CertiK-TunWu Audit Report For CCG

Request Date: 2019-09-02
Revision Date: 2019-09-06
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 CCG(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 as the product of the Smart Contract Audit request by CCG. This audit was conducted to discover issues and vulnerabilities in the source code of CCG's Smart Contracts. Utilizing CertiK-TunWu'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 issue found, CertiK-TunWu 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 conditions, or it could affect the security standard by lost of access control.

Low

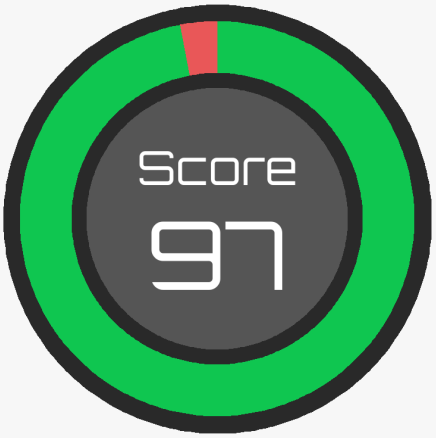
The code implementation is not a best practice, or use a suboptimal design pattern, which may lead to security vulnerabilies, but no concern found yet.

Testing Summary

PASS

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

Sep 06, 2019



Type of Issues

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

Title	Description	Issues	SWC ID
Integer Overflow 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.	1	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

- **CCG.sol**
  `977df98900f8d2323d9ed82b88c7182ae4941328a0b5447f517f975ff7e5222d`

Summary

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

CCG.sol

- Recommend using `safeMath` library to avoid integer overflow overall.
- `transferFrom(address _from, address _to, uint256 _value)`: Recommend adding `require (_from != _to)` to avoid decreasing the allowance while the balance stays the same.
- `transfer(address _to, uint256 _value)`: Recommend adding `require(msg.sender != _to)` to saving gas when `msg.sender` is the same as `_to`.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File CCG.sol

```
1 pragma solidity ^0.4.17;
```

Version to compile has the following bug: 0.4.17: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ExpExponentCleanup, EventStructWrongData, NestedArrayFunctionCallDecoder, ZeroFunctionSelector 0.4.18: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ExpExponentCleanup, EventStructWrongData, NestedArrayFunctionCallDecoder 0.4.19: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData, NestedArrayFunctionCallDecoder 0.4.20: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData, NestedArrayFunctionCallDecoder 0.4.21: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData, NestedArrayFunctionCallDecoder 0.4.22: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData, OneOfTwoConstructorsSkipped 0.4.23: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData 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

CCG constructor

06, Sep 2019
36.86 ms

Line 21-28 in File CCG.sol

```
21  /*@CTK "CCG constructor"
22    @tag assume_completion
23    @post __post.totalSupply == _initialAmount * 10 ** uint256(_decimalUnits)
24    @post __post.balances[msg.sender] == __post.totalSupply
25    @post __post.name == _tokenName
26    @post __post.decimals == _decimalUnits
27    @post __post.symbol == _tokenSymbol
28  */
```

Line 29-36 in File CCG.sol

```
29  function CCG (uint256 _initialAmount, string _tokenName, uint8 _decimalUnits,
        string _tokenSymbol) public {
30      totalSupply = _initialAmount * 10 ** uint256(_decimalUnits);
31      balances[msg.sender] = totalSupply;
32
33      name = _tokenName;
34      decimals = _decimalUnits;
35      symbol = _tokenSymbol;
36  }
```

The code meets the specification.

Formal Verification Request 2

If method completes, integer overflow would not happen.

06, Sep 2019
78.06 ms

Line 37 in File CCG.sol

```
37  //@CTK NO_OVERFLOW
```

Line 50-59 in File CCG.sol

```
50  function transfer(address _to, uint256 _value) public returns (bool success) {
51
52
53      require(balances[msg.sender] >= _value && balances[_to] + _value > balances[_to
            ]);
54      require(_to != 0x0);
55      balances[msg.sender] -= _value;
56      balances[_to] += _value;
57      Transfer(msg.sender, _to, _value);
58      return true;
59  }
```

The code meets the specification.

Formal Verification Request 3

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

06, Sep 2019
0.78 ms

Line 38 in File CCG.sol

```
38    //@CTK NO_BUF_OVERFLOW
```

Line 50-59 in File CCG.sol

```
50    function transfer(address _to, uint256 _value) public returns (bool success) {
51
52
53        require(balances[msg.sender] >= _value && balances[_to] + _value > balances[_to
              ]);
54        require(_to != 0x0);
55        balances[msg.sender] -= _value;
56        balances[_to] += _value;
57        Transfer(msg.sender, _to, _value);
58        return true;
59    }
```

The code meets the specification.

Formal Verification Request 4

Method will not encounter an assertion failure.

06, Sep 2019
0.68 ms

Line 39 in File CCG.sol

```
39    //@CTK NO_ASF
```

Line 50-59 in File CCG.sol

```
50    function transfer(address _to, uint256 _value) public returns (bool success) {
51
52
53        require(balances[msg.sender] >= _value && balances[_to] + _value > balances[_to
              ]);
54        require(_to != 0x0);
55        balances[msg.sender] -= _value;
56        balances[_to] += _value;
57        Transfer(msg.sender, _to, _value);
58        return true;
59    }
```

The code meets the specification.

Formal Verification Request 5

transfer

06, Sep 2019
145.64 ms

Line 40-49 in File CCG.sol

```
40    /*@CTK transfer
41      @tag assume_completion
42      @post balances[msg.sender] >= _value /\ balances[_to] + _value > balances[_to]
43      @post _to != 0x0
44      @post msg.sender != _to -> __post.balances[_to] == balances[_to] + _value
45      @post msg.sender != _to -> __post.balances[msg.sender] == balances[msg.sender] -
            _value
46      @post msg.sender == _to -> __post.balances[_to] == balances[_to]
47      @post msg.sender == _to -> __post.balances[msg.sender] == balances[msg.sender]
48      @post success
49    */
```

Line 50-59 in File CCG.sol

```
50    function transfer(address _to, uint256 _value) public returns (bool success) {
51
52
53        require(balances[msg.sender] >= _value && balances[_to] + _value > balances[_to
              ]);
54        require(_to != 0x0);
55        balances[msg.sender] -= _value;
56        balances[_to] += _value;
57        Transfer(msg.sender, _to, _value);
58        return true;
59    }
```

The code meets the specification.

Formal Verification Request 6

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

06, Sep 2019
39.03 ms

Line 61 in File CCG.sol

```
61    //@CTK NO_BUF_OVERFLOW
```

Line 72-81 in File CCG.sol

```
72    function transferFrom(address _from, address _to, uint256 _value) public returns
73    (bool success) {
74        require(balances[_from] >= _value && allowed[_from][msg.sender] >= _value);
75        require(balances[_to] + _value >= balances[_to]);
76        balances[_to] += _value;
77        balances[_from] -= _value;
78        allowed[_from][msg.sender] -= _value;
79        Transfer(_from, _to, _value);
80        return true;
```

```
81    }
```

The code meets the specification.

Formal Verification Request 7

Method will not encounter an assertion failure.

06, Sep 2019
0.8 ms

Line 62 in File CCG.sol

```
62    //@CTK NO_ASF
```

Line 72-81 in File CCG.sol

```
72    function transferFrom(address _from, address _to, uint256 _value) public returns
73    (bool success) {
74        require(balances[_from] >= _value && allowed[_from][msg.sender] >= _value);
75        require(balances[_to] + _value >= balances[_to]);
76        balances[_to] += _value;
77        balances[_from] -= _value;
78        allowed[_from][msg.sender] -= _value;
79        Transfer(_from, _to, _value);
80        return true;
81    }
```

The code meets the specification.

Formal Verification Request 8

If method completes, integer overflow would not happen.

06, Sep 2019
97.67 ms

Line 63 in File CCG.sol

```
63    //@CTK NO_OVERFLOW
```

Line 72-81 in File CCG.sol

```
72    function transferFrom(address _from, address _to, uint256 _value) public returns
73    (bool success) {
74        require(balances[_from] >= _value && allowed[_from][msg.sender] >= _value);
75        require(balances[_to] + _value >= balances[_to]);
76        balances[_to] += _value;
77        balances[_from] -= _value;
78        allowed[_from][msg.sender] -= _value;
79        Transfer(_from, _to, _value);
80        return true;
81    }
```

The code meets the specification.

Formal Verification Request 9

transferFrom

📅 06, Sep 2019
⏱ 395.98 ms

Line 64-71 in File CCG.sol

```
64  /*@CTK transferFrom
65    @tag assume_completion
66    @post balances[_from] >= _value /\ allowed[_from][msg.sender] >= _value
67    @post _to != _from -> __post.balances[_from] == balances[_from] - _value
68    @post _to != _from -> __post.balances[_to] == balances[_to] + _value
69    @post _to == _from -> __post.balances[_from] == balances[_from]
70    @post __post.allowed[_from][msg.sender] == allowed[_from][msg.sender] - _value
71  */
```

Line 72-81 in File CCG.sol

```
72  function transferFrom(address _from, address _to, uint256 _value) public returns
73  (bool success) {
74      require(balances[_from] >= _value && allowed[_from][msg.sender] >= _value);
75      require(balances[_to] + _value >= balances[_to]);
76      balances[_to] += _value;
77      balances[_from] -= _value;
78      allowed[_from][msg.sender] -= _value;
79      Transfer(_from, _to, _value);
80      return true;
81  }
```

✅ The code meets the specification.

Formal Verification Request 10

If method completes, integer overflow would not happen.

📅 06, Sep 2019
⏱ 6.03 ms

Line 82 in File CCG.sol

```
82  //@CTK NO_OVERFLOW
```

Line 90-92 in File CCG.sol

```
90  function balanceOf(address _owner) public constant returns (uint256 balance) {
91      return balances[_owner];
92  }
```

✅ The code meets the specification.

Formal Verification Request 11

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

📅 06, Sep 2019
⏱ 0.65 ms

Line 83 in File CCG.sol

```
83    //@CTK NO_BUF_OVERFLOW
```

Line 90-92 in File CCG.sol

```
90    function balanceOf(address _owner) public constant returns (uint256 balance) {
91        return balances[_owner];
92    }
```

The code meets the specification.

Formal Verification Request 12

Method will not encounter an assertion failure.

06, Sep 2019
0.4 ms

Line 84 in File CCG.sol

```
84    //@CTK NO_ASF
```

Line 90-92 in File CCG.sol

```
90    function balanceOf(address _owner) public constant returns (uint256 balance) {
91        return balances[_owner];
92    }
```

The code meets the specification.

Formal Verification Request 13

balanceOf

06, Sep 2019
0.47 ms

Line 85-89 in File CCG.sol

```
85    /*@CTK balanceOf
86      @post __reverted == false
87      @post balance == balances[_owner]
88      @post this == __post
89    */
```

Line 90-92 in File CCG.sol

```
90    function balanceOf(address _owner) public constant returns (uint256 balance) {
91        return balances[_owner];
92    }
```

The code meets the specification.

Formal Verification Request 14

If method completes, integer overflow would not happen.

06, Sep 2019
20.06 ms

Line 94 in File CCG.sol

```
94    //@CTK NO_OVERFLOW
```

Line 102-108 in File CCG.sol

```
102   function approve(address _spender, uint256 _value) public returns (bool success)
103   {
104       require((_value == 0) || (allowed[msg.sender][_spender] == 0));
105       allowed[msg.sender][_spender] = _value;
106       Approval(msg.sender, _spender, _value);
107       return true;
108   }
```

The code meets the specification.

Formal Verification Request 15

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

06, Sep 2019
0.52 ms

Line 95 in File CCG.sol

```
95    //@CTK NO_BUF_OVERFLOW
```

Line 102-108 in File CCG.sol

```
102   function approve(address _spender, uint256 _value) public returns (bool success)
103   {
104       require((_value == 0) || (allowed[msg.sender][_spender] == 0));
105       allowed[msg.sender][_spender] = _value;
106       Approval(msg.sender, _spender, _value);
107       return true;
108   }
```

The code meets the specification.

Formal Verification Request 16

Method will not encounter an assertion failure.

06, Sep 2019
0.57 ms

Line 96 in File CCG.sol

```
96    //@CTK NO_ASF
```

Line 102-108 in File CCG.sol

```
102     function approve(address _spender, uint256 _value) public returns (bool success)
103     {
104         require((_value == 0) || (allowed[msg.sender][_spender] == 0));
105         allowed[msg.sender][_spender] = _value;
106         Approval(msg.sender, _spender, _value);
107         return true;
108     }
```

The code meets the specification.

Formal Verification Request 17

approve

06, Sep 2019
2.04 ms

Line 97-101 in File CCG.sol

```
97      /*@CTK approve
98        @tag assume_completion
99        @post _value == 0 \/ allowed[msg.sender][_spender] == 0
100       @post __post.allowed[msg.sender][_spender] == _value
101     */
```

Line 102-108 in File CCG.sol

```
102     function approve(address _spender, uint256 _value) public returns (bool success)
103     {
104         require((_value == 0) || (allowed[msg.sender][_spender] == 0));
105         allowed[msg.sender][_spender] = _value;
106         Approval(msg.sender, _spender, _value);
107         return true;
108     }
```

The code meets the specification.

Formal Verification Request 18

If method completes, integer overflow would not happen.

06, Sep 2019
6.96 ms

Line 110 in File CCG.sol

```
110     //@CTK NO_OVERFLOW
```

Line 118-120 in File CCG.sol

```
118     function allowance(address _owner, address _spender) public constant returns (
            uint256 remaining) {
119         return allowed[_owner][_spender];
120     }
```

The code meets the specification.

Formal Verification Request 19

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

06, Sep 2019
0.45 ms

Line 111 in File CCG.sol

```
111     //@CTK NO_BUF_OVERFLOW
```

Line 118-120 in File CCG.sol

```
118     function allowance(address _owner, address _spender) public constant returns (
            uint256 remaining) {
119         return allowed[_owner][_spender];
120     }
```

The code meets the specification.

Formal Verification Request 20

Method will not encounter an assertion failure.

06, Sep 2019
0.46 ms

Line 112 in File CCG.sol

```
112     //@CTK NO_ASF
```

Line 118-120 in File CCG.sol

```
118     function allowance(address _owner, address _spender) public constant returns (
            uint256 remaining) {
119         return allowed[_owner][_spender];
120     }
```

The code meets the specification.

Formal Verification Request 21

allowance

06, Sep 2019
0.48 ms

Line 113-117 in File CCG.sol

```
113     /*@CTK allowance
114       @post __reverted == false
115       @post remaining == allowed[_owner][_spender]
116       @post this == __post
117     */
```

Line 118-120 in File CCG.sol

```
118     function allowance(address _owner, address _spender) public constant returns (
            uint256 remaining) {
119         return allowed[_owner][_spender];
120     }
```

The code meets the specification.

Source Code with CertiK-TunWu Labels

File CCG.sol

```
pragma solidity ^0.4.17;
contract Token{
    uint256 public totalSupply;

    function balanceOf(address _owner) public constant returns (uint256 balance);
    function transfer(address _to, uint256 _value) public returns (bool success);
    function transferFrom(address _from, address _to, uint256 _value) public returns
    (bool success);
    function approve(address _spender, uint256 _value) public returns (bool success);
    function allowance(address _owner, address _spender) public constant returns
    (uint256 remaining);
    event Transfer(address indexed _from, address indexed _to, uint256 _value);
    event Approval(address indexed _owner, address indexed _spender, uint256
    _value);
}
contract CCG is Token {

    string public name;
    uint8 public decimals;
    string public symbol;
    /*@CTK "CCG constructor"
      @tag assume_completion
      @post __post.totalSupply == _initialAmount * 10 ** uint256(_decimalUnits)
      @post __post.balances[msg.sender] == __post.totalSupply
      @post __post.name == _tokenName
      @post __post.decimals == _decimalUnits
      @post __post.symbol == _tokenSymbol
    */
    function CCG (uint256 _initialAmount, string _tokenName, uint8 _decimalUnits,
        string _tokenSymbol) public {
        totalSupply = _initialAmount * 10 ** uint256(_decimalUnits);
        balances[msg.sender] = totalSupply;

        name = _tokenName;
        decimals = _decimalUnits;
        symbol = _tokenSymbol;
    }
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK transfer
      @tag assume_completion
      @post balances[msg.sender] >= _value /\ balances[_to] + _value > balances[_to]
      @post _to != 0x0
      @post msg.sender != _to -> __post.balances[_to] == balances[_to] + _value
      @post msg.sender != _to -> __post.balances[msg.sender] == balances[msg.sender] -
          _value
      @post msg.sender == _to -> __post.balances[_to] == balances[_to]
      @post msg.sender == _to -> __post.balances[msg.sender] == balances[msg.sender]
      @post success
    */
    function transfer(address _to, uint256 _value) public returns (bool success) {


```

```
53          require(balances[msg.sender] >= _value && balances[_to] + _value > balances[_to
                ]);
54          require(_to != 0x0);
55          balances[msg.sender] -= _value;
56          balances[_to] += _value;
57          Transfer(msg.sender, _to, _value);
58          return true;
59      }
60
61      //@CTK NO_BUF_OVERFLOW
62      //@CTK NO_ASF
63      //@CTK NO_OVERFLOW
64      /*@CTK transferFrom
65        @tag assume_completion
66        @post balances[_from] >= _value /\ allowed[_from][msg.sender] >= _value
67        @post _to != _from -> __post.balances[_from] == balances[_from] - _value
68        @post _to != _from -> __post.balances[_to] == balances[_to] + _value
69        @post _to == _from -> __post.balances[_from] == balances[_from]
70        @post __post.allowed[_from][msg.sender] == allowed[_from][msg.sender] - _value
71      */
72      function transferFrom(address _from, address _to, uint256 _value) public returns
73      (bool success) {
74          require(balances[_from] >= _value && allowed[_from][msg.sender] >= _value);
75          require(balances[_to] + _value >= balances[_to]);
76          balances[_to] += _value;
77          balances[_from] -= _value;
78          allowed[_from][msg.sender] -= _value;
79          Transfer(_from, _to, _value);
80          return true;
81      }
82      //@CTK NO_OVERFLOW
83      //@CTK NO_BUF_OVERFLOW
84      //@CTK NO_ASF
85      /*@CTK balanceOf
86        @post __reverted == false
87        @post balance == balances[_owner]
88        @post this == __post
89      */
90      function balanceOf(address _owner) public constant returns (uint256 balance) {
91          return balances[_owner];
92      }
93
94      //@CTK NO_OVERFLOW
95      //@CTK NO_BUF_OVERFLOW
96      //@CTK NO_ASF
97      /*@CTK approve
98        @tag assume_completion
99        @post _value == 0 \/ allowed[msg.sender][_spender] == 0
100       @post __post.allowed[msg.sender][_spender] == _value
101      */
102     function approve(address _spender, uint256 _value) public returns (bool success)
103     {
104         require((_value == 0) || (allowed[msg.sender][_spender] == 0));
105         allowed[msg.sender][_spender] = _value;
106         Approval(msg.sender, _spender, _value);
107         return true;
108     }
109
```

```
110     //@CTK NO_OVERFLOW
111     //@CTK NO_BUF_OVERFLOW
112     //@CTK NO_ASF
113     /*@CTK allowance
114       @post __reverted == false
115       @post remaining == allowed[_owner][_spender]
116       @post this == __post
117     */
118     function allowance(address _owner, address _spender) public constant returns (
            uint256 remaining) {
119         return allowed[_owner][_spender];
120     }
121     mapping (address => uint256) balances;
122     mapping (address => mapping (address => uint256)) allowed;
123 }
```