# CertiK Verification Report For UULA

Request Date: 2019-01-16
Revision Date: 2019-01-21

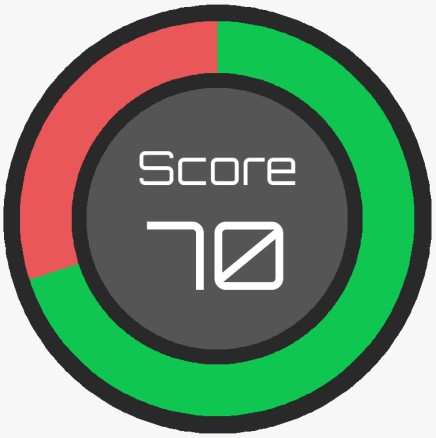
# Disclaimer

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

# WARNING

CERTIK *identified some potential security flaws in this contract and also provided corresponding solutions.*

*Jan 21, 2019*




## Summary

This is the report for smart contract verification service requestd by UULA. 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. | 1 | SWC-101 |
| Function incorrectness | Function implementation does not meet the specification, leading to intentional or unintentional vulnerabilities. | 0 | |
| Buffer Overflow | An attacker is able to write to arbitrary storage locations of a contract if array of out bound happens | 0 | SWC-124 |
| Reentrancy | A malicious contract can call back into the calling contract before the first invocation of the function is finished. | 0 | SWC-107 |
| Transaction Order Dependence | A race condition vulnerability occurs when code depends on the order of the transactions submitted to it. | 0 | SWC-114 |
| Timestamp Dependence | Timestamp can be influenced by minors to some degree. | 0 | SWC-116 |

| | | | |
|---|---|---|---|
| Insecure Compiler Version | Using an fixed outdated compiler version or floating pragma can be problematic, if there are publicly disclosed bugs and issues that affect the current compiler version used. | 0 | SWC-102<br>SWC-103 |
| Insecure Randomness | Block attributes are insecure to generate random numbers, as they can be influenced by minors to some degree. | 0 | SWC-120 |
| "tx.origin" for authorization | tx.origin should not be used for authorization. Use msg.sender instead. | 0 | SWC-115 |
| Delegatecall to Untrusted Callee | Calling into untrusted contracts is very dangerous, the target and arguments provided must be sanitized. | 0 | SWC-112 |
| State Variable Default Visibility | Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable. | 0 | SWC-108 |
| Function Default Visibility | Functions are public by default. A malicious user is able to make unauthorized or unintended state changes if a developer forgot to set the visibility. | 0 | SWC-100 |
| Uninitialized variables | Uninitialized local storage variables can point to other unexpected storage variables in the contract. | 0 | SWC-109 |
| Assertion Failure | The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement. | 0 | SWC-110 |
| Deprecated Solidity Features | Several functions and operators in Solidity are deprecated and should not be used as best practice. | 4 | SWC-111 |
| Unused variables | Unused variables reduce code quality | 0 | |

# Vulnerability Details

## Critical

No issue found.

## Medium

### TransferFrom condition

One of the conditon in which `transferFrom` fails is `balances[_from] < _value` and `balances[_to] + _value >= balances[_to]`. This basically means that if a malicious message sender is given a big enough allowance, he might be able to use this allowance to overflow other user's account and make them lower.

Specifically, one way to attack this is to setup two accounts, one as the `from` and the other one as the message sender. For any other account `to`, message sender can execute these two steps to reduce the balance of any other account:

1. Grant a big enough allowance from `from` to message sender by using `approve`
2. Message sender invokes `transferFrom(from, to)` to send an extraordinary huge amount from `from` to `to`, and overflows the balance of the `to` account. As a result, now `to` has a even smaller balance. This can be used to zero out any account.

This smart contract can executes there steps without failures. A fix is to simply change the condition to fail when `balances[_from] < _value` **OR** `balances[_to] + _value < balances[_to]`

## Low

### Use SafeMath if possible

There are extensive condition checks scattered in a couple of places from the codes to make sure arithmetic does not overflow. It is easier and less error prone to use the SafeMath library.

### Deprecated Syntax

1. Use `emit` to trigger an event
2. Use keyword `constructor` as the constructor.

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 uula.sol

```
pragma solidity ^0.4.13;

contract Ownable {
    address public owner;
    /*@CTK ownable
      @post __post.owner == msg.sender
     */
    function Ownable() public {
        owner = msg.sender;
    }
    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }
    /*@CTK transferOwnership
      @tag assume_completion
      @post owner == msg.sender
      @post __post.owner == newOwner
     */
    function transferOwnership(address newOwner) onlyOwner public {
        owner = newOwner;
    }
}

contract UULATokenCoin is Ownable {
    string public constant name = "Uulala";
    string public constant symbol = "UULA";
    uint32 public constant decimals = 4;
    uint public constant INITIAL_SUPPLY = 7500000000000;
    uint public totalSupply = 0;
    mapping (address => uint) balances;
    mapping (address => mapping(address => uint)) allowed;

    /*@CTK UULATokenCoin
      @pre INITIAL_SUPPLY == 100
      @post __post.totalSupply == INITIAL_SUPPLY
      @post __post.balances[msg.sender] == INITIAL_SUPPLY
     */
    function UULATokenCoin () public {
        totalSupply = INITIAL_SUPPLY;
        balances[msg.sender] = INITIAL_SUPPLY;
    }

    /*@CTK balanceOf
      @post balance == balances[_owner]
     */
    function balanceOf(address _owner) public constant returns (uint balance) {
        return balances[_owner];
    }

    /*@CTK transfer
      @pre success == true
      @pre msg.sender != _to
      @post __post.balances[msg.sender] == balances[msg.sender] - _value
```

```
    @post __post.balances[_to] == balances[_to] + _value
   */
  function transfer(address _to, uint _value) public returns (bool success) {
      if (balances[msg.sender] < _value || balances[msg.sender] + _value < balances[msg.sender]) {
          return false;
      }

      balances[msg.sender] -= _value;
      balances[_to] += _value;
      Transfer(msg.sender, _to, _value);

      return true;
  }

  // Buggy: malicious user can sabortage other users.
  /*@CTK transferFrom
    @pre success == true
    @pre _from != _to
    @post allowed[_from][msg.sender] >= _value
    @post balances[_from] >= _value || balances[_to] + _value < balances[_to]
    @post __post.allowed[_from][msg.sender] == allowed[_from][msg.sender] - _value
    @post __post.balances[_from] == balances[_from] - _value
    @post __post.balances[_to] == balances[_to] + _value
   */
  function transferFrom(address _from, address _to, uint _value) public returns (bool success) {
      if (allowed[_from][msg.sender] < _value || balances[_from] < _value && balances[_to] + _value >= balances[_to]) {
          return false;
      }

      allowed[_from][msg.sender] -= _value;
      balances[_from] -= _value;
      balances[_to] += _value;
      Transfer(_from, _to, _value);

      return true;
  }

  /*@CTK approve
    @post __post.allowed[msg.sender][_spender] == _value
   */
  function approve(address _spender, uint _value) public returns (bool success) {
      allowed[msg.sender][_spender] = _value;
      Approval(msg.sender, _spender, _value);

      return true;
  }

  /*@CTK allowance
    @post remaining == allowed[_owner][_spender]
   */
  function allowance(address _owner, address _spender) public constant returns (uint remaining) {
      return allowed[_owner][_spender];
  }

```

```
    /*CTK drop
      @tag assume_completion
      @post recipients.length == values.length
     */
    function drop(address[] recipients, uint256[] values) public {
        require(recipients.length == values.length);

        uint256 sum = 0;
        uint256 i = 0;

        /*CTK ForLoop_Sum
          @inv sum == sum__pre + values[i]
          @inv i <= recipients.length
          @post i == recipients.length
          @post !__should_return
         */
        for (i = 0; i < recipients.length; i++) {
            sum += values[i];
        }
        require(sum <= balances[msg.sender]);

        /*CTK ForLoop_Transfer
          @inv i <= recipients.length
          @post i == recipients.length
          @post !__should_return
         */
        for (i = 0; i < recipients.length; i++) {
            transfer(recipients[i], values[i]);
        }
    }

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

# 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 uula.sol

```
1 pragma solidity ^0.4.13;
```

Only these compiler versions are safe to compile your code: 0.4.25

# Formal Verification Request 1

### ownable

📅 21, Jan 2019
⏱ 6.51 ms

Line 5-7 in File uula.sol

```
5   /*@CTK ownable
6     @post __post.owner == msg.sender
7    */
```

Line 8-10 in File uula.sol

```
8   function Ownable() public {
9       owner = msg.sender;
10  }
```

✅ The code meets the specification

# Formal Verification Request 2

### transferOwnership

📅 21, Jan 2019
⏱ 15.41 ms

Line 15-19 in File uula.sol

```
15  /*@CTK transferOwnership
16    @tag assume_completion
17    @post owner == msg.sender
18    @post __post.owner == newOwner
19   */
```

Line 20-22 in File uula.sol

```
20  function transferOwnership(address newOwner) onlyOwner public {
21      owner = newOwner;
22  }
```

✅ The code meets the specification

# Formal Verification Request 3

### UULATokenCoin

📅 21, Jan 2019
⏱ 13.15 ms

Line 34-38 in File uula.sol

```
34  /*@CTK UULATokenCoin
35    @pre INITIAL_SUPPLY == 100
36    @post __post.totalSupply == INITIAL_SUPPLY
37    @post __post.balances[msg.sender] == INITIAL_SUPPLY
38   */
```

Line 39-42 in File uula.sol

```
39  function UULATokenCoin () public {
40      totalSupply = INITIAL_SUPPLY;
41      balances[msg.sender] = INITIAL_SUPPLY;
42  }
```

The code meets the specification

# Formal Verification Request 4

### balanceOf

21, Jan 2019
6.03 ms

Line 44-46 in File uula.sol

```
44  /*@CTK balanceOf
45    @post balance == balances[_owner]
46   */
```

Line 47-49 in File uula.sol

```
47  function balanceOf(address _owner) public constant returns (uint balance) {
48      return balances[_owner];
49  }
```

The code meets the specification

# Formal Verification Request 5

### transfer

21, Jan 2019
63.22 ms

Line 51-56 in File uula.sol

```
51  /*@CTK transfer
52    @pre success == true
53    @pre msg.sender != _to
54    @post __post.balances[msg.sender] == balances[msg.sender] - _value
55    @post __post.balances[_to] == balances[_to] + _value
56   */
```

Line 57-67 in File uula.sol

```
function transfer(address _to, uint _value) public returns (bool success) {
    if (balances[msg.sender] < _value || balances[msg.sender] + _value < balances[msg.sender]) {
        return false;
    }

    balances[msg.sender] -= _value;
    balances[_to] += _value;
    Transfer(msg.sender, _to, _value);

    return true;
}
```

The code meets the specification

# Formal Verification Request 6

### transferFrom

21, Jan 2019
115.37 ms

Line 70-78 in File uula.sol

```
/*@CTK transferFrom
  @pre success == true
  @pre _from != _to
  @post allowed[_from][msg.sender] >= _value
  @post balances[_from] >= _value || balances[_to] + _value < balances[_to]
  @post __post.allowed[_from][msg.sender] == allowed[_from][msg.sender] - _value
  @post __post.balances[_from] == balances[_from] - _value
  @post __post.balances[_to] == balances[_to] + _value
 */
```

Line 79-90 in File uula.sol

```
function transferFrom(address _from, address _to, uint _value) public returns (bool success) {
    if (allowed[_from][msg.sender] < _value || balances[_from] < _value && balances[_to] + _value >= balances[_to]) {
        return false;
    }

    allowed[_from][msg.sender] -= _value;
    balances[_from] -= _value;
    balances[_to] += _value;
    Transfer(_from, _to, _value);

    return true;
}
```

The code meets the specification

# Formal Verification Request 7

### approve

21, Jan 2019
12.55 ms

Line 92-94 in File uula.sol

```
92    /*@CTK approve
93      @post __post.allowed[msg.sender][_spender] == _value
94     */
```

Line 95-100 in File uula.sol

```
95    function approve(address _spender, uint _value) public returns (bool success) {
96        allowed[msg.sender][_spender] = _value;
97        Approval(msg.sender, _spender, _value);
98
99        return true;
100   }
```

The code meets the specification

# Formal Verification Request 8

### allowance

21, Jan 2019
5.85 ms

Line 102-104 in File uula.sol

```
102   /*@CTK allowance
103     @post remaining == allowed[_owner][_spender]
104    */
```

Line 105-107 in File uula.sol

```
105   function allowance(address _owner, address _spender) public constant returns (uint
          remaining) {
106       return allowed[_owner][_spender];
107   }
```

The code meets the specification