

# CERTIK VERIFICATION REPORT FOR TELCOIN



Request Date: 2018-11-19  
Revision Date: 2018-11-24

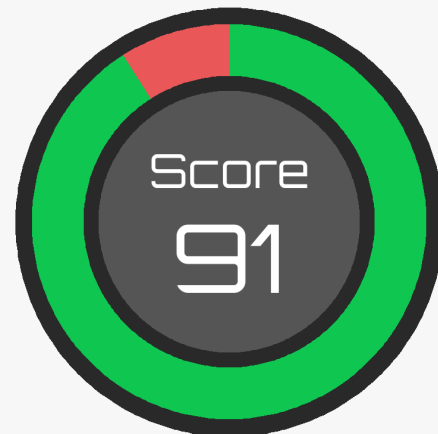
## Disclaimer

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 Telcoin(the “Company”), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the “Agreement”). This Report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This Report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiKs prior written consent.

# PASS

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

Nov 24, 2018



## Summary

This is the report for smart contract verification service requested by Telcoin. The goal of the audition is to guarantee that verified smart contracts are robust enough to avoid potentially unexpected loopholes.

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

## Type of Issues

CertiK smart label engine applied 100% covered formal verification labels on the source code, and scanned the code by 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	Com-	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	Ran-	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	au-	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115
Delegatecall to Untrusted Callee	to	Calling into untrusted contracts is very dangerous, the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Variable	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Default	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.	7	SWC-110
Deprecated Solidity Features		Several functions and operators in Solidity are deprecated and should not be used as best practice.	7	SWC-111
Unused variables		Unused variables reduce code quality	0	

## Vulnerability Details

### Critical

No issue found.

### Medium

No issue found.

### Low

#### Insecure Compiler Version:

- Using 0.4.18 is susceptible to *ExpExponentCleanup*, *EventStructWrongData* and *NestedArrayFunctionCallDecoder*.

#### Assertion failure

The Solidity `assert()` function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.

- `transfer()`, `transferFrom()` and `increaseApproval()`'s assertion errors are caused by using `SafeMath`.
- In `SafeMath.sol`, use `require()` to replace `assert()`.

### Deprecated Solidity Features:

The contract is using some deprecated feature, but no security concern found yet. since the compiler version is fixed to 0.4.18.

- Invoking events without `emit` prefix is deprecated since 0.4.21.
- Defining constructors as functions with the same name as the contract is deprecated since 0.4.22. Use `constructor()` instead.

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

- **Critical:** The code implementation does not match the specification, or it could result in loss of funds for contract owner or users.
- **Medium:** The code implementation does not match the specification at certain condition, or it could affect the security standard by lost of access control.
- **Low:** The code implementation is not a best practice, or use a suboptimal design pattern, which may lead to security vulnerability, but no concern found yet.

## Source Code with CertiK Labels

File Telcoin.sol

```

1  pragma solidity 0.4.18;
2
3  import "./lib/SafeMath.sol";
4
5
6  contract Telcoin {
7      using SafeMath for uint256;
8
9      event Transfer(address indexed _from, address indexed _to, uint _value);
10     event Approval(address indexed _owner, address indexed _spender, uint _value);
11
12     string public constant name = "Telcoin";
13     string public constant symbol = "TEL";
14     uint8 public constant decimals = 2;
15
16     /// The ERC20 total fixed supply of tokens.
17     uint256 public constant totalSupply = 100000000000 * (10 ** uint256(decimals));
18
19     /// Account balances.
20     mapping(address => uint256) balances;
21
22     /// The transfer allowances.
23     mapping (address => mapping (address => uint256)) internal allowed;
24
25     /// The initial distributor is responsible for allocating the supply
26     /// into the various pools described in the whitepaper. This can be
27     /// verified later from the event log.
28     //@CTK NO_OVERFLOW
29     //@CTK NO_BUF_OVERFLOW
30     //@CTK NO_ASF
31     /*@CTK "Telcoin constructor"
32     @post (__post.balances[_distributor]) == (totalSupply)
33     */
34     function Telcoin(address _distributor) public {
35         balances[_distributor] = totalSupply;
36         Transfer(0x0, _distributor, totalSupply);
37     }
38
39     /// ERC20 balanceOf().
40     //@CTK NO_OVERFLOW
41     //@CTK NO_BUF_OVERFLOW
42     //@CTK NO_ASF
43     /*@CTK "balanceOf"
44     @post (__reverted) == (false)
45     @post (__return) == (balances[_owner])
46     @post (this) == (__post)
47     */
48     function balanceOf(address _owner) public view returns (uint256) {
49         return balances[_owner];
50     }
51
52     /// ERC20 transfer().
53     //@CTK NO_OVERFLOW
54     //@CTK NO_BUF_OVERFLOW

```

```

55 // @CTK FAIL NO_ASF
56 /* @CTK "transfer2_same"
57     @pre (__reverted) == (false)
58     @pre (_to) == (msg.sender)
59     @post (__post.balances[_to]) == (balances[_to])
60     @post (__return) == (true)
61 */
62 /* @CTK "transfer2"
63     @pre (__reverted) == (false)
64     @pre (_to) != (msg.sender)
65     @post (__post.balances[_to]) == ((balances[_to]) + (_value))
66     @post (__post.balances[msg.sender]) == ((balances[msg.sender]) - (_value))
67     @post (__return) == (true)
68 */
69 function transfer(address _to, uint256 _value) public returns (bool) {
70     require(_to != address(0));
71     require(_value <= balances[msg.sender]);
72
73     // SafeMath.sub will throw if there is not enough balance.
74     balances[msg.sender] = balances[msg.sender].sub(_value);
75     balances[_to] = balances[_to].add(_value);
76     Transfer(msg.sender, _to, _value);
77     return true;
78 }
79
80 /// ERC20 transferFrom().
81 /// @CTK NO_OVERFLOW
82 /// @CTK NO_BUF_OVERFLOW
83 /// @CTK FAIL NO_ASF
84 /* @CTK "transferFrom"
85     @pre (__reverted) == (false)
86     @pre (_from) != (_to)
87     @post (__return) == (true)
88     @post (__post.balances[_to]) == ((balances[_to]) + (_value))
89     @post (__post.balances[_from]) == ((balances[_from]) - (_value))
90 */
91 /* @CTK "transferFrom_same"
92     @pre (__reverted) == (false)
93     @pre (_from) == (_to)
94     @post (__return) == (true)
95     @post (__post.balances[_from]) == (balances[_from])
96 */
97 function transferFrom(address _from, address _to, uint256 _value) public returns (
98     bool) {
99     require(_to != address(0));
100    require(_value <= balances[_from]);
101    require(_value <= allowed[_from][msg.sender]);
102
103    balances[_from] = balances[_from].sub(_value);
104    balances[_to] = balances[_to].add(_value);
105    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
106    Transfer(_from, _to, _value);
107    return true;
108 }
109
110 /// ERC20 approve(). Comes with the standard caveat that an approval
111 /// meant to limit spending may actually allow more to be spent due to
112 /// unfortunate ordering of transactions. For safety, this method

```

```

112     /// should only be called if the current allowance is 0. Alternatively,
113     /// non-ERC20 increaseApproval() and decreaseApproval() can be used.
114     ///@CTK NO_OVERFLOW
115     ///@CTK NO_BUF_OVERFLOW
116     ///@CTK NO_ASF
117     /*@CTK "approve correctness"
118     @post __post.allowed[msg.sender][_spender] == _value
119     @post (__return) == (true)
120     */
121     function approve(address _spender, uint256 _value) public returns (bool) {
122         allowed[msg.sender][_spender] = _value;
123         Approval(msg.sender, _spender, _value);
124         return true;
125     }
126
127     /// ERC20 allowance().
128     ///@CTK NO_OVERFLOW
129     ///@CTK NO_BUF_OVERFLOW
130     ///@CTK NO_ASF
131     /*@CTK "allowance correctness"
132     @post __return == allowed[_owner][_spender]
133     */
134     function allowance(address _owner, address _spender) public view returns (uint256)
135     {
136         return allowed[_owner][_spender];
137     }
138
139     /// Not officially ERC20. Allows an allowance to be increased safely.
140     ///@CTK NO_OVERFLOW
141     ///@CTK NO_BUF_OVERFLOW
142     ///@CTK FAIL_NO_ASF
143     /*@CTK "increaseApproval correctness"
144     @tag assume_completion
145     @post __post.allowed[msg.sender][_spender] == allowed[msg.sender][_spender] +
146         _addedValue
147     @post (__return) == (true)
148     */
149     function increaseApproval(address _spender, uint _addedValue) public returns (bool)
150     {
151         allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
152         Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
153         return true;
154     }
155
156     /// Not officially ERC20. Allows an allowance to be decreased safely.
157     ///@CTK NO_OVERFLOW
158     ///@CTK NO_BUF_OVERFLOW
159     /*@CTK "decreaseApproval correctness case 1"
160     @pre allowed[msg.sender][_spender] < _subtractedValue
161     @post __post.allowed[msg.sender][_spender] == 0
162     @post __return == true
163     @post (!__has_assertion_failure)
164     */
165     /*@CTK "decreaseApproval correctness case 2"
166     @pre allowed[msg.sender][_spender] >= _subtractedValue
167     @post __post.allowed[msg.sender][_spender] == allowed[msg.sender][_spender] -
168         _subtractedValue
169     @post __return == true

```



```

166     @post (!__has_assertion_failure)
167     */
168     function decreaseApproval(address _spender, uint _subtractedValue) public returns
        (bool) {
169         uint oldValue = allowed[msg.sender][_spender];
170         if (_subtractedValue > oldValue) {
171             allowed[msg.sender][_spender] = 0;
172         } else {
173             allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
174         }
175         Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
176         return true;
177     }
178 }

```

File lib/SafeMath.sol

```

1  pragma solidity 0.4.18;
2
3
4  /**
5   * @title SafeMath
6   * @dev Math operations with safety checks that throw on error
7   */
8  library SafeMath {
9
10     //@CTK NO_BUF_OVERFLOW
11     //@CTK FAIL_NO_ASF
12     /*@CTK "SafeMath mul"
13     @post (a > 0) && (((a * b) / a) != b) -> (__has_assertion_failure)
14     @post (__has_assertion_failure) -> (a > 0) && (((a * b) / a) != b)
15     @post __has_assertion_failure == __reverted
16     @post !__reverted -> __return == a * b
17     @post !__reverted == !__has_overflow
18     */
19     function mul(uint256 a, uint256 b) internal pure returns (uint256) {
20         uint256 c = a * b;
21         assert(a == 0 || c / a == b);
22         return c;
23     }
24
25     //@CTK NO_BUF_OVERFLOW
26     //@CTK FAIL_NO_ASF
27     /*@CTK "SafeMath div"
28     @post b != 0 -> !__reverted
29     @post !__reverted -> __return == a / b
30     @post !__reverted -> !__has_overflow
31     */
32     function div(uint256 a, uint256 b) internal pure returns (uint256) {
33         // assert(b > 0); // Solidity automatically throws when dividing by 0
34         uint256 c = a / b;
35         // assert(a == b * c + a % b); // There is no case in which this doesn't hold
36         return c;
37     }
38
39     //@CTK NO_BUF_OVERFLOW
40     //@CTK FAIL_NO_ASF
41     /*@CTK "SafeMath sub"
42     @post (a < b) == __reverted

```

```
43     @post !__reverted -> __return == a - b
44     @post !__reverted -> !__has_overflow
45     */
46     function sub(uint256 a, uint256 b) internal pure returns (uint256) {
47         assert(b <= a);
48         return a - b;
49     }
50
51     //@CTK NO_BUF_OVERFLOW
52     //@CTK FAIL_NO_ASF
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 add(uint256 a, uint256 b) internal pure returns (uint256) {
59         uint256 c = a + b;
60         assert(c >= a);
61         return c;
62     }
63 }
```

## 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          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      } </pre>

Counterexample	 This code violates the specification
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>

## Static Analysis Request

### INSECURE\_COMPILER\_VERSION

Line 1 in File Telcoin.sol

```
1 pragma solidity 0.4.18;
```

! Version to compile has the following bug: 0.4.18: ExpExponentCleanup, EventStructWrongData, NestedArrayFunctionCallDecoder

### INSECURE\_COMPILER\_VERSION

Line 1 in File SafeMath.sol


```
1 pragma solidity 0.4.18;
```

! Version to compile has the following bug: 0.4.18: ExpExponentCleanup, EventStructWrongData, NestedArrayFunctionCallDecoder

## Formal Verification Request 1

If method completes, integer overflow would not happen.

 24, Nov 2018

 26.5 ms

Line 28 in File Telcoin.sol

```
28 // @CTK NO_OVERFLOW
```

Line 34-37 in File Telcoin.sol


```
34 function Telcoin(address _distributor) public {
35     balances[_distributor] = totalSupply;
36     Transfer(0x0, _distributor, totalSupply);
37 }
```

 The code meets the specification

## Formal Verification Request 2

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

 24, Nov 2018

 0.71 ms

Line 29 in File Telcoin.sol

```
29 // @CTK NO_BUF_OVERFLOW
```

Line 34-37 in File Telcoin.sol


```
34 function Telcoin(address _distributor) public {
35     balances[_distributor] = totalSupply;
36     Transfer(0x0, _distributor, totalSupply);
37 }
```

 The code meets the specification

## Formal Verification Request 3

Method will not encounter an assertion failure.

 24, Nov 2018

 0.7 ms

Line 30 in File Telcoin.sol

```
30 // @CTK NO_ASF
```

Line 34-37 in File Telcoin.sol

```
34 function Telcoin(address _distributor) public {
35     balances[_distributor] = totalSupply;
36     Transfer(0x0, _distributor, totalSupply);
37 }
```

✓ The code meets the specification

## Formal Verification Request 4

Telcoin constructor

📅 24, Nov 2018

🕒 3.66 ms

Line 31-33 in File Telcoin.sol

```
31  /*@CTK "Telcoin constructor"
32     @post (__post.balances[_distributor]) == (totalSupply)
33     */
```

Line 34-37 in File Telcoin.sol

```
34  function Telcoin(address _distributor) public {
35      balances[_distributor] = totalSupply;
36      Transfer(0x0, _distributor, totalSupply);
37  }
```

✓ The code meets the specification

## Formal Verification Request 5

If method completes, integer overflow would not happen.

📅 24, Nov 2018

🕒 14.59 ms

Line 40 in File Telcoin.sol

```
40  //@CTK NO_OVERFLOW
```

Line 48-50 in File Telcoin.sol

```
48  function balanceOf(address _owner) public view returns (uint256) {
49      return balances[_owner];
50  }
```

✓ The code meets the specification

## Formal Verification Request 6

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

📅 24, Nov 2018

🕒 0.72 ms

Line 41 in File Telcoin.sol

```
41  //@CTK NO_BUF_OVERFLOW
```

Line 48-50 in File Telcoin.sol

```
48     function balanceOf(address _owner) public view returns (uint256) {
49         return balances[_owner];
50     }
```

✔ The code meets the specification

## Formal Verification Request 7

Method will not encounter an assertion failure.

📅 24, Nov 2018

🕒 0.73 ms

Line 42 in File Telcoin.sol

```
42     //@CTK NO_ASF
```

Line 48-50 in File Telcoin.sol

```
48     function balanceOf(address _owner) public view returns (uint256) {
49         return balances[_owner];
50     }
```

✔ The code meets the specification

## Formal Verification Request 8

balanceOf

📅 24, Nov 2018

🕒 0.76 ms

Line 43-47 in File Telcoin.sol

```
43     /*@CTK "balanceOf"
44         @post (__reverted) == (false)
45         @post (__return) == (balances[_owner])
46         @post (this) == (__post)
47     */
```

Line 48-50 in File Telcoin.sol


```
48     function balanceOf(address _owner) public view returns (uint256) {
49         return balances[_owner];
50     }
```

✔ The code meets the specification

## Formal Verification Request 9

If method completes, integer overflow would not happen.

 24, Nov 2018

 337.97 ms

Line 53 in File Telcoin.sol

```
53 // @CTK_NO_OVERFLOW
```

Line 69-78 in File Telcoin.sol


```
69 function transfer(address _to, uint256 _value) public returns (bool) {
70     require(_to != address(0));
71     require(_value <= balances[msg.sender]);
72
73     // SafeMath.sub will throw if there is not enough balance.
74     balances[msg.sender] = balances[msg.sender].sub(_value);
75     balances[_to] = balances[_to].add(_value);
76     Transfer(msg.sender, _to, _value);
77     return true;
78 }
```

 The code meets the specification

## Formal Verification Request 10

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

 24, Nov 2018

 34.01 ms

Line 54 in File Telcoin.sol

```
54 // @CTK_NO_BUF_OVERFLOW
```

Line 69-78 in File Telcoin.sol

```
69 function transfer(address _to, uint256 _value) public returns (bool) {
70     require(_to != address(0));
71     require(_value <= balances[msg.sender]);
72
73     // SafeMath.sub will throw if there is not enough balance.
74     balances[msg.sender] = balances[msg.sender].sub(_value);
75     balances[_to] = balances[_to].add(_value);
76     Transfer(msg.sender, _to, _value);
77     return true;
78 }
```


 The code meets the specification



## Formal Verification Request 11

Method will not encounter an assertion failure.

 24, Nov 2018

 188.27 ms

Line 55 in File Telcoin.sol

```
55 // @CTK FAIL NO_ASF
```

Line 69-78 in File Telcoin.sol

```
69 function transfer(address _to, uint256 _value) public returns (bool) {
70     require(_to != address(0));
71     require(_value <= balances[msg.sender]);
72
73     // SafeMath.sub will throw if there is not enough balance.
74     balances[msg.sender] = balances[msg.sender].sub(_value);
75     balances[_to] = balances[_to].add(_value);
76     Transfer(msg.sender, _to, _value);
77     return true;
78 }
```

 This code violates the specification

```
1 Counter Example:
2 Before Execution:
3   Input = {
4     _to = 32
5     _value = 34
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": "Telcoin",
32        "balance": 0,
33        "contract": {
34          "name": "",
```

```


35     "symbol": "",
36     "decimals": 0,
37     "totalSupply": 0,
38     "balances": [
39         {
40             "key": 32,
41             "value": 252
42         },
43         {
44             "key": 64,
45             "value": 0
46         },
47         {
48             "key": 40,
49             "value": 0
50         },
51         {
52             "key": 2,
53             "value": 16
54         },
55         {
56             "key": 8,
57             "value": 32
58         },
59         {
60             "key": "ALL_OTHERS",
61             "value": 128
62         }
63     ],
64     "allowed": [
65         {
66             "key": "ALL_OTHERS",
67             "value": [
68                 {
69                     "key": "ALL_OTHERS",
70                     "value": 128
71                 }
72             ]
73         }
74     ]
75 }
76 }
77 },
78 {
79     "key": "ALL_OTHERS",
80     "value": "EmptyAddress"
81 }
82 ]
83
84 Function invocation is reverted.

```

## Formal Verification Request 12

transfer2\_same

 24, Nov 2018

 122.21 ms

Line 56-61 in File Telcoin.sol

```
56  /*@CTK "transfer2_same"
57     @pre (__reverted) == (false)
58     @pre (_to) == (msg.sender)
59     @post (__post.balances[_to]) == (balances[_to])
60     @post (__return) == (true)
61  */
```

Line 69-78 in File Telcoin.sol


```
69  function transfer(address _to, uint256 _value) public returns (bool) {
70      require(_to != address(0));
71      require(_value <= balances[msg.sender]);
72
73      // SafeMath.sub will throw if there is not enough balance.
74      balances[msg.sender] = balances[msg.sender].sub(_value);
75      balances[_to] = balances[_to].add(_value);
76      Transfer(msg.sender, _to, _value);
77      return true;
78  }
```

 The code meets the specification

## Formal Verification Request 13

transfer2

 24, Nov 2018

 150.35 ms

Line 62-68 in File Telcoin.sol

```
62  /*@CTK "transfer2"
63     @pre (__reverted) == (false)
64     @pre (_to) != (msg.sender)
65     @post (__post.balances[_to]) == ((balances[_to]) + (_value))
66     @post (__post.balances[msg.sender]) == ((balances[msg.sender]) - (_value))
67     @post (__return) == (true)
68  */
```

Line 69-78 in File Telcoin.sol


```
69  function transfer(address _to, uint256 _value) public returns (bool) {
70      require(_to != address(0));
71      require(_value <= balances[msg.sender]);
72
73      // SafeMath.sub will throw if there is not enough balance.
74      balances[msg.sender] = balances[msg.sender].sub(_value);
75      balances[_to] = balances[_to].add(_value);
76      Transfer(msg.sender, _to, _value);
77      return true;
78  }
```

 The code meets the specification

## Formal Verification Request 14

If method completes, integer overflow would not happen.

 24, Nov 2018

 320.86 ms

Line 81 in File Telcoin.sol

```
81 // @CTK_NO_OVERFLOW
```

Line 97-107 in File Telcoin.sol


```
97 function transferFrom(address _from, address _to, uint256 _value) public returns (
    bool) {
98     require(_to != address(0));
99     require(_value <= balances[_from]);
100    require(_value <= allowed[_from][msg.sender]);
101
102    balances[_from] = balances[_from].sub(_value);
103    balances[_to] = balances[_to].add(_value);
104    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
105    Transfer(_from, _to, _value);
106    return true;
107 }
```

 The code meets the specification

## Formal Verification Request 15

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

 24, Nov 2018

 93.37 ms

Line 82 in File Telcoin.sol

```
82 // @CTK_NO_BUF_OVERFLOW
```

Line 97-107 in File Telcoin.sol

```
97 function transferFrom(address _from, address _to, uint256 _value) public returns (
    bool) {
98     require(_to != address(0));
99     require(_value <= balances[_from]);
100    require(_value <= allowed[_from][msg.sender]);
101
102    balances[_from] = balances[_from].sub(_value);
103    balances[_to] = balances[_to].add(_value);
104    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
105    Transfer(_from, _to, _value);
106    return true;
107 }
```

 The code meets the specification

## Formal Verification Request 16

Method will not encounter an assertion failure.

📅 24, Nov 2018

🕒 534.28 ms

Line 83 in File Telcoin.sol

```
83 //©TK FAIL NO_ASF
```

Line 97-107 in File Telcoin.sol

```

97 function transferFrom(address _from, address _to, uint256 _value) public returns (
    bool) {
98     require(_to != address(0));
99     require(_value <= balances[_from]);
100    require(_value <= allowed[_from][msg.sender]);
101
102    balances[_from] = balances[_from].sub(_value);
103    balances[_to] = balances[_to].add(_value);
104    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
105    Transfer(_from, _to, _value);
106    return true;
107 }

```

❌ This code violates the specification

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     _from = 0
5     _to = 32
6     _value = 1
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": "Telcoin",

```

```

33     "balance": 0,
34     "contract": {
35         "name": "",
36         "symbol": "",
37         "decimals": 0,
38         "totalSupply": 0,
39         "balances": [
40             {
41                 "key": 64,
42                 "value": 0
43             },
44             {
45                 "key": 0,
46                 "value": 64
47             },
48             {
49                 "key": 4,
50                 "value": 64
51             },
52             {
53                 "key": 2,
54                 "value": 0
55             },
56             {
57                 "key": 8,
58                 "value": 0
59             },
60             {
61                 "key": 34,
62                 "value": 0
63             },
64             {
65                 "key": 1,
66                 "value": 0
67             },
68             {
69                 "key": 32,
70                 "value": 255
71             },
72             {
73                 "key": "ALL_OTHERS",
74                 "value": 128
75             }
76         ],
77         "allowed": [
78             {
79                 "key": 0,
80                 "value": [
81                     {
82                         "key": 0,
83                         "value": 128
84                     },
85                     {
86                         "key": 2,
87                         "value": 8
88                     },
89                     {
90                         "key": "ALL_OTHERS",

```

```


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

```

## Formal Verification Request 17

transferFrom

 24, Nov 2018

 335.03 ms

Line 84-90 in File Telcoin.sol

```

84  /*@CTK "transferFrom"
85     @pre (__reverted) == (false)
86     @pre (_from) != (_to)
87     @post (__return) == (true)
88     @post (__post.balances[_to]) == ((balances[_to]) + (_value))
89     @post (__post.balances[_from]) == ((balances[_from]) - (_value))
90  */

```

Line 97-107 in File Telcoin.sol

```

97     function transferFrom(address _from, address _to, uint256 _value) public returns (
98         bool) {
99         require(_to != address(0));
100        require(_value <= balances[_from]);
101        require(_value <= allowed[_from][msg.sender]);
102
103        balances[_from] = balances[_from].sub(_value);
104        balances[_to] = balances[_to].add(_value);
105        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
106        Transfer(_from, _to, _value);
107        return true;
108    }

```

✔ The code meets the specification

## Formal Verification Request 18

transferFrom\_same

📅 24, Nov 2018

🕒 120.0 ms

Line 91-96 in File Telcoin.sol

```

91     /*@CTK "transferFrom_same"
92     @pre (__reverted) == (false)
93     @pre (_from) == (_to)
94     @post (__return) == (true)
95     @post (__post.balances[_from]) == (balances[_from])
96     */

```

Line 97-107 in File Telcoin.sol

```

97     function transferFrom(address _from, address _to, uint256 _value) public returns (
98         bool) {
99         require(_to != address(0));
100        require(_value <= balances[_from]);
101        require(_value <= allowed[_from][msg.sender]);
102
103        balances[_from] = balances[_from].sub(_value);
104        balances[_to] = balances[_to].add(_value);
105        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
106        Transfer(_from, _to, _value);
107        return true;
108    }

```

✔ The code meets the specification

## Formal Verification Request 19

If method completes, integer overflow would not happen.

📅 24, Nov 2018

🕒 12.9 ms



Line 114 in File Telcoin.sol

```
114 //@CTK_NO_OVERFLOW
```

Line 121-125 in File Telcoin.sol

```
121 function approve(address _spender, uint256 _value) public returns (bool) {  
122     allowed[msg.sender][_spender] = _value;  
123     Approval(msg.sender, _spender, _value);  
124     return true;  
125 }
```

✔ The code meets the specification

## Formal Verification Request 20

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

📅 24, Nov 2018

🕒 0.52 ms

Line 115 in File Telcoin.sol

```
115 //@CTK_NO_BUF_OVERFLOW
```

Line 121-125 in File Telcoin.sol

```
121 function approve(address _spender, uint256 _value) public returns (bool) {  
122     allowed[msg.sender][_spender] = _value;  
123     Approval(msg.sender, _spender, _value);  
124     return true;  
125 }
```

✔ The code meets the specification

## Formal Verification Request 21

Method will not encounter an assertion failure.

📅 24, Nov 2018

🕒 0.48 ms

Line 116 in File Telcoin.sol

```
116 //@CTK_NO_ASF
```

Line 121-125 in File Telcoin.sol


```
121 function approve(address _spender, uint256 _value) public returns (bool) {  
122     allowed[msg.sender][_spender] = _value;  
123     Approval(msg.sender, _spender, _value);  
124     return true;  
125 }
```

✔ The code meets the specification

## Formal Verification Request 22

approve correctness

 24, Nov 2018

 1.96 ms

Line 117-120 in File Telcoin.sol

```
117  /*@CTK "approve correctness"
118     @post __post.allowed[msg.sender][_spender] == _value
119     @post (__return) == (true)
120  */
```

Line 121-125 in File Telcoin.sol


```
121  function approve(address _spender, uint256 _value) public returns (bool) {
122      allowed[msg.sender][_spender] = _value;
123      Approval(msg.sender, _spender, _value);
124      return true;
125  }
```

 The code meets the specification

## Formal Verification Request 23

If method completes, integer overflow would not happen.

 24, Nov 2018

 7.32 ms

Line 128 in File Telcoin.sol

```
128  //@CTK NO_OVERFLOW
```

Line 134-136 in File Telcoin.sol


```
134  function allowance(address _owner, address _spender) public view returns (uint256)
135      {
136      return allowed[_owner][_spender];
137  }
```

 The code meets the specification

## Formal Verification Request 24

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

 24, Nov 2018

 0.44 ms

Line 129 in File Telcoin.sol

```
129  //@CTK NO_BUF_OVERFLOW
```

Line 134-136 in File Telcoin.sol

```

134     function allowance(address _owner, address _spender) public view returns (uint256)
135         {
136             return allowed[_owner][_spender];
        }
    
```

✓ The code meets the specification

## Formal Verification Request 25

Method will not encounter an assertion failure.

📅 24, Nov 2018

🕒 0.41 ms

Line 130 in File Telcoin.sol

```

130     //@CTK NO_ASF
    
```

Line 134-136 in File Telcoin.sol

```

134     function allowance(address _owner, address _spender) public view returns (uint256)
135         {
136             return allowed[_owner][_spender];
        }
    
```

✓ The code meets the specification

## Formal Verification Request 26

allowance correctness

📅 24, Nov 2018

🕒 0.43 ms

Line 131-133 in File Telcoin.sol

```

131     /*@CTK "allowance correctness"
132         @post __return == allowed[_owner][_spender]
133     */
    
```

Line 134-136 in File Telcoin.sol

```


134     function allowance(address _owner, address _spender) public view returns (uint256)
135         {
136             return allowed[_owner][_spender];
        }
    
```

✓ The code meets the specification

## Formal Verification Request 27

If method completes, integer overflow would not happen.

 24, Nov 2018

 51.36 ms

Line 139 in File Telcoin.sol

```
139 // @CTK_NO_OVERFLOW
```

Line 147-151 in File Telcoin.sol


```
147 function increaseApproval(address _spender, uint _addedValue) public returns (bool
    ) {
148     allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
149     Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
150     return true;
151 }
```

 The code meets the specification

## Formal Verification Request 28

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

 24, Nov 2018

 1.01 ms

Line 140 in File Telcoin.sol

```
140 // @CTK_NO_BUF_OVERFLOW
```

Line 147-151 in File Telcoin.sol


```
147 function increaseApproval(address _spender, uint _addedValue) public returns (bool
    ) {
148     allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
149     Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
150     return true;
151 }
```

 The code meets the specification

## Formal Verification Request 29

Method will not encounter an assertion failure.

 24, Nov 2018

 19.14 ms

Line 141 in File Telcoin.sol

```
141 // @CTK_FAIL_NO_ASF
```

Line 147-151 in File Telcoin.sol

```

147 function increaseApproval(address _spender, uint _addedValue) public returns (bool
    ) {
148     allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
149     Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
150     return true;
151 }

```

✘ This code violates the specification

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     _addedValue = 53
5     _spender = 0
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": "Telcoin",
32        "balance": 0,
33        "contract": {
34          "name": "",
35          "symbol": "",
36          "decimals": 0,
37          "totalSupply": 0,
38          "balances": [
39            {
40              "key": 16,
41              "value": 2
42            },
43            {
44              "key": 128,
45              "value": 16
46            },
47            {
48              "key": 0,
49              "value": 0
50            },

```

```

51     {
52         "key": "ALL_OTHERS",
53         "value": 53
54     }
55 ],
56 "allowed": [
57     {
58         "key": 0,
59         "value": [
60             {
61                 "key": 128,
62                 "value": 0
63             },
64             {
65                 "key": 0,
66                 "value": 203
67             },
68             {
69                 "key": 32,
70                 "value": 0
71             },
72             {
73                 "key": 2,
74                 "value": 0
75             },
76             {
77                 "key": "ALL_OTHERS",
78                 "value": 53
79             }
80         ]
81     },
82     {
83         "key": "ALL_OTHERS",
84         "value": [
85             {
86                 "key": "ALL_OTHERS",
87                 "value": 53
88             }
89         ]
90     }
91 ],
92 }
93 },
94 ],
95 {
96     "key": "ALL_OTHERS",
97     "value": "EmptyAddress"
98 }
99 ]


```

100  
101 Function invocation is reverted.

## Formal Verification Request 30

increaseApproval correctness

 24, Nov 2018

 3.72 ms

Line 142-146 in File Telcoin.sol

```
142  /*@CTK "increaseApproval correctness"
143     @tag assume_completion
144     @post __post.allowed[msg.sender][_spender] == allowed[msg.sender][_spender] +
         _addedValue
145     @post (__return) == (true)
146     */
```

Line 147-151 in File Telcoin.sol


```
147  function increaseApproval(address _spender, uint _addedValue) public returns (bool
    ) {
148     allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
149     Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
150     return true;
151 }
```

 The code meets the specification

## Formal Verification Request 31

If method completes, integer overflow would not happen.

 24, Nov 2018

 107.67 ms

Line 154 in File Telcoin.sol

```
154  //@CTK NO_OVERFLOW
```

Line 168-177 in File Telcoin.sol


```
168  function decreaseApproval(address _spender, uint _subtractedValue) public returns
    (bool) {
169     uint oldValue = allowed[msg.sender][_spender];
170     if (_subtractedValue > oldValue) {
171         allowed[msg.sender][_spender] = 0;
172     } else {
173         allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
174     }
175     Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
176     return true;
177 }
```

 The code meets the specification

## Formal Verification Request 32

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

 24, Nov 2018

 3.18 ms

Line 155 in File Telcoin.sol

```
155 // @CTK NO_BUF_OVERFLOW
```

Line 168-177 in File Telcoin.sol


```
168 function decreaseApproval(address _spender, uint _subtractedValue) public returns
    (bool) {
169     uint oldValue = allowed[msg.sender][_spender];
170     if (_subtractedValue > oldValue) {
171         allowed[msg.sender][_spender] = 0;
172     } else {
173         allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
174     }
175     Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
176     return true;
177 }
```

 The code meets the specification

## Formal Verification Request 33

decreaseApproval correctness case 1

 24, Nov 2018

 6.86 ms

Line 156-161 in File Telcoin.sol

```
156 /* @CTK "decreaseApproval correctness case 1"
157     @pre allowed[msg.sender][_spender] < _subtractedValue
158     @post __post.allowed[msg.sender][_spender] == 0
159     @post __return == true
160     @post (!__has_assertion_failure)
161 */
```

Line 168-177 in File Telcoin.sol

```
168 function decreaseApproval(address _spender, uint _subtractedValue) public returns
    (bool) {
169     uint oldValue = allowed[msg.sender][_spender];
170     if (_subtractedValue > oldValue) {
171         allowed[msg.sender][_spender] = 0;
172     } else {
173         allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
174     }
175     Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
176     return true;
177 }
```


 The code meets the specification



## Formal Verification Request 34

decreaseApproval correctness case 2

 24, Nov 2018

 14.54 ms

Line 162-167 in File Telcoin.sol

```
162 /*@CTK "decreaseApproval correctness case 2"
163    @pre allowed[msg.sender][_spender] >= _subtractedValue
164    @post __post.allowed[msg.sender][_spender] == allowed[msg.sender][_spender] -
        _subtractedValue
165    @post __return == true
166    @post (!__has_assertion_failure)
167 */
```

Line 168-177 in File Telcoin.sol


```
168 function decreaseApproval(address _spender, uint _subtractedValue) public returns
    (bool) {
169     uint oldValue = allowed[msg.sender][_spender];
170     if (_subtractedValue > oldValue) {
171         allowed[msg.sender][_spender] = 0;
172     } else {
173         allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
174     }
175     Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
176     return true;
177 }
```

 The code meets the specification

## Formal Verification Request 35

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

 24, Nov 2018

 49.09 ms

Line 10 in File SafeMath.sol

```
10 // @CTK NO_BUF_OVERFLOW
```

Line 19-23 in File SafeMath.sol


```
19 function mul(uint256 a, uint256 b) internal pure returns (uint256) {
20     uint256 c = a * b;
21     assert(a == 0 || c / a == b);
22     return c;
23 }
```

 The code meets the specification

## Formal Verification Request 36

Method will not encounter an assertion failure.

 24, Nov 2018


 4.42 ms

Line 11 in File SafeMath.sol

```
11 // @CTK FAIL NO_ASF
```

Line 19-23 in File SafeMath.sol

```
19 function mul(uint256 a, uint256 b) internal pure returns (uint256) {
20     uint256 c = a * b;
21     assert(a == 0 || c / a == b);
22     return c;
23 }
```


 This code violates the specification

```
1 Counter Example:
2 Before Execution:
3   Input = {
4     a = 94
5     b = 128
6   }
7   Internal = {
8     __has_assertion_failure = false
9     __has_buf_overflow = false
10    __has_overflow = false
11    __has_returned = false
12    __reverted = false
13    msg = {
14      "gas": 0,
15      "sender": 0,
16      "value": 0
17    }
18  }
19   Other = {
20     __return = 0
21     block = {
22       "number": 0,
23       "timestamp": 0
24     }
25   }
26   Address_Map = [
27     {
28       "key": "ALL_OTHERS",
29       "value": "EmptyAddress"
30     }
31   ]
32
33 Function invocation is reverted.
```

## Formal Verification Request 37

SafeMath mul

 24, Nov 2018

 183.34 ms

Line 12-18 in File SafeMath.sol

```
12  /*@CTK "SafeMath mul"
13     @post (a > 0) && (((a * b) / a) != b) -> (__has_assertion_failure)
14     @post (__has_assertion_failure) -> (a > 0) && (((a * b) / a) != b)
15     @post __has_assertion_failure == __reverted
16     @post !__reverted -> __return == a * b
17     @post !__reverted == !__has_overflow
18  */
```

Line 19-23 in File SafeMath.sol


```
19  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
20      uint256 c = a * b;
21      assert(a == 0 || c / a == b);
22      return c;
23  }
```

 The code meets the specification

## Formal Verification Request 38

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

 24, Nov 2018

 18.08 ms

Line 25 in File SafeMath.sol

```
25  //@CTK NO_BUF_OVERFLOW
```

Line 32-37 in File SafeMath.sol


```
32  function div(uint256 a, uint256 b) internal pure returns (uint256) {
33      // assert(b > 0); // Solidity automatically throws when dividing by 0
34      uint256 c = a / b;
35      // assert(a == b * c + a % b); // There is no case in which this doesn't hold
36      return c;
37  }
```

 The code meets the specification

## Formal Verification Request 39

Method will not encounter an assertion failure.

 24, Nov 2018

 1.12 ms

Line 26 in File SafeMath.sol

```
26 // @CTK FAIL NO_ASF
```

Line 32-37 in File SafeMath.sol

```
32 function div(uint256 a, uint256 b) internal pure returns (uint256) {
33     // assert(b > 0); // Solidity automatically throws when dividing by 0
34     uint256 c = a / b;
35     // assert(a == b * c + a % b); // There is no case in which this doesn't hold
36     return c;
37 }
```


✘ This code violates the specification

```
1 Counter Example:
2 Before Execution:
3   Input = {
4     a = 0
5     b = 0
6   }
7   Internal = {
8     __has_assertion_failure = false
9     __has_buf_overflow = false
10    __has_overflow = false
11    __has_returned = false
12    __reverted = false
13    msg = {
14      "gas": 0,
15      "sender": 0,
16      "value": 0
17    }
18  }
19  Other = {
20    __return = 0
21    block = {
22      "number": 0,
23      "timestamp": 0
24    }
25  }
26  Address_Map = [
27    {
28      "key": "ALL_OTHERS",
29      "value": "EmptyAddress"
30    }
31  ]
32
33 Function invocation is reverted.
```

## Formal Verification Request 40

SafeMath div

 24, Nov 2018

 2.43 ms

Line 27-31 in File SafeMath.sol

```
27  /*@CTK "SafeMath div"
28     @post b != 0 -> !__reverted
29     @post !__reverted -> __return == a / b
30     @post !__reverted -> !__has_overflow
31  */
```

Line 32-37 in File SafeMath.sol

```
32  function div(uint256 a, uint256 b) internal pure returns (uint256) {
33      // assert(b > 0); // Solidity automatically throws when dividing by 0
34      uint256 c = a / b;
35      // assert(a == b * c + a % b); // There is no case in which this doesn't hold
36      return c;
37  }
```

✓ The code meets the specification

## Formal Verification Request 41

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

📅 24, Nov 2018

🕒 38.91 ms

Line 39 in File SafeMath.sol

```
39  //@CTK NO_BUF_OVERFLOW
```

Line 46-49 in File SafeMath.sol

```
46  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
47      assert(b <= a);
48      return a - b;
49  }
```

✓ The code meets the specification

## Formal Verification Request 42

Method will not encounter an assertion failure.

📅 24, Nov 2018

🕒 2.58 ms

Line 40 in File SafeMath.sol

```
40  //@CTK FAIL_NO_ASF
```

Line 46-49 in File SafeMath.sol

```
46  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
47      assert(b <= a);
48      return a - b;
49  }
```

✗ This code violates the specification

```


1 Counter Example:
2 Before Execution:
3   Input = {
4     a = 0
5     b = 1
6   }
7   Internal = {
8     __has_assertion_failure = false
9     __has_buf_overflow = false
10    __has_overflow = false
11    __has_returned = false
12    __reverted = false
13    msg = {
14      "gas": 0,
15      "sender": 0,
16      "value": 0
17    }
18  }
19  Other = {
20    __return = 0
21    block = {
22      "number": 0,
23      "timestamp": 0
24    }
25  }
26  Address_Map = [
27    {
28      "key": "ALL_OTHERS",
29      "value": "EmptyAddress"
30    }
31  ]
32
33 Function invocation is reverted.

```

## Formal Verification Request 43

SafeMath sub

 24, Nov 2018

 3.1 ms

Line 41-45 in File SafeMath.sol

```

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

```

Line 46-49 in File SafeMath.sol

```

46 function sub(uint256 a, uint256 b) internal pure returns (uint256) {
47   assert(b <= a);
48   return a - b;
49 }


```

 The code meets the specification

## Formal Verification Request 44

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

 24, Nov 2018

 42.93 ms

Line 51 in File SafeMath.sol

```
51 // @CTK_NO_BUF_OVERFLOW
```

Line 58-62 in File SafeMath.sol


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

 The code meets the specification

## Formal Verification Request 45

Method will not encounter an assertion failure.

 24, Nov 2018

 4.31 ms

Line 52 in File SafeMath.sol

```
52 // @CTK_FAIL_NO_ASF
```

Line 58-62 in File SafeMath.sol

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

 This code violates the specification


```
1 Counter Example:
2 Before Execution:
3   Input = {
4     a = 83
5     b = 173
6   }
7   Internal = {
8     __has_assertion_failure = false
9     __has_buf_overflow = false
10    __has_overflow = false
11    __has_returned = false
12    __reverted = false
13    msg = {
14      "gas": 0,
15      "sender": 0,
16      "value": 0
```

```
17     }
18   }
19   Other = {
20     __return = 0
21     block = {
22       "number": 0,
23       "timestamp": 0
24     }
25   }
26   Address_Map = [
27     {
28       "key": "ALL_OTHERS",
29       "value": "EmptyAddress"
30     }
31   ]
32
33 Function invocation is reverted.
```

## Formal Verification Request 46

### SafeMath add

 24, Nov 2018

 6.15 ms

Line 53-57 in File SafeMath.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 SafeMath.sol

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

 The code meets the specification