CertiK-Tunwu Audit Report For Crazy Miner

Request Date: 2019-10-23
Revision Date: 2019-10-29
Platform Name: Ethereum

TUNWU

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-Tunwu and Crazy Miner(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-Tunwu'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 Crazy Miner to discover issues and vulnerabilities in the source code of their CAME smart contracts. A comprehensive examination has been performed, utilizing CertiK-Tunwu'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-Tunwu 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

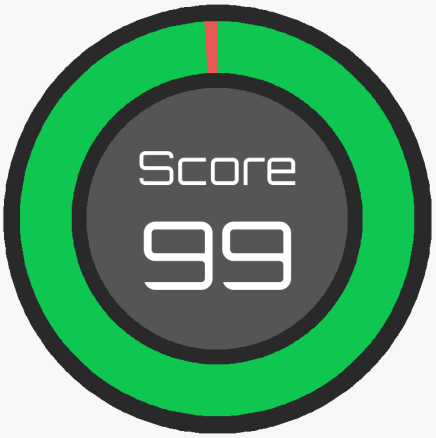
The code implementation does not follow best practices, or use suboptimal design patterns, which may lead to security vulnerabilies further down the line.

Testing Summary

PASS

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

Oct 26, 2019



Type of Issues

CertiK-Tunwu smart label engine applied 100% formal verification coverage on the source code. Our team of engineers ao 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

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 5 in File came.sol

```
5 pragma solidity ^0.4.24;
```

Version to compile has the following bug: 0.4.24: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData 0.4.25: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x 0.4.26: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2

Formal Verification Request 1

SafeMath mul

26, Oct 2019
168.56 ms

Line 19-24 in File came.sol

```
19  /*@CTK "SafeMath mul"
20    @post (a > 0) && (((a * b) / a) != b) -> __reverted
21    @post __reverted -> (a > 0) && (((a * b) / a) != b)
22    @post !__reverted -> __return == a * b
23    @post !__reverted == !__has_overflow
24   */
```

Line 25-29 in File came.sol

```
25  function safeMul(uint256 a, uint256 b) internal returns (uint256) {
26    uint256 c = a * b;
27    assert(a == 0 || c / a == b);
28    return c;
29  }
```

The code meets the specification.

Formal Verification Request 2

SafeMath div

26, Oct 2019
1163.73 ms

Line 31-35 in File came.sol

```
31  /*@CTK "SafeMath div"
32    @post b != 0 -> !__reverted
33    @post !__reverted -> __return == a / b
34    @post !__reverted -> !__has_overflow
35   */
```

Line 36-41 in File came.sol

```
36  function safeDiv(uint256 a, uint256 b) internal returns (uint256) {
37    assert(b > 0);
38    uint256 c = a / b;
39    assert(a == b * c + a % b);
40    return c;
41  }
```

The code meets the specification.

Formal Verification Request 3

SafeMath sub

26, Oct 2019
80.93 ms

Line 43-47 in File came.sol

```
43  /*@CTK "SafeMath sub"
44    @post (a < b) == __reverted
45    @post !__reverted -> __return == a - b
46    @post !__reverted -> !__has_overflow
47  */
```

Line 48-51 in File came.sol

```
48  function safeSub(uint256 a, uint256 b) internal returns (uint256) {
49    assert(b <= a);
50    return a - b;
51  }
```

✅ The code meets the specification.

Formal Verification Request 4

SafeMath add

📅 26, Oct 2019
⏱ 56.21 ms

Line 53-57 in File came.sol

```
53  /*@CTK "SafeMath add"
54    @post (a + b < a || a + b < b) == __reverted
55    @post !__reverted -> __return == a + b
56    @post !__reverted -> !__has_overflow
57   */
```

Line 58-62 in File came.sol

```
58  function safeAdd(uint256 a, uint256 b) internal returns (uint256) {
59    uint256 c = a + b;
60    assert(c>=a && c>=b);
61    return c;
62  }
```

✅ The code meets the specification.

Formal Verification Request 5

transfer

📅 26, Oct 2019
⏱ 659.0 ms

Line 67-76 in File came.sol

```
67    /*@CTK transfer
68      @tag assume_completion
69      @pre msg.sender != _to
70      @post _to != 0
71      @post _value > 0
```

```
72      @post balances[msg.sender] >= _value
73      @post balances[_to] + _value >= balances[_to]
74      @post __post.balances[msg.sender] == balances[msg.sender] - _value
75      @post __post.balances[_to] == balances[_to] + _value
76     */
```

Line 77-86 in File came.sol

```
77    function transfer(address _to, uint256 _value) returns (bool success){
78        if (_to == 0x0) assert(false); // revert('Address cannot be 0x0'); // Prevent
              transfer to 0x0 address. Use burn() instead
79        if (_value <= 0) assert(false); // revert('_value must be greater than 0');
80        if (balances[msg.sender] < _value) assert(false); // revert('Insufficient
              balance');// Check if the sender has enough
81        if (balances[_to] + _value < balances[_to]) assert(false); // revert('has
              overflows'); // Check for overflows
82        balances[msg.sender] = SafeMath.safeSub(balances[msg.sender], _value);
                      // Subtract from the sender
83        balances[_to] = SafeMath.safeAdd(balances[_to], _value);                    //
              Add the same to the recipient
84        emit Transfer(msg.sender, _to, _value);             // Notify anyone listening
               that this transfer took place
85        return true;
86    }
```

The code meets the specification.

Formal Verification Request 6

transferFrom

26, Oct 2019
629.13 ms

Line 89-100 in File came.sol

```
89    /*@CTK transferFrom
90      @tag assume_completion
91      @pre _from != _to
92      @post _to != 0
93      @post _value > 0
94      @post balances[_from] >= _value
95      @post balances[_to] + _value >= balances[_to]
96      @post _value <= allowed[_from][msg.sender]
97      @post __post.balances[_from] == balances[_from] - _value
98      @post __post.balances[_to] == balances[_to] + _value
99      @post __post.allowed[_from][msg.sender] == allowed[_from][msg.sender] - _value
100    */
```

Line 101-112 in File came.sol

```
101   function transferFrom(address _from, address _to, uint256 _value) returns (bool
          success) {
102       if (_to == 0x0) assert(false); // revert('Address cannot be 0x0'); // Prevent
              transfer to 0x0 address. Use burn() instead
103       if (_value <= 0) assert(false); // revert('_value must be greater than 0');
104       if (balances[_from] < _value) assert(false); // revert('Insufficient balance')
              ;// Check if the sender has enough
```

```
105         if (balances[_to] + _value < balances[_to]) assert(false); // revert('has
                overflows'); // Check for overflows
106         if (_value > allowed[_from][msg.sender]) assert(false); // revert('not allowed
                '); // Check allowed
107         balances[_from] = SafeMath.safeSub(balances[_from], _value);
                // Subtract from the sender
108         balances[_to] = SafeMath.safeAdd(balances[_to], _value);                    //
                 Add the same to the recipient
109         allowed[_from][msg.sender] = SafeMath.safeSub(allowed[_from][msg.sender],
                _value);
110         emit Transfer(_from, _to, _value);
111         return true;
112     }
```

The code meets the specification.

Formal Verification Request 7

balanceOf

26, Oct 2019
13.08 ms

Line 114-116 in File came.sol

```
114     /*@CTK balanceOf
115       @post __return == balances[_owner]
116      */
```

Line 117-119 in File came.sol

```
117     function balanceOf(address _owner) constant returns (uint256) {
118         return balances[_owner];
119     }
```

The code meets the specification.

Formal Verification Request 8

approve

26, Oct 2019
58.77 ms

Line 121-125 in File came.sol

```
121     /*@CTK approve
122       @tag assume_completion
123       @post _value > 0
124       @post __post.allowed[msg.sender][_spender] == _value
125      */
```

Line 126-131 in File came.sol

```
126     function approve(address _spender, uint256 _value) returns (bool) {
127         if (_value <= 0) assert(false); // revert('_value must be greater than 0');
128         allowed[msg.sender][_spender] = _value;
```

```
129         emit Approval(msg.sender, _spender, _value);
130         return true;
131     }
```

The code meets the specification.

Formal Verification Request 9

allowance

26, Oct 2019
17.19 ms

Line 133-135 in File came.sol

```
133     /*@CTK allowance
134       @post __return == allowed[_owner][_spender]
135      */
```

Line 136-138 in File came.sol

```
136     function allowance(address _owner, address _spender) constant returns (uint256) {
137         return allowed[_owner][_spender];
138     }
```

The code meets the specification.

Formal Verification Request 10

CAME

26, Oct 2019
19.98 ms

Line 152-154 in File came.sol

```
152     /*@CTK CAME
153       @post __post.balances[msg.sender] == totalSupply
154      */
```

Line 155-158 in File came.sol

```
155     constructor() public{
156         balances[msg.sender] = totalSupply;
157         emit Transfer(address(0), msg.sender, totalSupply);
158     }
```

The code meets the specification.

Source Code with CertiK-Tunwu Labels

File came.sol

```
1  /**
2   *Submitted for verification at Etherscan.io on 2019-08-07
3  */
4
5  pragma solidity ^0.4.24;
6  contract Token {
7      function totalSupply() constant returns (uint256 supply) {}
8      function balanceOf(address _owner) constant returns (uint256 balance) {}
9      function transfer(address _to, uint256 _value) returns (bool success) {}
10     function transferFrom(address _from, address _to, uint256 _value) returns (bool
           success) {}
11     function approve(address _spender, uint256 _value) returns (bool success) {}
12     function allowance(address _owner, address _spender) constant returns (uint256
           remaining) {}
13
14     event Transfer(address indexed _from, address indexed _to, uint256 _value);
15     event Approval(address indexed _owner, address indexed _spender, uint256 _value);
16 }
17
18 contract SafeMath {
19   /*@CTK "SafeMath mul"
20     @post (a > 0) && (((a * b) / a) != b) -> __reverted
21     @post __reverted -> (a > 0) && (((a * b) / a) != b)
22     @post !__reverted -> __return == a * b
23     @post !__reverted == !__has_overflow
24    */
25   function safeMul(uint256 a, uint256 b) internal returns (uint256) {
26     uint256 c = a * b;
27     assert(a == 0 || c / a == b);
28     return c;
29   }
30
31   /*@CTK "SafeMath div"
32     @post b != 0 -> !__reverted
33     @post !__reverted -> __return == a / b
34     @post !__reverted -> !__has_overflow
35    */
36   function safeDiv(uint256 a, uint256 b) internal returns (uint256) {
37     assert(b > 0);
38     uint256 c = a / b;
39     assert(a == b * c + a % b);
40     return c;
41   }
42
43   /*@CTK "SafeMath sub"
44     @post (a < b) == __reverted
45     @post !__reverted -> __return == a - b
46     @post !__reverted -> !__has_overflow
47   */
48   function safeSub(uint256 a, uint256 b) internal returns (uint256) {
49     assert(b <= a);
50     return a - b;
51   }
52
```

```
53    /*@CTK "SafeMath add"
54      @post (a + b < a || a + b < b) == __reverted
55      @post !__reverted -> __return == a + b
56      @post !__reverted -> !__has_overflow
57     */
58    function safeAdd(uint256 a, uint256 b) internal returns (uint256) {
59      uint256 c = a + b;
60      assert(c>=a && c>=b);
61      return c;
62    }
63
64  }
65
66  contract RegularToken is Token,SafeMath {
67      /*@CTK transfer
68        @tag assume_completion
69        @pre msg.sender != _to
70        @post _to != 0
71        @post _value > 0
72        @post balances[msg.sender] >= _value
73        @post balances[_to] + _value >= balances[_to]
74        @post __post.balances[msg.sender] == balances[msg.sender] - _value
75        @post __post.balances[_to] == balances[_to] + _value
76       */
77      function transfer(address _to, uint256 _value) returns (bool success){
78          if (_to == 0x0) assert(false); // revert('Address cannot be 0x0'); // Prevent
                transfer to 0x0 address. Use burn() instead
79          if (_value <= 0) assert(false); // revert('_value must be greater than 0');
80          if (balances[msg.sender] < _value) assert(false); // revert('Insufficient
                balance');// Check if the sender has enough
81          if (balances[_to] + _value < balances[_to]) assert(false); // revert('has
                overflows'); // Check for overflows
82          balances[msg.sender] = SafeMath.safeSub(balances[msg.sender], _value);
                            // Subtract from the sender
83          balances[_to] = SafeMath.safeAdd(balances[_to], _value);                    //
                Add the same to the recipient
84          emit Transfer(msg.sender, _to, _value);             // Notify anyone listening
                 that this transfer took place
85          return true;
86      }
87
88      /* A contract attempts to get the coins */
89      /*@CTK transferFrom
90        @tag assume_completion
91        @pre _from != _to
92        @post _to != 0
93        @post _value > 0
94        @post balances[_from] >= _value
95        @post balances[_to] + _value >= balances[_to]
96        @post _value <= allowed[_from][msg.sender]
97        @post __post.balances[_from] == balances[_from] - _value
98        @post __post.balances[_to] == balances[_to] + _value
99        @post __post.allowed[_from][msg.sender] == allowed[_from][msg.sender] - _value
100      */
101     function transferFrom(address _from, address _to, uint256 _value) returns (bool
            success) {
102         if (_to == 0x0) assert(false); // revert('Address cannot be 0x0'); // Prevent
                transfer to 0x0 address. Use burn() instead
```

```
103         if (_value <= 0) assert(false); // revert('_value must be greater than 0');
104         if (balances[_from] < _value) assert(false); // revert('Insufficient balance')
                ;// Check if the sender has enough
105         if (balances[_to] + _value < balances[_to]) assert(false); // revert('has
                overflows'); // Check for overflows
106         if (_value > allowed[_from][msg.sender]) assert(false); // revert('not allowed
                '); // Check allowed
107         balances[_from] = SafeMath.safeSub(balances[_from], _value);
                // Subtract from the sender
108         balances[_to] = SafeMath.safeAdd(balances[_to], _value);                    //
                 Add the same to the recipient
109         allowed[_from][msg.sender] = SafeMath.safeSub(allowed[_from][msg.sender],
                _value);
110         emit Transfer(_from, _to, _value);
111         return true;
112     }
113
114     /*@CTK balanceOf
115       @post __return == balances[_owner]
116      */
117     function balanceOf(address _owner) constant returns (uint256) {
118         return balances[_owner];
119     }
120
121     /*@CTK approve
122       @tag assume_completion
123       @post _value > 0
124       @post __post.allowed[msg.sender][_spender] == _value
125      */
126     function approve(address _spender, uint256 _value) returns (bool) {
127         if (_value <= 0) assert(false); // revert('_value must be greater than 0');
128         allowed[msg.sender][_spender] = _value;
129         emit Approval(msg.sender, _spender, _value);
130         return true;
131     }
132
133     /*@CTK allowance
134       @post __return == allowed[_owner][_spender]
135      */
136     function allowance(address _owner, address _spender) constant returns (uint256) {
137         return allowed[_owner][_spender];
138     }
139
140     mapping (address => uint256) balances;
141     mapping (address => mapping (address => uint256)) allowed;
142 }
143
144
145 contract CAME is RegularToken {
146
147     uint256 public totalSupply = 10*10**26;
148     uint256 constant public decimals = 18;
149     string constant public name = "CAME";
150     string constant public symbol = "CAME";
151
152     /*@CTK CAME
153       @post __post.balances[msg.sender] == totalSupply
154      */
```

```
155     constructor() public{
156         balances[msg.sender] = totalSupply;
157         emit Transfer(address(0), msg.sender, totalSupply);
158     }
159 }
```