CertiK Verification Report For Fleta Token

Request Date: 2019-03-21
Revision Date: 2019-04-08

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 Fleta Token(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.

Summary

This audit report summarises the smart contract verification service requested by Fleta Token. The goal of this security audit is to guarantee that the audited smart contracts are robust enough to avoid any potential security 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 time of the audit.

Type of Issues

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

Title	Description	Issues	SWC ID
Integer Overflow and Underflow	An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function incorrectness	Function implementation does not meet the specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker is able to write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by minors to some degree.	0	SWC-116

Insecure Compiler Version	Using an fixed outdated compiler version or floating pragma can be problematic, if there are publicly disclosed bugs and issues that affect the current compiler version used.	0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

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

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 fleta.sol

```
pragma solidity ^0.5.0;

// ----------------------------------------------------------------------------
// 'FLETA' 'Fleta Token' token contract
//
// Symbol  : FLETA
// Name  : Fleta Token
// Total supply: 2,000,000,000.000000000000000000
// Decimals : 18
//
// Enjoy.
//
// (c) Sam Jeong / Firstchain Co. 2018. The MIT Licence.
// ----------------------------------------------------------------------------


// ----------------------------------------------------------------------------
// Safe maths
// ----------------------------------------------------------------------------
library SafeMath {
  /*@CTK SafeMath_add
      @tag spec
      @post __reverted == false -> c == a + b
      @post msg == msg__post
      @post (a + b < a) == __reverted
      @post __addr_map == __addr_map__post
      @post !(__has_buf_overflow)
      @post !(__has_assertion_failure)
      @post (__reverted) == (__has_overflow)
    */
  function add(uint a, uint b) internal pure returns (uint c) {
    c = a + b;
    require(c >= a);
  }
  /*@CTK SafeMath_sub
      @tag spec
      @post __has_overflow == true -> b > a
      @post __reverted == false -> c == a - b
      @post msg == msg__post
      @post (a < b) == __reverted
      @post __addr_map == __addr_map__post
      @post !(__has_buf_overflow)
      @post !(__has_assertion_failure)
    */
  function sub(uint a, uint b) internal pure returns (uint c) {
    require(b <= a);
    c = a - b;
  }
  /*@CTK SafeMath_mul
      @tag spec
      @post __reverted == __has_overflow
      @post __reverted == false -> c == a * b
      @post a == 0 -> c == 0
      @post msg == msg__post
```

```
    @post (a > 0 && (a * b / a != b)) == __reverted
    @post __addr_map == __addr_map__post
    @post !(__has_buf_overflow)
    @post !(__has_assertion_failure)
  */
  function mul(uint a, uint b) internal pure returns (uint c) {
    c = a * b;
    require(a == 0 || c / a == b);
  }
  /*@CTK SafeMath_div
    @tag spec
    @post __reverted == (b <= 0)
    @post b == 0 -> __reverted == true // solidity throws on 0.
    @post __reverted == false -> c == a / b
    @post msg == msg__post
    @post __addr_map == __addr_map__post
    @post !(__has_buf_overflow)
    @post !(__has_assertion_failure)
  */
  function div(uint a, uint b) internal pure returns (uint c) {
    require(b > 0);
    c = a / b;
  }
}


// ----------------------------------------------------------------------------
// ERC Token Standard #20 Interface
// https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
// ----------------------------------------------------------------------------
contract ERC20Interface {
  function totalSupply() public view returns (uint);
  function balanceOf(address tokenOwner) public view returns (uint balance);
  function allowance(address tokenOwner, address spender) public view returns (uint
      remaining);
  function transfer(address to, uint tokens) public returns (bool success);
  function approve(address spender, uint tokens) public returns (bool success);
  function transferFrom(address from, address to, uint tokens) public returns (bool
      success);

  event Transfer(address indexed from, address indexed to, uint tokens);
  event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}


// ----------------------------------------------------------------------------
// Contract function to receive approval and execute function in one call
//
// Borrowed from MiniMeToken
// ----------------------------------------------------------------------------
contract ApproveAndCallFallBack {
  function receiveApproval(address from, uint256 tokens, address token, bytes memory
      data) public;
}


// ----------------------------------------------------------------------------
// Owned contract
```

```
// ----------------------------------------------------------------------------
contract Owned {
  address public owner;

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK Ownable
      @post __post.owner == msg.sender
    */
  constructor() public {
    owner = msg.sender;
  }

  modifier onlyOwner {
    require(msg.sender == owner);
    _;
  }
}


// ----------------------------------------------------------------------------
// ERC20 Token, with the addition of symbol, name and decimals and a
// fixed supply
// ----------------------------------------------------------------------------
contract FletaToken is ERC20Interface, Owned {
  using SafeMath for uint;

  string public symbol;
  string public name;
  uint8 public decimals;
  uint _totalSupply;
  bool _stopTrade;

  mapping(address => uint) balances;
  mapping(address => mapping(address => uint)) allowed;


  // ------------------------------------------------------------------------
  // Constructor
  // ------------------------------------------------------------------------
  constructor() public {
    symbol = "FLETA";
    name = "Fleta Token";
    decimals = 18;
    _totalSupply = 2000000000 * 10**uint(decimals);
    _stopTrade = false;
    balances[owner] = _totalSupply;
    emit Transfer(address(0), owner, _totalSupply);
  }


  // ------------------------------------------------------------------------
  // Total supply
  // ------------------------------------------------------------------------
  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
```

```
/*@CTK totalSupply
    @tag assume_completion
    @post __return == _totalSupply - balances[address(0)]
  */
function totalSupply() public view returns (uint) {
  return _totalSupply.sub(balances[address(0)]);
}


// ----------------------------------------------------------------------
// Stop Trade
// ----------------------------------------------------------------------
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK stopTrade
  @post _stopTrade == true -> __reverted
  @post msg.sender != owner -> __reverted
  @post _stopTrade != true && msg.sender == owner -> __post._stopTrade == true
  */
function stopTrade() public onlyOwner {
  require(_stopTrade != true);
  _stopTrade = true;
}


// ----------------------------------------------------------------------
// Start Trade
// ----------------------------------------------------------------------
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK startTrade
  @post _stopTrade != true -> __reverted
  @post msg.sender != owner -> __reverted
  @post _stopTrade == true && msg.sender == owner -> __post._stopTrade != true
  */
function startTrade() public onlyOwner {
  require(_stopTrade == true);
  _stopTrade = false;
}


// ----------------------------------------------------------------------
// Get the token balance for account `tokenOwner`
// ----------------------------------------------------------------------
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK balanceOf
    @post balance == balances[tokenOwner]
  */
function balanceOf(address tokenOwner) public view returns (uint balance) {
  return balances[tokenOwner];
}


// ----------------------------------------------------------------------
```

```
// Transfer the balance from token owner's account to `to` account
// - Owner's account must have sufficient balance to transfer
// - 0 value transfers are allowed
// ------------------------------------------------------------------------
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK transfer
  @tag assume_completion
    @pre to != msg.sender
    @post __post.balances[msg.sender] == balances[msg.sender] - tokens
    @post __post.balances[to] == balances[to] + tokens
  */
/*@CTK transfer_same
  @tag assume_completion
    @pre to == msg.sender
    @post __post.balances[msg.sender] == balances[msg.sender]
  */
function transfer(address to, uint tokens) public returns (bool success) {
  require(_stopTrade != true);
  require(to > address(0));

  balances[msg.sender] = balances[msg.sender].sub(tokens);
  balances[to] = balances[to].add(tokens);
  emit Transfer(msg.sender, to, tokens);
  return true;
}


// ------------------------------------------------------------------------
// Token owner can approve for `spender` to transferFrom(...) `tokens`
// from the token owner's account
//
// https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md
// recommends that there are no checks for the approval double-spend attack
// as this should be implemented in user interfaces
// ------------------------------------------------------------------------
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK approve
  @pre _stopTrade != true
    @post __post.allowed[msg.sender][spender] == tokens
  */
function approve(address spender, uint tokens) public returns (bool success) {
  require(_stopTrade != true);

  allowed[msg.sender][spender] = tokens;
  emit Approval(msg.sender, spender, tokens);
  return true;
}


// ------------------------------------------------------------------------
// Transfer `tokens` from the `from` account to the `to` account
//
// The calling account must already have sufficient tokens approve(...)-d
// for spending from the `from` account and
```

```
// - From account must have sufficient balance to transfer
// - Spender must have sufficient allowance to transfer
// - 0 value transfers are allowed
// ----------------------------------------------------------------------
  //@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK transferFrom
    @tag assume_completion
    @pre from != to
    @pre from != msg.sender
    @post __post.balances[from] == balances[from] - tokens
    @post __post.allowed[from][msg.sender] == allowed[from][msg.sender] - tokens
    @post __post.balances[to] == balances[to] + tokens
  */
  /*@CTK "transferFrom_sameAddress"
    @tag assume_completion
    @pre from == to
    @post __post.allowed[from][msg.sender] == allowed[from][msg.sender]
    @post __post.balances[to] == balances[to]
  */
/*@CTK "transferFrom_sameAddress2"
    @tag assume_completion
    @pre from != to
    @pre from == msg.sender
    @post __post.allowed[from][msg.sender] == allowed[from][msg.sender]
    @post __post.balances[from] == balances[from] - tokens
    @post __post.balances[to] == balances[to] + tokens
  */
function transferFrom(address from, address to, uint tokens) public returns (bool
    success) {
  require(_stopTrade != true);
  require(from > address(0));
  require(to > address(0));

  balances[from] = balances[from].sub(tokens);
  if(from != to && from != msg.sender) {
    allowed[from][msg.sender] = allowed[from][msg.sender].sub(tokens);
  }
  balances[to] = balances[to].add(tokens);
  emit Transfer(from, to, tokens);
  return true;
}


// ----------------------------------------------------------------------
// Returns the amount of tokens approved by the owner that can be
// transferred to the spender's account
// ----------------------------------------------------------------------
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK allowance
  @pre _stopTrade != true
    @post remaining == allowed[tokenOwner][spender]
  */
function allowance(address tokenOwner, address spender) public view returns (uint
    remaining) {
```

```
    require(_stopTrade != true);

    return allowed[tokenOwner][spender];
  }


  // ----------------------------------------------------------------------
  // Token owner can approve for ‘spender‘ to transferFrom(...) ‘tokens‘
  // from the token owner’s account. The ‘spender‘ contract function
  // ‘receiveApproval(...)‘ is then executed
  // ----------------------------------------------------------------------
  function approveAndCall(address spender, uint tokens, bytes memory data) public
      returns (bool success) {
    require(msg.sender != spender);

    allowed[msg.sender][spender] = tokens;
    emit Approval(msg.sender, spender, tokens);
    ApproveAndCallFallBack(spender).receiveApproval(msg.sender, tokens, address(this),
        data);
    return true;
  }


  // ----------------------------------------------------------------------
  // Don’t accept ETH
  // ----------------------------------------------------------------------
  function () external payable {
    revert();
  }


  // ----------------------------------------------------------------------
  // Owner can transfer out any accidentally sent ERC20 tokens
  // ----------------------------------------------------------------------
  function transferAnyERC20Token(address tokenAddress, uint tokens) public onlyOwner
      returns (bool success) {
    return ERC20Interface(tokenAddress).transfer(owner, tokens);
  }
}
```

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 fleta.sol

```
1 pragma solidity ^0.5.0;
```

Only these compiler versions are safe to compile your code: 0.5.0, 0.5.1, 0.5.2

Formal Verification Request 1

SafeMath_add

08, Apr 2019
24.18 ms

Line 21-30 in File fleta.sol

```
21 /*@CTK SafeMath_add
22     @tag spec
23     @post __reverted == false -> c == a + b
24     @post msg == msg__post
25     @post (a + b < a) == __reverted
26     @post __addr_map == __addr_map__post
27     @post !(__has_buf_overflow)
28     @post !(__has_assertion_failure)
29     @post (__reverted) == (__has_overflow)
30   */
```

Line 31-34 in File fleta.sol

```
31 function add(uint a, uint b) internal pure returns (uint c) {
32   c = a + b;
33   require(c >= a);
34 }
```

The code meets the specification

Formal Verification Request 2

SafeMath_sub

08, Apr 2019
18.63 ms

Line 35-44 in File fleta.sol

```
35 /*@CTK SafeMath_sub
36     @tag spec
37     @post __has_overflow == true -> b > a
38     @post __reverted == false -> c == a - b
39     @post msg == msg__post
40     @post (a < b) == __reverted
41     @post __addr_map == __addr_map__post
42     @post !(__has_buf_overflow)
43     @post !(__has_assertion_failure)
44   */
```

Line 45-48 in File fleta.sol

```
45 function sub(uint a, uint b) internal pure returns (uint c) {
46   require(b <= a);
47   c = a - b;
48 }
```

The code meets the specification

Formal Verification Request 3

SafeMath_mul

08, Apr 2019
130.95 ms

Line 49-59 in File fleta.sol

```
49 /*@CTK SafeMath_mul
50     @tag spec
51     @post __reverted == __has_overflow
52     @post __reverted == false -> c == a * b
53     @post a == 0 -> c == 0
54     @post msg == msg__post
55     @post (a > 0 && (a * b / a != b)) == __reverted
56     @post __addr_map == __addr_map__post
57     @post !(__has_buf_overflow)
58     @post !(__has_assertion_failure)
59   */
```

Line 60-63 in File fleta.sol

```
60 function mul(uint a, uint b) internal pure returns (uint c) {
61   c = a * b;
62   require(a == 0 || c / a == b);
63 }
```

The code meets the specification

Formal Verification Request 4

SafeMath_div

08, Apr 2019
17.9 ms

Line 64-73 in File fleta.sol

```
64 /*@CTK SafeMath_div
65     @tag spec
66     @post __reverted == (b <= 0)
67     @post b == 0 -> __reverted == true // solidity throws on 0.
68     @post __reverted == false -> c == a / b
69     @post msg == msg__post
70     @post __addr_map == __addr_map__post
71     @post !(__has_buf_overflow)
72     @post !(__has_assertion_failure)
73   */
```

Line 74-77 in File fleta.sol

```
74 function div(uint a, uint b) internal pure returns (uint c) {
75   require(b > 0);
76   c = a / b;
77 }
```

The code meets the specification

Formal Verification Request 5

If method completes, integer overflow would not happen.

08, Apr 2019
7.54 ms

Line 114 in File fleta.sol

```
114 //@CTK NO_OVERFLOW
```

Line 120-122 in File fleta.sol

```
120 constructor() public {
121   owner = msg.sender;
122 }
```

The code meets the specification

Formal Verification Request 6

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

08, Apr 2019
0.47 ms

Line 115 in File fleta.sol

```
115 //@CTK NO_BUF_OVERFLOW
```

Line 120-122 in File fleta.sol

```
120 constructor() public {
121   owner = msg.sender;
122 }
```

The code meets the specification

Formal Verification Request 7

Method will not encounter an assertion failure.

08, Apr 2019
0.43 ms

Line 116 in File fleta.sol

```
116 //@CTK NO_ASF
```

Line 120-122 in File fleta.sol

```
120 constructor() public {
121   owner = msg.sender;
122 }
```

The code meets the specification

Formal Verification Request 8

Ownable

08, Apr 2019
1.09 ms

Line 117-119 in File fleta.sol

```
117 /*@CTK Ownable
118     @post __post.owner == msg.sender
119   */
```

Line 120-122 in File fleta.sol

```
120 constructor() public {
121   owner = msg.sender;
122 }
```

The code meets the specification

Formal Verification Request 9

If method completes, integer overflow would not happen.

08, Apr 2019
21.86 ms

Line 165 in File fleta.sol

```
165 //@CTK NO_OVERFLOW
```

Line 172-174 in File fleta.sol

```
172 function totalSupply() public view returns (uint) {
173   return _totalSupply.sub(balances[address(0)]);
174 }
```

The code meets the specification

Formal Verification Request 10

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

08, Apr 2019
0.86 ms

Line 166 in File fleta.sol

```
166 //@CTK NO_BUF_OVERFLOW
```

Line 172-174 in File fleta.sol

```
172 function totalSupply() public view returns (uint) {
173   return _totalSupply.sub(balances[address(0)]);
174 }
```

The code meets the specification

Formal Verification Request 11

Method will not encounter an assertion failure.

08, Apr 2019
0.7 ms

Line 167 in File fleta.sol

```
167 //@CTK NO_ASF
```

Line 172-174 in File fleta.sol

```
172 function totalSupply() public view returns (uint) {
173   return _totalSupply.sub(balances[address(0)]);
174 }
```

The code meets the specification

Formal Verification Request 12

totalSupply

08, Apr 2019
1.21 ms

Line 168-171 in File fleta.sol

```
168 /*@CTK totalSupply
169     @tag assume_completion
170     @post __return == _totalSupply - balances[address(0)]
171   */
```

Line 172-174 in File fleta.sol

```
172 function totalSupply() public view returns (uint) {
173   return _totalSupply.sub(balances[address(0)]);
174 }
```

The code meets the specification

Formal Verification Request 13

If method completes, integer overflow would not happen.

08, Apr 2019
25.81 ms

Line 180 in File fleta.sol

```
180 //@CTK NO_OVERFLOW
```

Line 188-191 in File fleta.sol

```
188 function stopTrade() public onlyOwner {
189   require(_stopTrade != true);
190   _stopTrade = true;
191 }
```

The code meets the specification

Formal Verification Request 14

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

08, Apr 2019
0.68 ms

Line 181 in File fleta.sol

```
181 //@CTK NO_BUF_OVERFLOW
```

Line 188-191 in File fleta.sol

```
188 function stopTrade() public onlyOwner {
189   require(_stopTrade != true);
190   _stopTrade = true;
191 }
```

The code meets the specification

Formal Verification Request 15

Method will not encounter an assertion failure.

08, Apr 2019
0.67 ms

Line 182 in File fleta.sol

```
182 //@CTK NO_ASF
```

Line 188-191 in File fleta.sol

```
188 function stopTrade() public onlyOwner {
189   require(_stopTrade != true);
190   _stopTrade = true;
191 }
```

The code meets the specification

Formal Verification Request 16

stopTrade

08, Apr 2019
3.2 ms

Line 183-187 in File fleta.sol

```
183 /*@CTK stopTrade
184   @post _stopTrade == true -> __reverted
185   @post msg.sender != owner -> __reverted
186   @post _stopTrade != true && msg.sender == owner -> __post._stopTrade == true
187   */
```

Line 188-191 in File fleta.sol

```
188 function stopTrade() public onlyOwner {
189   require(_stopTrade != true);
190   _stopTrade = true;
191 }
```

✅ The code meets the specification

Formal Verification Request 17

If method completes, integer overflow would not happen.

📅 08, Apr 2019
⏱ 27.76 ms

Line 197 in File fleta.sol

```
197 //@CTK NO_OVERFLOW
```

Line 205-208 in File fleta.sol

```
205 function startTrade() public onlyOwner {
206   require(_stopTrade == true);
207   _stopTrade = false;
208 }
```

✅ The code meets the specification

Formal Verification Request 18

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

📅 08, Apr 2019
⏱ 0.63 ms

Line 198 in File fleta.sol

```
198 //@CTK NO_BUF_OVERFLOW
```

Line 205-208 in File fleta.sol

```
205 function startTrade() public onlyOwner {
206   require(_stopTrade == true);
207   _stopTrade = false;
208 }
```

✅ The code meets the specification

Formal Verification Request 19

Method will not encounter an assertion failure.

08, Apr 2019
0.64 ms

Line 199 in File fleta.sol

```
199 //@CTK NO_ASF
```

Line 205-208 in File fleta.sol

```
205 function startTrade() public onlyOwner {
206   require(_stopTrade == true);
207   _stopTrade = false;
208 }
```

The code meets the specification

Formal Verification Request 20

startTrade

08, Apr 2019
4.08 ms

Line 200-204 in File fleta.sol

```
200 /*@CTK startTrade
201   @post _stopTrade != true -> __reverted
202   @post msg.sender != owner -> __reverted
203   @post _stopTrade == true && msg.sender == owner -> __post._stopTrade != true
204   */
```

Line 205-208 in File fleta.sol

```
205 function startTrade() public onlyOwner {
206   require(_stopTrade == true);
207   _stopTrade = false;
208 }
```

The code meets the specification

Formal Verification Request 21

If method completes, integer overflow would not happen.

08, Apr 2019
7.81 ms

Line 214 in File fleta.sol

```
214 //@CTK NO_OVERFLOW
```

Line 220-222 in File fleta.sol

```
220 function balanceOf(address tokenOwner) public view returns (uint balance) {
221   return balances[tokenOwner];
222 }
```

The code meets the specification

Formal Verification Request 22

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

08, Apr 2019
0.45 ms

Line 215 in File fleta.sol

```
215 //@CTK NO_BUF_OVERFLOW
```

Line 220-222 in File fleta.sol

```
220 function balanceOf(address tokenOwner) public view returns (uint balance) {
221   return balances[tokenOwner];
222 }
```

The code meets the specification

Formal Verification Request 23

Method will not encounter an assertion failure.

08, Apr 2019
0.45 ms

Line 216 in File fleta.sol

```
216 //@CTK NO_ASF
```

Line 220-222 in File fleta.sol

```
220 function balanceOf(address tokenOwner) public view returns (uint balance) {
221   return balances[tokenOwner];
222 }
```

The code meets the specification

Formal Verification Request 24

balanceOf

08, Apr 2019
0.43 ms

Line 217-219 in File fleta.sol

```
/*@CTK balanceOf
    @post balance == balances[tokenOwner]
  */
```

Line 220-222 in File fleta.sol

```
function balanceOf(address tokenOwner) public view returns (uint balance) {
  return balances[tokenOwner];
}
```

✅ The code meets the specification

Formal Verification Request 25

If method completes, integer overflow would not happen.

📅 08, Apr 2019
⏱ 59.44 ms

Line 230 in File fleta.sol

```
//@CTK NO_OVERFLOW
```

Line 244-252 in File fleta.sol

```
function transfer(address to, uint tokens) public returns (bool success) {
  require(_stopTrade != true);
  require(to > address(0));

  balances[msg.sender] = balances[msg.sender].sub(tokens);
  balances[to] = balances[to].add(tokens);
  emit Transfer(msg.sender, to, tokens);
  return true;
}
```

✅ The code meets the specification

Formal Verification Request 26

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

📅 08, Apr 2019
⏱ 9.44 ms

Line 231 in File fleta.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 244-252 in File fleta.sol

```
function transfer(address to, uint tokens) public returns (bool success) {
  require(_stopTrade != true);
  require(to > address(0));

  balances[msg.sender] = balances[msg.sender].sub(tokens);
  balances[to] = balances[to].add(tokens);
```

```
250     emit Transfer(msg.sender, to, tokens);
251     return true;
252   }
```

The code meets the specification

Formal Verification Request 27

Method will not encounter an assertion failure.

08, Apr 2019
9.34 ms

Line 232 in File fleta.sol

```
232   //@CTK NO_ASF
```

Line 244-252 in File fleta.sol

```
244   function transfer(address to, uint tokens) public returns (bool success) {
245     require(_stopTrade != true);
246     require(to > address(0));
247
248     balances[msg.sender] = balances[msg.sender].sub(tokens);
249     balances[to] = balances[to].add(tokens);
250     emit Transfer(msg.sender, to, tokens);
251     return true;
252   }
```

The code meets the specification

Formal Verification Request 28

transfer

08, Apr 2019
129.55 ms

Line 233-238 in File fleta.sol

```
233   /*@CTK transfer
234     @tag assume_completion
235       @pre to != msg.sender
236       @post __post.balances[msg.sender] == balances[msg.sender] - tokens
237       @post __post.balances[to] == balances[to] + tokens
238     */
```

Line 244-252 in File fleta.sol

```
244   function transfer(address to, uint tokens) public returns (bool success) {
245     require(_stopTrade != true);
246     require(to > address(0));
247
248     balances[msg.sender] = balances[msg.sender].sub(tokens);
249     balances[to] = balances[to].add(tokens);
```

```
    emit Transfer(msg.sender, to, tokens);
    return true;
  }
```

The code meets the specification

Formal Verification Request 29

transfer_same

08, Apr 2019
50.23 ms

Line 239-243 in File fleta.sol

```
239 /*@CTK transfer_same
240   @tag assume_completion
241     @pre to == msg.sender
242     @post __post.balances[msg.sender] == balances[msg.sender]
243   */
```

Line 244-252 in File fleta.sol

```
244 function transfer(address to, uint tokens) public returns (bool success) {
245   require(_stopTrade != true);
246   require(to > address(0));
247
248   balances[msg.sender] = balances[msg.sender].sub(tokens);
249   balances[to] = balances[to].add(tokens);
250   emit Transfer(msg.sender, to, tokens);
251   return true;
252 }
```

The code meets the specification

Formal Verification Request 30

If method completes, integer overflow would not happen.

08, Apr 2019
23.2 ms

Line 263 in File fleta.sol

```
263 //@CTK NO_OVERFLOW
```

Line 270-276 in File fleta.sol

```
270 function approve(address spender, uint tokens) public returns (bool success) {
271   require(_stopTrade != true);
272
273   allowed[msg.sender][spender] = tokens;
274   emit Approval(msg.sender, spender, tokens);
275   return true;
276 }
```

The code meets the specification

Formal Verification Request 31

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

08, Apr 2019
0.54 ms

Line 264 in File fleta.sol

```
264 //@CTK NO_BUF_OVERFLOW
```

Line 270-276 in File fleta.sol

```
270 function approve(address spender, uint tokens) public returns (bool success) {
271   require(_stopTrade != true);
272
273   allowed[msg.sender][spender] = tokens;
274   emit Approval(msg.sender, spender, tokens);
275   return true;
276 }
```

The code meets the specification

Formal Verification Request 32

Method will not encounter an assertion failure.

08, Apr 2019
0.51 ms

Line 265 in File fleta.sol

```
265 //@CTK NO_ASF
```

Line 270-276 in File fleta.sol

```
270 function approve(address spender, uint tokens) public returns (bool success) {
271   require(_stopTrade != true);
272
273   allowed[msg.sender][spender] = tokens;
274   emit Approval(msg.sender, spender, tokens);
275   return true;
276 }
```

The code meets the specification

Formal Verification Request 33

approve

08, Apr 2019
2.22 ms

Line 266-269 in File fleta.sol

```
266 /*@CTK approve
267   @pre _stopTrade != true
268     @post __post.allowed[msg.sender][spender] == tokens
269   */
```

Line 270-276 in File fleta.sol

```
270 function approve(address spender, uint tokens) public returns (bool success) {
271   require(_stopTrade != true);
272
273   allowed[msg.sender][spender] = tokens;
274   emit Approval(msg.sender, spender, tokens);
275   return true;
276 }
```

The code meets the specification

Formal Verification Request 34

If method completes, integer overflow would not happen.

08, Apr 2019
106.23 ms

Line 288 in File fleta.sol

```
288   //@CTK NO_OVERFLOW
```

Line 313-325 in File fleta.sol

```
313 function transferFrom(address from, address to, uint tokens) public returns (bool
      success) {
314   require(_stopTrade != true);
315   require(from > address(0));
316   require(to > address(0));
317
318   balances[from] = balances[from].sub(tokens);
319   if(from != to && from != msg.sender) {
320     allowed[from][msg.sender] = allowed[from][msg.sender].sub(tokens);
321   }
322   balances[to] = balances[to].add(tokens);
323   emit Transfer(from, to, tokens);
324   return true;
325 }
```

The code meets the specification

Formal Verification Request 35

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

08, Apr 2019
24.87 ms

Line 289 in File fleta.sol

```
289 //@CTK NO_BUF_OVERFLOW
```

Line 313-325 in File fleta.sol

```
313 function transferFrom(address from, address to, uint tokens) public returns (bool
        success) {
314   require(_stopTrade != true);
315   require(from > address(0));
316   require(to > address(0));
317
318   balances[from] = balances[from].sub(tokens);
319   if(from != to && from != msg.sender) {
320     allowed[from][msg.sender] = allowed[from][msg.sender].sub(tokens);
321   }
322   balances[to] = balances[to].add(tokens);
323   emit Transfer(from, to, tokens);
324   return true;
325 }
```

✅ The code meets the specification

Formal Verification Request 36

Method will not encounter an assertion failure.

📅 08, Apr 2019
⏱ 22.25 ms

Line 290 in File fleta.sol

```
290 //@CTK NO_ASF
```

Line 313-325 in File fleta.sol

```
313 function transferFrom(address from, address to, uint tokens) public returns (bool
        success) {
314   require(_stopTrade != true);
315   require(from > address(0));
316   require(to > address(0));
317
318   balances[from] = balances[from].sub(tokens);
319   if(from != to && from != msg.sender) {
320     allowed[from][msg.sender] = allowed[from][msg.sender].sub(tokens);
321   }
322   balances[to] = balances[to].add(tokens);
323   emit Transfer(from, to, tokens);
324   return true;
325 }
```

✅ The code meets the specification

Formal Verification Request 37

transferFrom

📅 08, Apr 2019
⏱ 317.88 ms

Line 291-298 in File fleta.sol

```
291 /*@CTK transferFrom
292     @tag assume_completion
293     @pre from != to
294     @pre from != msg.sender
295     @post __post.balances[from] == balances[from] - tokens
296     @post __post.allowed[from][msg.sender] == allowed[from][msg.sender] - tokens
297     @post __post.balances[to] == balances[to] + tokens
298   */
```

Line 313-325 in File fleta.sol

```
313 function transferFrom(address from, address to, uint tokens) public returns (bool
        success) {
314   require(_stopTrade != true);
315   require(from > address(0));
316   require(to > address(0));
317
318   balances[from] = balances[from].sub(tokens);
319   if(from != to && from != msg.sender) {
320     allowed[from][msg.sender] = allowed[from][msg.sender].sub(tokens);
321   }
322   balances[to] = balances[to].add(tokens);
323   emit Transfer(from, to, tokens);
324   return true;
325 }
```

✅ The code meets the specification

Formal Verification Request 38

transferFrom_sameAddress

📅 08, Apr 2019
⏱ 70.09 ms

Line 299-304 in File fleta.sol

```
299   /*@CTK "transferFrom_sameAddress"
300     @tag assume_completion
301     @pre from == to
302     @post __post.allowed[from][msg.sender] == allowed[from][msg.sender]
303     @post __post.balances[to] == balances[to]
304   */
```

Line 313-325 in File fleta.sol

```
313 function transferFrom(address from, address to, uint tokens) public returns (bool
        success) {
314   require(_stopTrade != true);
315   require(from > address(0));
316   require(to > address(0));
317
318   balances[from] = balances[from].sub(tokens);
319   if(from != to && from != msg.sender) {
320     allowed[from][msg.sender] = allowed[from][msg.sender].sub(tokens);
```

```
    }
    balances[to] = balances[to].add(tokens);
    emit Transfer(from, to, tokens);
    return true;
  }
```

The code meets the specification

Formal Verification Request 39

transferFrom_sameAddress2

08, Apr 2019
153.43 ms

Line 305-312 in File fleta.sol

```
305 /*@CTK "transferFrom_sameAddress2"
306     @tag assume_completion
307     @pre from != to
308     @pre from == msg.sender
309     @post __post.allowed[from][msg.sender] == allowed[from][msg.sender]
310     @post __post.balances[from] == balances[from] - tokens
311     @post __post.balances[to] == balances[to] + tokens
312   */
```

Line 313-325 in File fleta.sol

```
313 function transferFrom(address from, address to, uint tokens) public returns (bool
        success) {
314   require(_stopTrade != true);
315   require(from > address(0));
316   require(to > address(0));
317
318   balances[from] = balances[from].sub(tokens);
319   if(from != to && from != msg.sender) {
320     allowed[from][msg.sender] = allowed[from][msg.sender].sub(tokens);
321   }
322   balances[to] = balances[to].add(tokens);
323   emit Transfer(from, to, tokens);
324   return true;
325 }
```

The code meets the specification

Formal Verification Request 40

If method completes, integer overflow would not happen.

08, Apr 2019
16.36 ms

Line 332 in File fleta.sol

```
332 //@CTK NO_OVERFLOW
```

Line 339-343 in File fleta.sol

```
339 function allowance(address tokenOwner, address spender) public view returns (uint
        remaining) {
340   require(_stopTrade != true);
341
342   return allowed[tokenOwner][spender];
343 }
```

The code meets the specification

Formal Verification Request 41

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

08, Apr 2019
0.51 ms

Line 333 in File fleta.sol

```
333 //@CTK NO_BUF_OVERFLOW
```

Line 339-343 in File fleta.sol

```
339 function allowance(address tokenOwner, address spender) public view returns (uint
        remaining) {
340   require(_stopTrade != true);
341
342   return allowed[tokenOwner][spender];
343 }
```

The code meets the specification

Formal Verification Request 42

Method will not encounter an assertion failure.

08, Apr 2019
0.5 ms

Line 334 in File fleta.sol

```
334 //@CTK NO_ASF
```

Line 339-343 in File fleta.sol

```
339 function allowance(address tokenOwner, address spender) public view returns (uint
        remaining) {
340   require(_stopTrade != true);
341
342   return allowed[tokenOwner][spender];
343 }
```

The code meets the specification

Formal Verification Request 43

allowance

08, Apr 2019
1.43 ms

Line 335-338 in File fleta.sol

```
335 /*@CTK allowance
336   @pre _stopTrade != true
337     @post remaining == allowed[tokenOwner][spender]
338   */
```

Line 339-343 in File fleta.sol

```
339 function allowance(address tokenOwner, address spender) public view returns (uint
        remaining) {
340   require(_stopTrade != true);
341
342   return allowed[tokenOwner][spender];
343 }
```

The code meets the specification