

CERTIK AUDIT REPORT FOR VID.CAMERA



Request Date: 2019-06-07
Revision Date: 2019-06-10
Platform Name: Ethereum



Contents

Disclaimer	1
About CertiK	2
Exective 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	17

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 Vid.Camera(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 1.4B in assets.

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

Executive Summary

This report has been prepared as product of the Smart Contract Audit request by Vid.Camera. This audit was conducted to discover issues and vulnerabilities in the source code of Vid.Camera's Smart Contracts. Utilizing CertiK's Formal Verification Platform, Static Analysis and Manual Review, a comprehensive examination has been performed. The auditing process pays special attention to the following considerations.

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessment of the codebase for 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

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.

Testing Summary

PASS

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

Jun 10, 2019



Type of Issues

CertiK smart label engine applied 100% covered 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 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

Unused Solidity library (`Roles`) included in the source codes. Recommend removing unused parts.

Manual Review Notes

Review Details

Source Code SHA-256 Checksum

- **VidERC20.sol** c4d1c9d7083691aa6dca96994608cf50f3991e17442c6dddb733c2ad471435a1

Summary

CertiK was chosen by Vid to audit the design and implementation of its soon to be released Vid.Camera 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.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 5 in File VidERC20.sol

```
5 pragma solidity >=0.4.22 <0.6.0;
```



 Only these compiler versions are safe to compile your code: 0.5.9

Formal Verification Results

How to read

Detail for Request 1

transferFrom to same address


Verification date	 20, Oct 2018
Verification timespan	 395.38 ms

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

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

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

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

Counterexample	 This code violates the specification	
Initial environment	1	Counter Example:
	2	Before Execution:
	3	Input = {
	4	from = 0x0
	5	to = 0x0
	6	tokens = 0x6c
	7	}
	8	This = 0
	52	}
	53	balance: 0x0
	54	}
	55	}
Post environment	57	After Execution:
	58	Input = {
	59	from = 0x0
	60	to = 0x0
	61	tokens = 0x6c

Formal Verification Request 1

SafeMath mul

📅 10, Jun 2019

🕒 287.98 ms

Line 32-37 in File VidERC20.sol

```
32  /*@CTK "SafeMath mul"
33     @post (a > 0) && (((a * b) / a) != b) -> __reverted
34     @post __reverted -> (a > 0) && (((a * b) / a) != b)
35     @post !__reverted -> __return == a * b
36     @post !__reverted == !__has_overflow
37 */
```

Line 38-50 in File VidERC20.sol

```
38  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
39      // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
40      // benefit is lost if 'b' is also tested.
41      // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
42      if (a == 0) {
43          return 0;
44      }
45
46      uint256 c = a * b;
47      require(c / a == b);
48
49      return c;
50 }
```

✅ The code meets the specification.

Formal Verification Request 2

SafeMath div

📅 10, Jun 2019

🕒 11.82 ms

Line 55-59 in File VidERC20.sol

```
55  /*@CTK "SafeMath div"
56     @post b != 0 -> !__reverted
57     @post !__reverted -> __return == a / b
58     @post !__reverted -> !__has_overflow
59 */
```

Line 60-67 in File VidERC20.sol

```
60  function div(uint256 a, uint256 b) internal pure returns (uint256) {
61      // Solidity only automatically asserts when dividing by 0
62      require(b > 0);
63      uint256 c = a / b;
64      // assert(a == b * c + a % b); // There is no case in which this doesn't hold
65
66      return c;
67 }
```

✓ The code meets the specification.

Formal Verification Request 3

SafeMath sub

📅 10, Jun 2019

🕒 11.12 ms

Line 72-76 in File VidERC20.sol

```
72  /*@CTK "SafeMath sub"
73     @post (a < b) == __reverted
74     @post !__reverted -> __return == a - b
75     @post !__reverted -> !__has_overflow
76 */
```

Line 77-82 in File VidERC20.sol

```
77  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
78      require(b <= a);
79      uint256 c = a - b;
80
81      return c;
82  }
```

✓ The code meets the specification.

Formal Verification Request 4

SafeMath add

📅 10, Jun 2019

🕒 12.52 ms

Line 87-91 in File VidERC20.sol

```
87  /*@CTK "SafeMath add"
88     @post (a + b < a || a + b < b) == __reverted
89     @post !__reverted -> __return == a + b
90     @post !__reverted -> !__has_overflow
91 */
```


Line 92-97 in File VidERC20.sol


```
92  function add(uint256 a, uint256 b) internal pure returns (uint256) {
93      uint256 c = a + b;
94      require(c >= a);
95
96      return c;
97  }
```

✓ The code meets the specification.

Formal Verification Request 5

SafeMath mod

 10, Jun 2019

 12.3 ms

Line 103-108 in File VidERC20.sol

```
103  /*@CTK "SafeMath mod"
104      @post (b == 0) == __reverted
105      @post !__reverted -> b != 0
106      @post !__reverted -> __return == a % b
107      @post !__reverted -> !__has_overflow
108  */
```


Line 109-112 in File VidERC20.sol


```
109  function mod(uint256 a, uint256 b) internal pure returns (uint256) {
110      require(b != 0);
111      return a % b;
112  }
```

 The code meets the specification.

Formal Verification Request 6

Ownable

 10, Jun 2019

 5.75 ms

Line 159-161 in File VidERC20.sol

```
159  /*@CTK Ownable
160      @post __post._owner == msg.sender
161  */
```

Line 162-165 in File VidERC20.sol


```
162  constructor () internal {
163      _owner = msg.sender;
164      emit OwnershipTransferred(address(0), _owner);
165  }
```

 The code meets the specification.

Formal Verification Request 7

owner

 10, Jun 2019

 5.21 ms

Line 170-172 in File VidERC20.sol

```
170  /*@CTK owner
171      @post __return == _owner
172  */
```

Line 173-175 in File VidERC20.sol

```
173  function owner() public view returns (address) {
174      return _owner;
175  }
```

✓ The code meets the specification.

Formal Verification Request 8

isOwner

📅 10, Jun 2019

🕒 5.16 ms

Line 188-190 in File VidERC20.sol

```
188  /*@CTK isOwner
189      @post __return == (msg.sender == _owner)
190  */
```

Line 191-193 in File VidERC20.sol

```
191  function isOwner() public view returns (bool) {
192      return msg.sender == _owner;
193  }
```

✓ The code meets the specification.

Formal Verification Request 9

renounceOwnership

📅 10, Jun 2019

🕒 23.0 ms

Line 201-205 in File VidERC20.sol

```
201  /*@CTK renounceOwnership
202      @tag assume_completion
203      @post _owner == msg.sender
204      @post __post._owner == address(0)
205  */
```

Line 206-209 in File VidERC20.sol

```
206  function renounceOwnership() public onlyOwner {
207      emit OwnershipTransferred(_owner, address(0));
208      _owner = address(0);
209  }
```

✓ The code meets the specification.

Formal Verification Request 10

_transferOwnership

📅 10, Jun 2019

🕒 12.79 ms

Line 223-227 in File VidERC20.sol

```
223  /*@CTK _transferOwnership
224      @tag assume_completion
225      @post newOwner != address(0)
226      @post __post._owner == newOwner
227  */
```

Line 228-232 in File VidERC20.sol

```
228  function _transferOwnership(address newOwner) internal {
229      require(newOwner != address(0));
230      emit OwnershipTransferred(_owner, newOwner);
231      _owner = newOwner;
232  }
```

✅ The code meets the specification.

Formal Verification Request 11

VIDERC20

📅 10, Jun 2019

🕒 16.72 ms

Line 252-255 in File VidERC20.sol

```
252  /*@CTK VIDERC20
253      @post __post._totalSupply == initialSupply
254      @post __post._balances[msg.sender] == initialSupply
255  */
```

Line 256-259 in File VidERC20.sol

```
256  constructor () public {
257      _totalSupply = initialSupply;
258      _balances[msg.sender] = initialSupply;
259  }
```

✅ The code meets the specification.

Formal Verification Request 12

totalSupply

📅 10, Jun 2019

🕒 5.58 ms

Line 264-266 in File VidERC20.sol

```
264  /*@CTK totalSupply
265      @post __return == _totalSupply
266  */
```

Line 267-269 in File VidERC20.sol

```
267  function totalSupply() public view returns (uint256) {
268      return _totalSupply;
269  }
```

✓ The code meets the specification.

Formal Verification Request 13

balanceOf

📅 10, Jun 2019

🕒 5.84 ms

Line 276-278 in File VidERC20.sol

```
276  /*@CTK balanceOf
277      @post __return == _balances[owner]
278  */
```

Line 279-281 in File VidERC20.sol

```
279  function balanceOf(address owner) public view returns (uint256) {
280      return _balances[owner];
281  }
```

✓ The code meets the specification.

Formal Verification Request 14

allowance

📅 10, Jun 2019

🕒 5.86 ms

Line 289-291 in File VidERC20.sol

```
289  /*@CTK allowance
290      @post __return == _allowed[owner][spender]
291  */
```


Line 292-294 in File VidERC20.sol


```
292  function allowance(address owner, address spender) public view returns (uint256) {
293      return _allowed[owner][spender];
294  }
```

✓ The code meets the specification.

Formal Verification Request 15

`_transfer`

 10, Jun 2019

 186.89 ms

Line 301-307 in File VidERC20.sol

```
301  /*@CTK _transfer
302      @tag assume_completion
303      @pre to != msg.sender
304      @post to != address(0)
305      @post __post._balances[msg.sender] == _balances[msg.sender] - value
306      @post __post._balances[to] == _balances[to] + value
307  */
```


Line 308-311 in File VidERC20.sol


```
308  function transfer(address to, uint256 value) public returns (bool) {
309      _transfer(msg.sender, to, value);
310      return true;
311  }
```

 The code meets the specification.

Formal Verification Request 16

`approve`

 10, Jun 2019

 18.86 ms

Line 322-326 in File VidERC20.sol

```
322  /*@CTK approve
323      @tag assume_completion
324      @post spender != address(0)
325      @post __post._allowed[msg.sender][spender] == value
326  */
```

Line 327-333 in File VidERC20.sol


```
327  function approve(address spender, uint256 value) public returns (bool) {
328      require(spender != address(0));
329
330      _allowed[msg.sender][spender] = value;
331      emit Approval(msg.sender, spender, value);
332      return true;
333  }
```

 The code meets the specification.

Formal Verification Request 17

`transferFrom`

 10, Jun 2019

 251.94 ms

Line 343-350 in File VidERC20.sol

```
343  /*@CTK transferFrom
344      @tag assume_completion
345      @pre to != from
346      @post to != address(0)
347      @post __post._allowed[from][msg.sender] == _allowed[from][msg.sender] - value
348      @post __post._balances[from] == _balances[from] - value
349      @post __post._balances[to] == _balances[to] + value
350  */
```

Line 351-356 in File VidERC20.sol

```
351  function transferFrom(address from, address to, uint256 value) public returns (
352      bool) {
353      _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
354      _transfer(from, to, value);
355      emit Approval(from, msg.sender, _allowed[from][msg.sender]);
356      return true;
357  }
```

✓ The code meets the specification.

Formal Verification Request 18

_transfer

📅 10, Jun 2019

🕒 63.35 ms

Line 364-370 in File VidERC20.sol

```
364  /*@CTK _transfer
365      @tag assume_completion
366      @pre to != from
367      @post to != address(0)
368      @post __post._balances[from] == _balances[from] - value
369      @post __post._balances[to] == _balances[to] + value
370  */
```

Line 371-377 in File VidERC20.sol

```
371  function _transfer(address from, address to, uint256 value) internal {
372      require(to != address(0));
373
374      _balances[from] = _balances[from].sub(value);
375      _balances[to] = _balances[to].add(value);
376      emit Transfer(from, to, value);
377  }
```

✓ The code meets the specification.

Source Code with CertiK Labels

File VidERC20.sol

```

1  /**
2   *Submitted for verification at Etherscan.io on 2019-04-22
3   */
4
5  pragma solidity >=0.4.22 <0.6.0;
6  /**
7   * @title ERC20 interface
8   * @dev see https://github.com/ethereum/EIPs/issues/20
9   */
10 interface IERC20 {
11     function totalSupply() external view returns (uint256);
12
13     function balanceOf(address who) external view returns (uint256);
14
15     function allowance(address owner, address spender) external view returns (uint256)
16         ;
17
18     function transfer(address to, uint256 value) external returns (bool);
19
20     function approve(address spender, uint256 value) external returns (bool);
21
22     function transferFrom(address from, address to, uint256 value) external returns (
23         bool);
24
25     event Transfer(address indexed from, address indexed to, uint256 value);
26
27     event Approval(address indexed owner, address indexed spender, uint256 value);
28 }
29
30 library SafeMath {
31     /**
32     * @dev Multiplies two numbers, reverts on overflow.
33     */
34     /*@CTK "SafeMath mul"
35     @post (a > 0) && (((a * b) / a) != b) -> __reverted
36     @post __reverted -> (a > 0) && (((a * b) / a) != b)
37     @post !__reverted -> __return == a * b
38     @post !__reverted == !__has_overflow
39     */
40     function mul(uint256 a, uint256 b) internal pure returns (uint256) {
41         // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
42         // benefit is lost if 'b' is also tested.
43         // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
44         if (a == 0) {
45             return 0;
46         }
47
48         uint256 c = a * b;
49         require(c / a == b);
50
51         return c;
52     }
53 }
54
55 /**

```

```

53  * @dev Integer division of two numbers truncating the quotient, reverts on
    division by zero.
54  */
55  /*@CTK "SafeMath div"
56   @post b != 0 -> !__reverted
57   @post !__reverted -> __return == a / b
58   @post !__reverted -> !__has_overflow
59  */
60  function div(uint256 a, uint256 b) internal pure returns (uint256) {
61      // Solidity only automatically asserts when dividing by 0
62      require(b > 0);
63      uint256 c = a / b;
64      // assert(a == b * c + a % b); // There is no case in which this doesn't hold
65
66      return c;
67  }
68
69  /**
70   * @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater
    than minuend).
71  */
72  /*@CTK "SafeMath sub"
73   @post (a < b) == __reverted
74   @post !__reverted -> __return == a - b
75   @post !__reverted -> !__has_overflow
76  */
77  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
78      require(b <= a);
79      uint256 c = a - b;
80
81      return c;
82  }
83
84  /**
85   * @dev Adds two numbers, reverts on overflow.
86  */
87  /*@CTK "SafeMath add"
88   @post (a + b < a || a + b < b) == __reverted
89   @post !__reverted -> __return == a + b
90   @post !__reverted -> !__has_overflow
91  */
92  function add(uint256 a, uint256 b) internal pure returns (uint256) {
93      uint256 c = a + b;
94      require(c >= a);
95
96      return c;
97  }
98
99  /**
100   * @dev Divides two numbers and returns the remainder (unsigned integer modulo),
    * reverts when dividing by zero.
101  */
102  /*@CTK "SafeMath mod"
103   @post (b == 0) == __reverted
104   @post !__reverted -> b != 0
105   @post !__reverted -> __return == a % b
106   @post !__reverted -> !__has_overflow
107  */
108  */

```

```

109     function mod(uint256 a, uint256 b) internal pure returns (uint256) {
110         require(b != 0);
111         return a % b;
112     }
113 }
114
115 library Roles {
116     struct Role {
117         mapping (address => bool) bearer;
118     }
119
120     /**
121     * @dev give an account access to this role
122     */
123     function add(Role storage role, address account) internal {
124         require(account != address(0));
125         require(!has(role, account));
126
127         role.bearer[account] = true;
128     }
129
130     /**
131     * @dev remove an account's access to this role
132     */
133     function remove(Role storage role, address account) internal {
134         require(account != address(0));
135         require(has(role, account));
136
137         role.bearer[account] = false;
138     }
139
140     /**
141     * @dev check if an account has this role
142     * @return bool
143     */
144     function has(Role storage role, address account) internal view returns (bool) {
145         require(account != address(0));
146         return role.bearer[account];
147     }
148 }
149
150 contract Ownable {
151     address private _owner;
152
153     event OwnershipTransferred(address indexed previousOwner, address indexed newOwner
154         );
155
156     /**
157     * @dev The Ownable constructor sets the original 'owner' of the contract to the
158     * account.
159     */
160     /*@CTK Ownable
161     @post __post._owner == msg.sender
162     */
163     constructor () internal {
164         _owner = msg.sender;
165         emit OwnershipTransferred(address(0), _owner);

```

```

165     }
166
167     /**
168      * @return the address of the owner.
169      */
170     /**@CTK owner
171      @post __return == _owner
172      */
173     function owner() public view returns (address) {
174         return _owner;
175     }
176
177     /**
178      * @dev Throws if called by any account other than the owner.
179      */
180     modifier onlyOwner() {
181         require(isOwner());
182         _;
183     }
184
185     /**
186      * @return true if 'msg.sender' is the owner of the contract.
187      */
188     /**@CTK isOwner
189      @post __return == (msg.sender == _owner)
190      */
191     function isOwner() public view returns (bool) {
192         return msg.sender == _owner;
193     }
194
195     /**
196      * @dev Allows the current owner to relinquish control of the contract.
197      * @notice Renouncing to ownership will leave the contract without an owner.
198      * It will not be possible to call the functions with the 'onlyOwner'
199      * modifier anymore.
200      */
201     /**@CTK renounceOwnership
202      @tag assume_completion
203      @post _owner == msg.sender
204      @post __post._owner == address(0)
205      */
206     function renounceOwnership() public onlyOwner {
207         emit OwnershipTransferred(_owner, address(0));
208         _owner = address(0);
209     }
210
211     /**
212      * @dev Allows the current owner to transfer control of the contract to a newOwner
213      *
214      * @param newOwner The address to transfer ownership to.
215      */
216     function transferOwnership(address newOwner) public onlyOwner {
217         _transferOwnership(newOwner);
218     }
219
220     /**
221      * @dev Transfers control of the contract to a newOwner.
222      * @param newOwner The address to transfer ownership to.

```

```

222     */
223     /*@CTK _transferOwnership
224         @tag assume_completion
225         @post newOwner != address(0)
226         @post __post._owner == newOwner
227     */
228     function _transferOwnership(address newOwner) internal {
229         require(newOwner != address(0));
230         emit OwnershipTransferred(_owner, newOwner);
231         _owner = newOwner;
232     }
233 }
234
235 contract VIDERC20 is IERC20, Ownable {
236
237     using SafeMath for uint256;
238
239     mapping (address => uint256) private _balances;
240
241     mapping (address => mapping (address => uint256)) private _allowed;
242
243     uint256 public sellPrice;
244     uint256 public buyPrice;
245
246     // Public variables of the token
247     string public name = "VID";
248     string public symbol = "VID";
249     uint8 public decimals = 5;
250     uint256 private _totalSupply;
251     uint256 public constant initialSupply = 62500000000000;
252     /*@CTK VIDERC20
253         @post __post._totalSupply == initialSupply
254         @post __post._balances[msg.sender] == initialSupply
255     */
256     constructor () public {
257         _totalSupply = initialSupply;
258         _balances[msg.sender] = initialSupply;
259     }
260
261     /**
262     * @dev Total number of tokens in existence
263     */
264     /*@CTK totalSupply
265         @post __return == _totalSupply
266     */
267     function totalSupply() public view returns (uint256) {
268         return _totalSupply;
269     }
270
271     /**
272     * @dev Gets the balance of the specified address.
273     * @param owner The address to query the balance of.
274     * @return An uint256 representing the amount owned by the passed address.
275     */
276     /*@CTK balanceOf
277         @post __return == _balances[owner]
278     */
279     function balanceOf(address owner) public view returns (uint256) {

```

```

280     return _balances[owner];
281 }
282
283 /**
284  * @dev Function to check the amount of tokens that an owner allowed to a spender.
285  * @param owner address The address which owns the funds.
286  * @param spender address The address which will spend the funds.
287  * @return A uint256 specifying the amount of tokens still available for the
288         spender.
289  */
290 /**@CTK allowance
291  @post __return == _allowed[owner][spender]
292  */
293 function allowance(address owner, address spender) public view returns (uint256) {
294     return _allowed[owner][spender];
295 }
296
297 /**
298  * @dev Transfer token for a specified address
299  * @param to The address to transfer to.
300  * @param value The amount to be transferred.
301  */
302 /**@CTK _transfer
303  @tag assume_completion
304  @pre to != msg.sender
305  @post to != address(0)
306  @post __post._balances[msg.sender] == _balances[msg.sender] - value
307  @post __post._balances[to] == _balances[to] + value
308  */
309 function transfer(address to, uint256 value) public returns (bool) {
310     _transfer(msg.sender, to, value);
311     return true;
312 }
313
314 /**
315  * @dev Approve the passed address to spend the specified amount of tokens on
316         behalf of msg.sender.
317  * Beware that changing an allowance with this method brings the risk that someone
318         may use both the old
319  * and the new allowance by unfortunate transaction ordering. One possible
320         solution to mitigate this
321  * race condition is to first reduce the spender's allowance to 0 and set the
322         desired value afterwards:
323  * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
324  * @param spender The address which will spend the funds.
325  * @param value The amount of tokens to be spent.
326  */
327 /**@CTK approve
328  @tag assume_completion
329  @post spender != address(0)
330  @post __post._allowed[msg.sender][spender] == value
331  */
332 function approve(address spender, uint256 value) public returns (bool) {
333     require(spender != address(0));
334
335     _allowed[msg.sender][spender] = value;
336     emit Approval(msg.sender, spender, value);
337     return true;

```



```

333     }
334
335     /**
336      * @dev Transfer tokens from one address to another.
337      * Note that while this function emits an Approval event, this is not required as
338      * per the specification,
339      * and other compliant implementations may not emit the event.
340      * @param from address The address which you want to send tokens from
341      * @param to address The address which you want to transfer to
342      * @param value uint256 the amount of tokens to be transferred
343      */
344     /*@CTK transferFrom
345      @tag assume_completion
346      @pre to != from
347      @post to != address(0)
348      @post __post._allowed[from][msg.sender] == _allowed[from][msg.sender] - value
349      @post __post._balances[from] == _balances[from] - value
350      @post __post._balances[to] == _balances[to] + value
351      */
352     function transferFrom(address from, address to, uint256 value) public returns (
353         bool) {
354         _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
355         _transfer(from, to, value);
356         emit Approval(from, msg.sender, _allowed[from][msg.sender]);
357         return true;
358     }
359
360     /**
361      * @dev Transfer token for a specified addresses
362      * @param from The address to transfer from.
363      * @param to The address to transfer to.
364      * @param value The amount to be transferred.
365      */
366     /*@CTK _transfer
367      @tag assume_completion
368      @pre to != from
369      @post to != address(0)
370      @post __post._balances[from] == _balances[from] - value
371      @post __post._balances[to] == _balances[to] + value
372      */
373     function _transfer(address from, address to, uint256 value) internal {
374         require(to != address(0));
375
376         _balances[from] = _balances[from].sub(value);
377         _balances[to] = _balances[to].add(value);
378         emit Transfer(from, to, value);
379     }

```