CertiK Verification Report For Telcoin

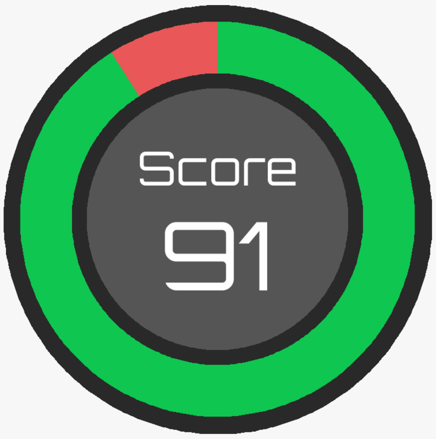
Request Date: 2018-11-19
Revision Date: 2018-11-24

Disclaimer

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 requestd 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% coveraged 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Insecure Compiler Version	Using an fixed outdated compiler version or floating pragma can be problematic, if there are publicly disclosed bugs and issues that affect the current compiler version used.	1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.	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 uing 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

```
pragma solidity 0.4.18;

import "./lib/SafeMath.sol";


contract Telcoin {
    using SafeMath for uint256;

    event Transfer(address indexed _from, address indexed _to, uint _value);
    event Approval(address indexed _owner, address indexed _spender, uint _value);

    string public constant name = "Telcoin";
    string public constant symbol = "TEL";
    uint8 public constant decimals = 2;

    /// The ERC20 total fixed supply of tokens.
    uint256 public constant totalSupply = 100000000000 * (10 ** uint256(decimals));

    /// Account balances.
    mapping(address => uint256) balances;

    /// The transfer allowances.
    mapping (address => mapping (address => uint256)) internal allowed;

    /// The initial distributor is responsible for allocating the supply
    /// into the various pools described in the whitepaper. This can be
    /// verified later from the event log.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "Telcoin constructor"
      @post (__post.balances[_distributor]) == (totalSupply)
     */
    function Telcoin(address _distributor) public {
        balances[_distributor] = totalSupply;
        Transfer(0x0, _distributor, totalSupply);
    }

    /// ERC20 balanceOf().
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "balanceOf"
      @post (__reverted) == (false)
      @post (__return) == (balances[_owner])
      @post (this) == (__post)
    */
    function balanceOf(address _owner) public view returns (uint256) {
        return balances[_owner];
    }

    /// ERC20 transfer().
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK FAIL NO_ASF
    /*@CTK "transfer2_same"
      @pre (__reverted) == (false)
      @pre (_to) == (msg.sender)
      @post (__post.balances[_to]) == (balances[_to])
      @post (__return) == (true)
    */
    /*@CTK "transfer2"
      @pre (__reverted) == (false)
      @pre (_to) != (msg.sender)
      @post (__post.balances[_to]) == ((balances[_to]) + (_value))
      @post (__post.balances[msg.sender]) == ((balances[msg.sender]) - (_value))
      @post (__return) == (true)
    */
    function transfer(address _to, uint256 _value) public returns (bool) {
        require(_to != address(0));
        require(_value <= balances[msg.sender]);

        // SafeMath.sub will throw if there is not enough balance.
        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        Transfer(msg.sender, _to, _value);
        return true;
    }

    /// ERC20 transferFrom().
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK FAIL NO_ASF
    /*@CTK "transferFrom"
      @pre (__reverted) == (false)
      @pre (_from) != (_to)
      @post (__return) == (true)
      @post (__post.balances[_to]) == ((balances[_to]) + (_value))
      @post (__post.balances[_from]) == ((balances[_from]) - (_value))
    */
    /*@CTK "transferFrom_same"
      @pre (__reverted) == (false)
      @pre (_from) == (_to)
      @post (__return) == (true)
      @post (__post.balances[_from]) == (balances[_from])
    */
    function transferFrom(address _from, address _to, uint256 _value) public returns (
        bool) {
        require(_to != address(0));
        require(_value <= balances[_from]);
        require(_value <= allowed[_from][msg.sender]);

        balances[_from] = balances[_from].sub(_value);
        balances[_to] = balances[_to].add(_value);
        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
        Transfer(_from, _to, _value);
        return true;
    }

    /// ERC20 approve(). Comes with the standard caveat that an approval
    /// meant to limit spending may actually allow more to be spent due to
    /// unfortunate ordering of transactions. For safety, this method
    /// should only be called if the current allowance is 0. Alternatively,
    /// non-ERC20 increaseApproval() and decreaseApproval() can be used.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "approve correctness"
      @post __post.allowed[msg.sender][_spender] == _value
      @post (__return) == (true)
    */
    function approve(address _spender, uint256 _value) public returns (bool) {
        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
        return true;
    }

    /// ERC20 allowance().
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "allowance correctness"
      @post __return == allowed[_owner][_spender]
    */
    function allowance(address _owner, address _spender) public view returns (uint256)
         {
        return allowed[_owner][_spender];
    }

    /// Not officially ERC20. Allows an allowance to be increased safely.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK FAIL NO_ASF
    /*@CTK "increaseApproval correctness"
      @tag assume_completion
      @post __post.allowed[msg.sender][_spender] == allowed[msg.sender][_spender] +
          _addedValue
      @post (__return) == (true)
     */
    function increaseApproval(address _spender, uint _addedValue) public returns (bool
        ) {
        allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
        Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }

    /// Not officially ERC20. Allows an allowance to be decreased safely.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    /*@CTK "decreaseApproval correctness case 1"
      @pre allowed[msg.sender][_spender] < _subtractedValue
      @post __post.allowed[msg.sender][_spender] == 0
      @post __return == true
      @post (!__has_assertion_failure)
    */
    /*@CTK "decreaseApproval correctness case 2"
      @pre allowed[msg.sender][_spender] >= _subtractedValue
      @post __post.allowed[msg.sender][_spender] == allowed[msg.sender][_spender] -
          _subtractedValue
      @post __return == true
      @post (!__has_assertion_failure)
    */
    function decreaseApproval(address _spender, uint _subtractedValue) public returns
        (bool) {
        uint oldValue = allowed[msg.sender][_spender];
        if (_subtractedValue > oldValue) {
            allowed[msg.sender][_spender] = 0;
        } else {
            allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
        }
        Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }
}
```

File lib/SafeMath.sol

```
pragma solidity 0.4.18;


/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {

    //@CTK NO_BUF_OVERFLOW
    //@CTK FAIL NO_ASF
    /*@CTK "SafeMath mul"
      @post (a > 0) && (((a * b) / a) != b) -> (__has_assertion_failure)
      @post (__has_assertion_failure) -> (a > 0) && (((a * b) / a) != b)
      @post __has_assertion_failure == __reverted
      @post !__reverted -> __return == a * b
      @post !__reverted == !__has_overflow
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a * b;
        assert(a == 0 || c / a == b);
        return c;
    }

    //@CTK NO_BUF_OVERFLOW
    //@CTK FAIL NO_ASF
    /*@CTK "SafeMath div"
      @post b != 0 -> !__reverted
      @post !__reverted -> __return == a / b
      @post !__reverted -> !__has_overflow
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // assert(b > 0); // Solidity automatically throws when dividing by 0
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold
        return c;
    }

    //@CTK NO_BUF_OVERFLOW
    //@CTK FAIL NO_ASF
    /*@CTK "SafeMath sub"
      @post (a < b) == __reverted
      @post !__reverted -> __return == a - b
      @post !__reverted -> !__has_overflow
    */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        assert(b <= a);
        return a - b;
    }

    //@CTK NO_BUF_OVERFLOW
    //@CTK FAIL NO_ASF
    /*@CTK "SafeMath add"
      @post (a + b < a || a + b < b) == __reverted
      @post !__reverted -> __return == a + b
      @post !__reverted -> !__has_overflow
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        assert(c >= a);
        return c;
    }
}
```

How to read

Detail for Request 1

transferFrom to same address

Verification date	20, Oct 2018
Verification timespan	395.38 ms
CERTIK *label location*	Line 30-34 in File howtoread.sol

CERTIK *label*

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

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

Raw code

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

Counterexample	✗ This code violates the specification

Initial environment

```
1   Counter Example:
2   Before Execution:
3       Input = {
4           from = 0x0
5           to = 0x0
6           tokens = 0x6c
7       }
8       This = 0

53              balance: 0x0
54          }
55      }
56
```

Post environment

```
57  After Execution:
58      Input = {
59          from = 0x0
60          to = 0x0
61          tokens = 0x6c
```

Static Analysis Request

INSECURE_COMPILER_VERSION

Line 1 in File Telcoin.sol

```
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

```
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

```
//@CTK NO_OVERFLOW
```

Line 34-37 in File Telcoin.sol

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

✓ 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

```
//@CTK NO_BUF_OVERFLOW
```

Line 34-37 in File Telcoin.sol

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

✓ 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

```
//@CTK NO_ASF
```

Line 34-37 in File Telcoin.sol

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

✓ The code meets the specification

Formal Verification Request 4

Telcoin constructor

24, Nov 2018
3.66 ms

Line 31-33 in File Telcoin.sol

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

Line 34-37 in File Telcoin.sol

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

✓ 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

```
//@CTK NO_OVERFLOW
```

Line 48-50 in File Telcoin.sol

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

✓ 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

```
//@CTK NO_BUF_OVERFLOW
```

Line 48-50 in File Telcoin.sol

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

✓ 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

```
//@CTK NO_ASF
```

Line 48-50 in File Telcoin.sol

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

✓ The code meets the specification

Formal Verification Request 8

balanceOf

24, Nov 2018
0.76 ms

Line 43-47 in File Telcoin.sol

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

Line 48-50 in File Telcoin.sol

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

✓ 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

```
    //@CTK NO_OVERFLOW
```

Line 69-78 in File Telcoin.sol

```
    function transfer(address _to, uint256 _value) public returns (bool) {
        require(_to != address(0));
        require(_value <= balances[msg.sender]);

        // SafeMath.sub will throw if there is not enough balance.
        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        Transfer(msg.sender, _to, _value);
        return true;
    }
```

✅ 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

```
    //@CTK NO_BUF_OVERFLOW
```

Line 69-78 in File Telcoin.sol

```
    function transfer(address _to, uint256 _value) public returns (bool) {
        require(_to != address(0));
        require(_value <= balances[msg.sender]);

        // SafeMath.sub will throw if there is not enough balance.
        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        Transfer(msg.sender, _to, _value);
        return true;
    }
```

✅ 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

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

Line 69-78 in File Telcoin.sol

```
    function transfer(address _to, uint256 _value) public returns (bool) {
        require(_to != address(0));
        require(_value <= balances[msg.sender]);

        // SafeMath.sub will throw if there is not enough balance.
        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        Transfer(msg.sender, _to, _value);
        return true;
    }
```

❌ This code violates the specification

```
Counter Example:
Before Execution:
    Input = {
        _to = 32
        _value = 34
    }
    This = 0
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 0
        }
    }
    Other = {
        __return = false
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": 0,
        "value": {
          "contract_name": "Telcoin",
          "balance": 0,
          "contract": {
            "name": "",
            "symbol": "",
            "decimals": 0,
            "totalSupply": 0,
            "balances": [
              {
                "key": 32,
                "value": 252
              },
              {
                "key": 64,
                "value": 0
              },
              {
                "key": 40,
                "value": 0
              },
              {
                "key": 2,
                "value": 16
              },
              {
                "key": 8,
                "value": 32
              },
              {
                "key": "ALL_OTHERS",
                "value": 128
              }
            ],
            "allowed": [
              {
                "key": "ALL_OTHERS",
                "value": [
                  {
                    "key": "ALL_OTHERS",
                    "value": 128
                  }
                ]
              }
            ]
          }
        }
      },
      {
        "key": "ALL_OTHERS",
        "value": "EmptyAddress"
      }
    ]

Function invocation is reverted.
```

Formal Verification Request 12

transfer2_same

📅 24, Nov 2018

⏱ 122.21 ms

Line 56-61 in File Telcoin.sol

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

Line 69-78 in File Telcoin.sol

```
function transfer(address _to, uint256 _value) public returns (bool) {
   require(_to != address(0));
   require(_value <= balances[msg.sender]);

   // SafeMath.sub will throw if there is not enough balance.
   balances[msg.sender] = balances[msg.sender].sub(_value);
   balances[_to] = balances[_to].add(_value);
   Transfer(msg.sender, _to, _value);
   return true;
}
```

✓ The code meets the specification

Formal Verification Request 13

transfer2

📅 24, Nov 2018
⏱ 150.35 ms

Line 62-68 in File Telcoin.sol

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

Line 69-78 in File Telcoin.sol

```
function transfer(address _to, uint256 _value) public returns (bool) {
   require(_to != address(0));
   require(_value <= balances[msg.sender]);

   // SafeMath.sub will throw if there is not enough balance.
   balances[msg.sender] = balances[msg.sender].sub(_value);
   balances[_to] = balances[_to].add(_value);
   Transfer(msg.sender, _to, _value);
   return true;
}
```

✓ 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

```
    //@CTK NO_OVERFLOW
```

Line 97-107 in File Telcoin.sol

```
    function transferFrom(address _from, address _to, uint256 _value) public returns (
        bool) {
        require(_to != address(0));
        require(_value <= balances[_from]);
        require(_value <= allowed[_from][msg.sender]);

        balances[_from] = balances[_from].sub(_value);
        balances[_to] = balances[_to].add(_value);
        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
        Transfer(_from, _to, _value);
        return true;
    }
```

✅ 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

```
    //@CTK NO_BUF_OVERFLOW
```

Line 97-107 in File Telcoin.sol

```
    function transferFrom(address _from, address _to, uint256 _value) public returns (
        bool) {
        require(_to != address(0));
        require(_value <= balances[_from]);
        require(_value <= allowed[_from][msg.sender]);

        balances[_from] = balances[_from].sub(_value);
        balances[_to] = balances[_to].add(_value);
        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
        Transfer(_from, _to, _value);
        return true;
    }
```

✅ 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

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

Line 97-107 in File Telcoin.sol

```
    function transferFrom(address _from, address _to, uint256 _value) public returns (
        bool) {
        require(_to != address(0));
        require(_value <= balances[_from]);
        require(_value <= allowed[_from][msg.sender]);

        balances[_from] = balances[_from].sub(_value);
        balances[_to] = balances[_to].add(_value);
        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
        Transfer(_from, _to, _value);
        return true;
    }
```

✖ This code violates the specification

```
Counter Example:
Before Execution:
    Input = {
        _from = 0
        _to = 32
        _value = 1
    }
    This = 0
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 0
        }
    }
    Other = {
        __return = false
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": 0,
        "value": {
          "contract_name": "Telcoin",
        "balance": 0,
        "contract": {
          "name": "",
          "symbol": "",
          "decimals": 0,
          "totalSupply": 0,
          "balances": [
            {
              "key": 64,
              "value": 0
            },
            {
              "key": 0,
              "value": 64
            },
            {
              "key": 4,
              "value": 64
            },
            {
              "key": 2,
              "value": 0
            },
            {
              "key": 8,
              "value": 0
            },
            {
              "key": 34,
              "value": 0
            },
            {
              "key": 1,
              "value": 0
            },
            {
              "key": 32,
              "value": 255
            },
            {
              "key": "ALL_OTHERS",
              "value": 128
            }
          ],
          "allowed": [
            {
              "key": 0,
              "value": [
                {
                  "key": 0,
                  "value": 128
                },
                {
                  "key": 2,
                  "value": 8
                },
                {
                  "key": "ALL_OTHERS",
                    "value": 1
                  }
                ]
              },
              {
                "key": 16,
                "value": [
                  {
                    "key": 0,
                    "value": 0
                  },
                  {
                    "key": "ALL_OTHERS",
                    "value": 128
                  }
                ]
              },
              {
                "key": "ALL_OTHERS",
                "value": [
                  {
                    "key": "ALL_OTHERS",
                    "value": 128
                  }
                ]
              }
            ]
          }
        }
      },
      {
        "key": "ALL_OTHERS",
        "value": "EmptyAddress"
      }
    ]

Function invocation is reverted.
```

Formal Verification Request 17

transferFrom

📅 24, Nov 2018
⏱ 335.03 ms

Line 84-90 in File Telcoin.sol

```
    /*@CTK "transferFrom"
      @pre (__reverted) == (false)
      @pre (_from) != (_to)
      @post (__return) == (true)
      @post (__post.balances[_to]) == ((balances[_to]) + (_value))
      @post (__post.balances[_from]) == ((balances[_from]) - (_value))
    */
```

Line 97-107 in File Telcoin.sol

```
    function transferFrom(address _from, address _to, uint256 _value) public returns (
        bool) {
        require(_to != address(0));
        require(_value <= balances[_from]);
        require(_value <= allowed[_from][msg.sender]);

        balances[_from] = balances[_from].sub(_value);
        balances[_to] = balances[_to].add(_value);
        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
        Transfer(_from, _to, _value);
        return true;
    }
```

✓ The code meets the specification

Formal Verification Request 18

transferFrom_same

24, Nov 2018
120.0 ms

Line 91-96 in File Telcoin.sol

```
    /*@CTK "transferFrom_same"
      @pre (__reverted) == (false)
      @pre (_from) == (_to)
      @post (__return) == (true)
      @post (__post.balances[_from]) == (balances[_from])
    */
```

Line 97-107 in File Telcoin.sol

```
    function transferFrom(address _from, address _to, uint256 _value) public returns (
        bool) {
        require(_to != address(0));
        require(_value <= balances[_from]);
        require(_value <= allowed[_from][msg.sender]);

        balances[_from] = balances[_from].sub(_value);
        balances[_to] = balances[_to].add(_value);
        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
        Transfer(_from, _to, _value);
        return true;
    }
```

✓ 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

```
    //@CTK NO_OVERFLOW
```

Line 121-125 in File Telcoin.sol

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

✓ 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

```
    //@CTK NO_BUF_OVERFLOW
```

Line 121-125 in File Telcoin.sol

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

✓ 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

```
    //@CTK NO_ASF
```

Line 121-125 in File Telcoin.sol

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

✓ The code meets the specification

Formal Verification Request 22

approve correctness

📅 24, Nov 2018
⏱ 1.96 ms

Line 117-120 in File Telcoin.sol

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

Line 121-125 in File Telcoin.sol

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

✓ 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

```
//@CTK NO_OVERFLOW
```

Line 134-136 in File Telcoin.sol

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

✓ 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

```
//@CTK NO_BUF_OVERFLOW
```

Line 134-136 in File Telcoin.sol

```
    function allowance(address _owner, address _spender) public view returns (uint256)
         {
        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

```
    //@CTK NO_ASF
```

Line 134-136 in File Telcoin.sol

```
    function allowance(address _owner, address _spender) public view returns (uint256)
         {
        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

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

Line 134-136 in File Telcoin.sol

```
    function allowance(address _owner, address _spender) public view returns (uint256)
         {
        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

```
    //@CTK NO_OVERFLOW
```

Line 147-151 in File Telcoin.sol

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

✓ 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

```
    //@CTK NO_BUF_OVERFLOW
```

Line 147-151 in File Telcoin.sol

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

✓ 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

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

Line 147-151 in File Telcoin.sol

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

❌ This code violates the specification

```
Counter Example:
Before Execution:
    Input = {
        _addedValue = 53
        _spender = 0
    }
    This = 0
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 0
        }
    }
    Other = {
        __return = false
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": 0,
        "value": {
          "contract_name": "Telcoin",
          "balance": 0,
          "contract": {
            "name": "",
            "symbol": "",
            "decimals": 0,
            "totalSupply": 0,
            "balances": [
              {
                "key": 16,
                "value": 2
              },
              {
                "key": 128,
                "value": 16
              },
              {
                "key": 0,
                "value": 0
              },
              {
                "key": "ALL_OTHERS",
                "value": 53
              }
            ],
            "allowed": [
              {
                "key": 0,
                "value": [
                  {
                    "key": 128,
                    "value": 0
                  },
                  {
                    "key": 0,
                    "value": 203
                  },
                  {
                    "key": 32,
                    "value": 0
                  },
                  {
                    "key": 2,
                    "value": 0
                  },
                  {
                    "key": "ALL_OTHERS",
                    "value": 53
                  }
                ]
              },
              {
                "key": "ALL_OTHERS",
                "value": [
                  {
                    "key": "ALL_OTHERS",
                    "value": 53
                  }
                ]
              }
            ]
          }
        }
      },
      {
        "key": "ALL_OTHERS",
        "value": "EmptyAddress"
      }
    ]

Function invocation is reverted.
```

Formal Verification Request 30

increaseApproval correctness

📅 24, Nov 2018
⏱ 3.72 ms

Line 142-146 in File Telcoin.sol

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

Line 147-151 in File Telcoin.sol

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

✅ 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

```
    //@CTK NO_OVERFLOW
```

Line 168-177 in File Telcoin.sol

```
    function decreaseApproval(address _spender, uint _subtractedValue) public returns
        (bool) {
        uint oldValue = allowed[msg.sender][_spender];
        if (_subtractedValue > oldValue) {
            allowed[msg.sender][_spender] = 0;
        } else {
            allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
        }
        Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }
```

✅ 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

```
    //@CTK NO_BUF_OVERFLOW
```

Line 168-177 in File Telcoin.sol

```
    function decreaseApproval(address _spender, uint _subtractedValue) public returns
        (bool) {
        uint oldValue = allowed[msg.sender][_spender];
        if (_subtractedValue > oldValue) {
            allowed[msg.sender][_spender] = 0;
        } else {
            allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
        }
        Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }
```

✅ 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

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

Line 168-177 in File Telcoin.sol

```
    function decreaseApproval(address _spender, uint _subtractedValue) public returns
        (bool) {
        uint oldValue = allowed[msg.sender][_spender];
        if (_subtractedValue > oldValue) {
            allowed[msg.sender][_spender] = 0;
        } else {
            allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
        }
        Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }
```

✅ 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

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

Line 168-177 in File Telcoin.sol

```
function decreaseApproval(address _spender, uint _subtractedValue) public returns
    (bool) {
    uint oldValue = allowed[msg.sender][_spender];
    if (_subtractedValue > oldValue) {
        allowed[msg.sender][_spender] = 0;
    } else {
        allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
    }
    Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
}
```

✓ 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

```
//@CTK NO_BUF_OVERFLOW
```

Line 19-23 in File SafeMath.sol

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

✓ 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

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

Line 19-23 in File SafeMath.sol

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

❌ This code violates the specification

```
Counter Example:
Before Execution:
    Input = {
        a = 94
        b = 128
    }
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 0
        }
    }
    Other = {
        __return = 0
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": "ALL_OTHERS",
        "value": "EmptyAddress"
      }
    ]

Function invocation is reverted.
```

Formal Verification Request 37

SafeMath mul

📅 24, Nov 2018
⏱ 183.34 ms

Line 12-18 in File SafeMath.sol

```
    /*@CTK "SafeMath mul"
      @post (a > 0) && (((a * b) / a) != b) -> (__has_assertion_failure)
      @post (__has_assertion_failure) -> (a > 0) && (((a * b) / a) != b)
      @post __has_assertion_failure == __reverted
      @post !__reverted -> __return == a * b
      @post !__reverted == !__has_overflow
     */
```

Line 19-23 in File SafeMath.sol

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

✅ 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

```
    //@CTK NO_BUF_OVERFLOW
```

Line 32-37 in File SafeMath.sol

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

✅ 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

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

Line 32-37 in File SafeMath.sol

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

❌ This code violates the specification

```
Counter Example:
Before Execution:
    Input = {
        a = 0
        b = 0
    }
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 0
        }
    }
    Other = {
        __return = 0
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": "ALL_OTHERS",
        "value": "EmptyAddress"
      }
    ]

Function invocation is reverted.
```

Formal Verification Request 40

SafeMath div

📅 24, Nov 2018
⏱ 2.43 ms

Line 27-31 in File SafeMath.sol

```
    /*@CTK "SafeMath div"
      @post b != 0 -> !__reverted
      @post !__reverted -> __return == a / b
      @post !__reverted -> !__has_overflow
     */
```

Line 32-37 in File SafeMath.sol

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

✓ 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

```
    //@CTK NO_BUF_OVERFLOW
```

Line 46-49 in File SafeMath.sol

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

✓ 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

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

Line 46-49 in File SafeMath.sol

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

✗ This code violates the specification

```
Counter Example:
Before Execution:
    Input = {
        a = 0
        b = 1
    }
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 0
        }
    }
    Other = {
        __return = 0
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": "ALL_OTHERS",
        "value": "EmptyAddress"
      }
    ]

Function invocation is reverted.
```

Formal Verification Request 43

SafeMath sub

📅 24, Nov 2018
⏱️ 3.1 ms

Line 41-45 in File SafeMath.sol

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

Line 46-49 in File SafeMath.sol

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

✅ 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

```
    //@CTK NO_BUF_OVERFLOW
```

Line 58-62 in File SafeMath.sol

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

✓ 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

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

Line 58-62 in File SafeMath.sol

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

✗ This code violates the specification

```
Counter Example:
Before Execution:
    Input = {
        a = 83
        b = 173
    }
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 0
        }
    }
    Other = {
        __return = 0
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": "ALL_OTHERS",
        "value": "EmptyAddress"
      }
    ]

Function invocation is reverted.
```

Formal Verification Request 46

SafeMath add

📅 24, Nov 2018
⏱ 6.15 ms

Line 53-57 in File SafeMath.sol

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

Line 58-62 in File SafeMath.sol

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

✓ The code meets the specification