Line 513-516 in File VantaToken.sol


```
513     constructor () internal {
514         _owner = msg.sender;
515         emit OwnershipTransferred(address(0), _owner);
516     }
```

✓ The code meets the specification.

Formal Verification Request 60

isOwner

 14, Sep 2019

 4.81 ms

Line 536-539 in File VantaToken.sol

```
536     /*@CTK isOwner
537         @tag assume_completion
538         @post __return == (msg.sender == _owner)
539     */
```

Line 540-542 in File VantaToken.sol

```
540     function isOwner() public view returns (bool) {
541         return msg.sender == _owner;
542     }
```


✓ The code meets the specification.



Formal Verification Request 61

renounceOwnership

 14, Sep 2019

 20.2 ms

Line 551-555 in File VantaToken.sol

```
551  /*@CTK renounceOwnership
552      @tag assume_completion
553      @post msg.sender == _owner
554      @post __post._owner == address(0)
555  */
```

Line 556-559 in File VantaToken.sol


```
556  function renounceOwnership() public onlyOwner {
557      emit OwnershipTransferred(_owner, address(0));
558      _owner = address(0);
559  }
```

 The code meets the specification.

Formal Verification Request 62

transferOwnership

 14, Sep 2019

 52.88 ms

Line 565-570 in File VantaToken.sol

```
565  /*@CTK transferOwnership
566      @tag assume_completion
567      @pre msg.sender == _owner
568      @pre newOwner != address(0)
569      @post __post._owner == newOwner
570  */
```

Line 571-573 in File VantaToken.sol


```
571  function transferOwnership(address newOwner) public onlyOwner {
572      _transferOwnership(newOwner);
573  }
```

 The code meets the specification.

Formal Verification Request 63

__transferOwnership

 14, Sep 2019

 1.37 ms

Line 579-584 in File VantaToken.sol



```
579  /*@CTK _transferOwnership
580      @tag assume_completion
581      @pre msg.sender == _owner
582      @pre newOwner != address(0)
583      @post __post._owner == newOwner
584  */
```

Line 585-589 in File VantaToken.sol

```
585  function _transferOwnership(address newOwner) internal {
586      require(newOwner != address(0));
587      emit OwnershipTransferred(_owner, newOwner);
588      _owner = newOwner;
589  }
```

✓ The code meets the specification.

Source Code with CertiK Labels

File VantaToken.sol

```

1  /**
2   *Submitted for verification at Etherscan.io on 2019-08-16
3   */
4
5  /**
6   * Copyright 2019 Vanta Network.
7   *
8   * Licensed under the Apache License, Version 2.0 (the "License");
9   * you may not use this file except in compliance with the License.
10  * You may obtain a copy of the License at
11  *
12  *    http://www.apache.org/licenses/LICENSE-2.0
13  *
14  * Unless required by applicable law or agreed to in writing, software
15  * distributed under the License is distributed on an "AS IS" BASIS,
16  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
17  * See the License for the specific language governing permissions and
18  * limitations under the License.
19  */
20
21 pragma solidity ^0.5.2;
22
23 /**
24  * @title ERC20 interface
25  * @dev see https://eips.ethereum.org/EIPS/eip-20
26  */
27 interface IERC20 {
28     function transfer(address to, uint256 value) external returns (bool);
29
30     function approve(address spender, uint256 value) external returns (bool);
31
32     function transferFrom(address from, address to, uint256 value) external returns (
33         bool);
34
35     function totalSupply() external view returns (uint256);
36
37     function balanceOf(address who) external view returns (uint256);
38
39     function allowance(address owner, address spender) external view returns (uint256)
40         ;
41
42     event Transfer(address indexed from, address indexed to, uint256 value);
43
44     event Approval(address indexed owner, address indexed spender, uint256 value);
45 }
46
47 /**
48  * @title SafeMath
49  * @dev Unsigned math operations with safety checks that revert on error
50  */
51 library SafeMath {
52     /**
53      * @dev Multiplies two unsigned integers, reverts on overflow.
54      */

```

```

53  /*@CTK "SafeMath mul zero"
54      @tag spec
55      @tag is_pure
56      @pre (a == 0)
57      @post __return == 0
58  */
59  /*@CTK "SafeMath mul nonzero"
60      @tag spec
61      @tag is_pure
62      @pre (a != 0)
63      @post (a * b / a != b) == __reverted
64      @post !__reverted -> __return == a * b
65      @post !__reverted -> !__has_overflow
66      @post !__reverted -> !__has_assertion_failure
67      @post !(__has_buf_overflow)
68  */
69  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
70      // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
71      // benefit is lost if 'b' is also tested.
72      // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
73      if (a == 0) {
74          return 0;
75      }
76
77      uint256 c = a * b;
78      require(c / a == b);
79
80      return c;
81  }
82
83  /**
84   * @dev Integer division of two unsigned integers truncating the quotient, reverts
85   *       on division by zero.
86   */
87  /*@CTK "SafeMath div"
88      @tag spec
89      @tag is_pure
90      @post (b == 0) == __reverted
91      @post !__reverted -> __return == a / b
92      @post !__reverted -> !__has_overflow
93      @post !__reverted -> !__has_assertion_failure
94      @post !(__has_buf_overflow)
95  */
96  function div(uint256 a, uint256 b) internal pure returns (uint256) {
97      // Solidity only automatically asserts when dividing by 0
98      require(b > 0);
99      uint256 c = a / b;
100      // assert(a == b * c + a % b); // There is no case in which this doesn't hold
101
102      return c;
103  }
104
105  /**
106   * @dev Subtracts two unsigned integers, reverts on overflow (i.e. if subtrahend
107   *       is greater than minuend).
108   */
109  /*@CTK "SafeMath sub"
110      @tag spec

```



```

109     @tag is_pure
110     @post (b > a) == __reverted
111     @post !__reverted -> __return == a - b
112     @post !__reverted -> !__has_overflow
113     @post !__reverted -> !__has_assertion_failure
114     @post !(__has_buf_overflow)
115     */
116     function sub(uint256 a, uint256 b) internal pure returns (uint256) {
117         require(b <= a);
118         uint256 c = a - b;
119
120         return c;
121     }
122
123     /**
124     * @dev Adds two unsigned integers, reverts on overflow.
125     */
126     /*@CTK "SafeMath add"
127     @tag spec
128     @tag is_pure
129     @post (a + b < a || a + b < b) == __reverted
130     @post !__reverted -> __return == a + b
131     @post !__reverted -> !__has_overflow
132     @post !__reverted -> !__has_assertion_failure
133     @post !(__has_buf_overflow)
134     */
135     function add(uint256 a, uint256 b) internal pure returns (uint256) {
136         uint256 c = a + b;
137         require(c >= a);
138
139         return c;
140     }
141
142     /**
143     * @dev Divides two unsigned integers and returns the remainder (unsigned integer
144         modulo),
145     * reverts when dividing by zero.
146     */
147     function mod(uint256 a, uint256 b) internal pure returns (uint256) {
148         require(b != 0);
149         return a % b;
150     }
151
152     /**
153     * @title Standard ERC20 token
154     *
155     * @dev Implementation of the basic standard token.
156     * https://eips.ethereum.org/EIPS/eip-20
157     * Originally based on code by FirstBlood:
158     * https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.
159     * sol
160     *
161     * This implementation emits additional Approval events, allowing applications to
162         reconstruct the allowance status for
163     * all accounts just by listening to said events. Note that this isn't required by the
164         specification, and other
165     * compliant implementations may not do it.

```



```

163  */
164  contract ERC20 is IERC20 {
165      using SafeMath for uint256;
166
167      mapping (address => uint256) private _balances;
168
169      mapping (address => mapping (address => uint256)) private _allowed;
170
171      uint256 private _totalSupply;
172
173      /**
174       * @dev Total number of tokens in existence
175       */
176      //@CTK NO_OVERFLOW
177      //@CTK NO_BUF_OVERFLOW
178      //@CTK NO_ASF
179      /*@CTK totalSupply
180       @tag assume_completion
181       @post (__return) == (_totalSupply)
182       */
183      function totalSupply() public view returns (uint256) {
184          return _totalSupply;
185      }
186
187      /**
188       * @dev Gets the balance of the specified address.
189       * @param owner The address to query the balance of.
190       * @return A uint256 representing the amount owned by the passed address.
191       */
192      //@CTK NO_OVERFLOW
193      //@CTK NO_BUF_OVERFLOW
194      //@CTK NO_ASF
195      /*@CTK balanceOf
196       @post __return == __post._balances[owner]
197       */
198      function balanceOf(address owner) public view returns (uint256) {
199          return _balances[owner];
200      }
201
202      /**
203       * @dev Function to check the amount of tokens that an owner allowed to a spender.
204       * @param owner address The address which owns the funds.
205       * @param spender address The address which will spend the funds.
206       * @return A uint256 specifying the amount of tokens still available for the
207           spender.
208       */
209      //@CTK NO_OVERFLOW
210      //@CTK NO_BUF_OVERFLOW
211      //@CTK NO_ASF
212      /*@CTK "allowance correctness"
213       @post __return == _allowed[owner][spender]
214       */
215      function allowance(address owner, address spender) public view returns (uint256) {
216          return _allowed[owner][spender];
217      }
218
219      /**
220       * @dev Transfer token to a specified address

```



```

220     * @param to The address to transfer to.
221     * @param value The amount to be transferred.
222     */
223     //@CTK NO_OVERFLOW
224     //@CTK NO_BUF_OVERFLOW
225     //@CTK FAIL NO_ASF
226     /*@CTK transfer
227         @tag assume_completion
228         @pre msg.sender != to
229         @post value <= _balances[msg.sender]
230         @post __post._balances[msg.sender] == _balances[msg.sender] - value
231         @post __post._balances[to] == _balances[to] + value
232     */
233     function transfer(address to, uint256 value) public returns (bool) {
234         _transfer(msg.sender, to, value);
235         return true;
236     }
237
238     /**
239     * @dev Approve the passed address to spend the specified amount of tokens on
240     *       behalf of msg.sender.
241     * Beware that changing an allowance with this method brings the risk that someone
242     * may use both the old
243     * and the new allowance by unfortunate transaction ordering. One possible
244     * solution to mitigate this
245     * race condition is to first reduce the spender's allowance to 0 and set the
246     * desired value afterwards:
247     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
248     * @param spender The address which will spend the funds.
249     * @param value The amount of tokens to be spent.
250     */
251     //@CTK NO_OVERFLOW
252     //@CTK NO_BUF_OVERFLOW
253     //@CTK NO_ASF
254     /*@CTK approve
255         @tag assume_completion
256         @post (__post._allowed[msg.sender][spender]) == (value)
257     */
258     function approve(address spender, uint256 value) public returns (bool) {
259         _approve(msg.sender, spender, value);
260         return true;
261     }
262
263     /**
264     * @dev Transfer tokens from one address to another.
265     * Note that while this function emits an Approval event, this is not required as
266     * per the specification,
267     * and other compliant implementations may not emit the event.
268     * @param from address The address which you want to send tokens from
269     * @param to address The address which you want to transfer to
270     * @param value uint256 the amount of tokens to be transferred
271     */
272     //@CTK NO_OVERFLOW
273     //@CTK NO_BUF_OVERFLOW
274     //@CTK FAIL NO_ASF
275     /*@CTK "transferFrom correctness"
276         @tag assume_completion
277         @post value <= _balances[from] && value <= _allowed[from][msg.sender]

```



```

273     @post to != from -> __post._balances[from] == _balances[from] - value
274     @post to != from -> __post._balances[to] == _balances[to] + value
275     @post to == from -> __post._balances[from] == _balances[from]
276     @post __post._allowed[from][msg.sender] == _allowed[from][msg.sender] - value
277     */
278     function transferFrom(address from, address to, uint256 value) public returns (
279         bool) {
280         _transfer(from, to, value);
281         _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
282         return true;
283     }
284     /**
285     * @dev Increase the amount of tokens that an owner allowed to a spender.
286     * approve should be called when _allowed[msg.sender][spender] == 0. To increment
287     * allowed value is better to use this function to avoid 2 calls (and wait until
288     * the first transaction is mined)
289     * From MonolithDAO Token.sol
290     * Emits an Approval event.
291     * @param spender The address which will spend the funds.
292     * @param addedValue The amount of tokens to increase the allowance by.
293     */
294     //@CTK NO_OVERFLOW
295     //@CTK NO_BUF_OVERFLOW
296     //@CTK FAIL NO_ASF
297     /*@CTK "increaseApproval correctness"
298     @tag assume_completion
299     @post __post._allowed[msg.sender][spender] == _allowed[msg.sender][spender] +
        addedValue
300     */
301     function increaseAllowance(address spender, uint256 addedValue) public returns (
302         bool) {
303         _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
304         return true;
305     }
306     /**
307     * @dev Decrease the amount of tokens that an owner allowed to a spender.
308     * approve should be called when _allowed[msg.sender][spender] == 0. To decrement
309     * allowed value is better to use this function to avoid 2 calls (and wait until
310     * the first transaction is mined)
311     * From MonolithDAO Token.sol
312     * Emits an Approval event.
313     * @param spender The address which will spend the funds.
314     * @param subtractedValue The amount of tokens to decrease the allowance by.
315     */
316     //@CTK NO_OVERFLOW
317     //@CTK NO_BUF_OVERFLOW
318     //@CTK FAIL NO_ASF
319     /*@CTK decreaseApproval0
320     @pre __return == true
321     @pre _allowed[msg.sender][spender] <= subtractedValue
322     @post __post._allowed[msg.sender][spender] == 0
323     */
324     /*@CTK decreaseApproval
325     @pre __return == true
326     @pre _allowed[msg.sender][spender] > subtractedValue
327     @post __post._allowed[msg.sender][spender] ==

```

```

328     _allowed[msg.sender][spender] - subtractedValue
329     */
330     function decreaseAllowance(address spender, uint256 subtractedValue) public
331         returns (bool) {
332         _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue
333             ));
334         return true;
335     }
336
337     /**
338     * @dev Transfer token for a specified addresses
339     * @param from The address to transfer from.
340     * @param to The address to transfer to.
341     * @param value The amount to be transferred.
342     */
343     // @CTK NO_OVERFLOW
344     // @CTK NO_BUF_OVERFLOW
345     // @CTK FAIL NO_ASF
346     /* @CTK _transfer
347     @tag assume_completion
348     @pre from != to
349     @post value <= _balances[from]
350     @post __post._balances[from] == _balances[from] - value
351     @post __post._balances[to] == _balances[to] + value
352     */
353     function _transfer(address from, address to, uint256 value) internal {
354
355         _balances[from] = _balances[from].sub(value);
356         _balances[to] = _balances[to].add(value);
357         emit Transfer(from, to, value);
358     }
359
360     /**
361     * @dev Internal function that mints an amount of the token and assigns it to
362     * an account. This encapsulates the modification of balances such that the
363     * proper events are emitted.
364     * @param account The account that will receive the created tokens.
365     * @param value The amount that will be created.
366     */
367     // @CTK NO_OVERFLOW
368     // @CTK NO_BUF_OVERFLOW
369     // @CTK FAIL NO_ASF
370     /* @CTK mint
371     @tag assume_completion
372     @post account != address(0)
373     @post (__post._totalSupply) == (_totalSupply + value)
374     @post (__post._balances[account]) == (_balances[account] + value)
375     */
376     function _mint(address account, uint256 value) internal {
377         require(account != address(0));
378
379         _totalSupply = _totalSupply.add(value);
380         _balances[account] = _balances[account].add(value);
381         emit Transfer(address(0), account, value);
382     }
383
384     /**
385     * @dev Internal function that burns an amount of the token of a given

```



```

384     * account.
385     * @param account The account whose tokens will be burnt.
386     * @param value The amount that will be burnt.
387     */
388
389     //@CTK NO_OVERFLOW
390     //@CTK NO_BUF_OVERFLOW
391     //@CTK FAIL NO_ASF
392     /*@CTK burn
393         @tag assume_completion
394         @post account != address(0)
395         @post (value <= _balances[account])
396         @post (__post._totalSupply) == (_totalSupply - value)
397         @post (__post._balances[account]) == (_balances[account] - value)
398     */
399     function _burn(address account, uint256 value) internal {
400         require(account != address(0));
401
402         _totalSupply = _totalSupply.sub(value);
403         _balances[account] = _balances[account].sub(value);
404         emit Transfer(account, address(0), value);
405     }
406
407     /**
408     * @dev Approve an address to spend another addresses' tokens.
409     * @param owner The address that owns the tokens.
410     * @param spender The address that will spend the tokens.
411     * @param value The number of tokens that can be spent.
412     */
413
414     //@CTK NO_OVERFLOW
415     //@CTK NO_BUF_OVERFLOW
416     //@CTK NO_ASF
417     /*@CTK _approve
418         @tag assume_completion
419         @post spender != address(0)
420         @post owner != address(0)
421         @post __post._allowed[owner][spender] == value
422     */
423     function _approve(address owner, address spender, uint256 value) internal {
424         require(spender != address(0));
425         require(owner != address(0));
426
427         _allowed[owner][spender] = value;
428         emit Approval(owner, spender, value);
429     }
430
431     /**
432     * @dev Internal function that burns an amount of the token of a given
433     * account, deducting from the sender's allowance for said account. Uses the
434     * internal burn function.
435     * Emits an Approval event (reflecting the reduced allowance).
436     * @param account The account whose tokens will be burnt.
437     * @param value The amount that will be burnt.
438     */
439     //@CTK NO_OVERFLOW
440     //@CTK NO_BUF_OVERFLOW
441     //@CTK FAIL NO_ASF

```



```

442  /*@CTK burnFrom
443      @tag assume_completion
444      @post (value <= _allowed[account][msg.sender])
445      @post (value <= _balances[account])
446      @post (__post._allowed[account][msg.sender]) == (_allowed[account][msg.sender] -
          value)
447      @post (__post._balances[account]) == (_balances[account] - (value))
448      @post __post._totalSupply == (_totalSupply - value)
449  */
450  function _burnFrom(address account, uint256 value) internal {
451      _burn(account, value);
452      _approve(account, msg.sender, _allowed[account][msg.sender].sub(value));
453  }
454 }
455
456 /**
457  * @title ERC20Detailed token
458  * @dev The decimals are only for visualization purposes.
459  * All the operations are done using the smallest and indivisible token unit,
460  * just as on Ethereum all the operations are done in wei.
461  */
462 contract ERC20Detailed is IERC20 {
463     string private _name;
464     string private _symbol;
465     uint8 private _decimals;
466
467     constructor (string memory name, string memory symbol, uint8 decimals) public {
468         _name = name;
469         _symbol = symbol;
470         _decimals = decimals;
471     }
472
473     /**
474      * @return the name of the token.
475      */
476     function name() public view returns (string memory) {
477         return _name;
478     }
479
480     /**
481      * @return the symbol of the token.
482      */
483     function symbol() public view returns (string memory) {
484         return _symbol;
485     }
486
487     /**
488      * @return the number of decimals of the token.
489      */
490     function decimals() public view returns (uint8) {
491         return _decimals;
492     }
493 }
494
495 /**
496  * @title Ownable
497  * @dev The Ownable contract has an owner address, and provides basic authorization
      control

```

```

498 * functions, this simplifies the implementation of "user permissions".
499 */
500 contract Ownable {
501     address private _owner;
502
503     event OwnershipTransferred(address indexed previousOwner, address indexed newOwner
504         );
505
506     /**
507      * @dev The Ownable constructor sets the original `owner` of the contract to the
508      * sender
509      * account.
510      */
511     /*@CTK Ownable
512      @tag assume_completion
513      @post __post._owner == msg.sender
514      */
515     constructor () internal {
516         _owner = msg.sender;
517         emit OwnershipTransferred(address(0), _owner);
518     }
519
520     /**
521      * @return the address of the owner.
522      */
523     function owner() public view returns (address) {
524         return _owner;
525     }
526
527     /**
528      * @dev Throws if called by any account other than the owner.
529      */
530     modifier onlyOwner() {
531         require(isOwner());
532         _;
533     }
534
535     /**
536      * @return true if `msg.sender` is the owner of the contract.
537      */
538     /*@CTK isOwner
539      @tag assume_completion
540      @post __return == (msg.sender == _owner)
541      */
542     function isOwner() public view returns (bool) {
543         return msg.sender == _owner;
544     }
545
546     /**
547      * @dev Allows the current owner to relinquish control of the contract.
548      * It will not be possible to call the functions with the `onlyOwner`
549      * modifier anymore.
550      * @notice Renouncing ownership will leave the contract without an owner,
551      * thereby removing any functionality that is only available to the owner.
552      */
553     /*@CTK renounceOwnership
554      @tag assume_completion
555      @post msg.sender == _owner

```

page 58



```
611      * @dev Withdraw the ERC20 Token in the VANTAToken contract.
612      * @param erc20 ERC20 Token address.
613      * @param to To receive tokens.
614      * @param amount Tokens amount.
615      */
616      function withdrawERC20Token(address erc20, address to, uint256 amount) external
        onlyOwner {
617          require(to != address(0x0));
618          require(ITransferable(erc20).transfer(to, amount));
619      }
620 }
```