CertiK Verification Report For Sparrow Exchange

Request Date: 2019-04-02
Revision Date: 2019-04-07

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 Sparrow Exchange(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.

PASS

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

Apr 07, 2019

Summary

This audit report summarises the smart contract verification service requested by Sparrow Exchange. 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 openzeppelin-solidity/contracts/token/ERC20/ERC20.sol

```
pragma solidity ^0.5.0;

import "./IERC20.sol";
import "../../math/SafeMath.sol";

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
 * Originally based on code by FirstBlood:
 * https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
 *
 * This implementation emits additional Approval events, allowing applications to reconstruct the allowance status for
 * all accounts just by listening to said events. Note that this isn't required by the specification, and other
 * compliant implementations may not do it.
 */
contract ERC20 is IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowed;

    uint256 private _totalSupply;

    /**
    * @dev Total number of tokens in existence
    */
    /*@CTK totalSupply
      @post __return == _totalSupply
     */
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    /**
    * @dev Gets the balance of the specified address.
    * @param owner The address to query the balance of.
    * @return An uint256 representing the amount owned by the passed address.
    */
    /*@CTK balanceOf
      @post __return == _balances[owner]
     */
    function balanceOf(address owner) public view returns (uint256) {
        return _balances[owner];
    }

    /**
     * @dev Function to check the amount of tokens that an owner allowed to a spender.
     * @param owner address The address which owns the funds.
```

```
     * @param spender address The address which will spend the funds.
     * @return A uint256 specifying the amount of tokens still available for the spender.
     */
    /*@CTK allowance
      @post __return == _allowed[owner][spender]
     */
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowed[owner][spender];
    }

    /**
    * @dev Transfer token for a specified address
    * @param to The address to transfer to.
    * @param value The amount to be transferred.
    */
    /*@CTK transfer
      @tag assume_completion
      @pre msg.sender != to
      @post to != address(0)
      @post value <= _balances[msg.sender]
      @post __post._balances[to] == _balances[to] + value
      @post __post._balances[msg.sender] == _balances[msg.sender] - value
     */
    function transfer(address to, uint256 value) public returns (bool) {
        _transfer(msg.sender, to, value);
        return true;
    }

    /**
     * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
     * Beware that changing an allowance with this method brings the risk that someone may use both the old
     * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this
     * race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     * @param spender The address which will spend the funds.
     * @param value The amount of tokens to be spent.
     */
    /*@CTK approve
      @tag assume_completion
      @post spender != address(0)
      @post __post._allowed[msg.sender][spender] == value
     */
    function approve(address spender, uint256 value) public returns (bool) {
        require(spender != address(0));

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

    /**
     * @dev Transfer tokens from one address to another.
```

```
     * Note that while this function emits an Approval event, this is not required as per the specification,
     * and other compliant implementations may not emit the event.
     * @param from address The address which you want to send tokens from
     * @param to address The address which you want to transfer to
     * @param value uint256 the amount of tokens to be transferred
     */
    /*@CTK transfer_from
      @tag assume_completion
      @pre from != to
      @post to != address(0)
      @post value <= _allowed[from][msg.sender]
      @post __post._balances[from] == _balances[from] - value
      @post __post._balances[to] == _balances[to] + value
      @post __post._allowed[from][msg.sender] ==
      _allowed[from][msg.sender] - value
     */
    function transferFrom(address from, address to, uint256 value) public returns (bool) {
        _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
        _transfer(from, to, value);
        emit Approval(from, msg.sender, _allowed[from][msg.sender]);
        return true;
    }

    /**
     * @dev Increase the amount of tokens that an owner allowed to a spender.
     * approve should be called when allowed_[_spender] == 0. To increment
     * allowed value is better to use this function to avoid 2 calls (and wait until
     * the first transaction is mined)
     * From MonolithDAO Token.sol
     * Emits an Approval event.
     * @param spender The address which will spend the funds.
     * @param addedValue The amount of tokens to increase the allowance by.
     */
    /*@CTK increaseAllowance
      @tag assume_completion
      @post spender != address(0)
      @post __post._allowed[msg.sender][spender] ==
          _allowed[msg.sender][spender] + addedValue
     */
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        require(spender != address(0));

        _allowed[msg.sender][spender] = _allowed[msg.sender][spender].add(addedValue);
        emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
        return true;
    }

    /**
     * @dev Decrease the amount of tokens that an owner allowed to a spender.
     * approve should be called when allowed_[_spender] == 0. To decrement
     * allowed value is better to use this function to avoid 2 calls (and wait until
     * the first transaction is mined)
     * From MonolithDAO Token.sol
     * Emits an Approval event.
     * @param spender The address which will spend the funds.
```

```
     * @param subtractedValue The amount of tokens to decrease the allowance by.
     */
    /*@CTK decreaseAllowance
      @tag assume_completion
      @post spender != address(0)
      @post __post._allowed[msg.sender][spender] ==
          _allowed[msg.sender][spender] - subtractedValue
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public
        returns (bool) {
        require(spender != address(0));

        _allowed[msg.sender][spender] = _allowed[msg.sender][spender].sub(
            subtractedValue);
        emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
        return true;
    }

    /**
    * @dev Transfer token for a specified addresses
    * @param from The address to transfer from.
    * @param to The address to transfer to.
    * @param value The amount to be transferred.
    */
    /*@CTK _transfer
      @tag assume_completion
      @pre from != to
      @post to != address(0)
      @post __post._balances[from] == _balances[from] - value
      @post __post._balances[to] == _balances[to] + value
     */
    function _transfer(address from, address to, uint256 value) internal {
        require(to != address(0));

        _balances[from] = _balances[from].sub(value);
        _balances[to] = _balances[to].add(value);
        emit Transfer(from, to, value);
    }

    /**
     * @dev Internal function that mints an amount of the token and assigns it to
     * an account. This encapsulates the modification of balances such that the
     * proper events are emitted.
     * @param account The account that will receive the created tokens.
     * @param value The amount that will be created.
     */
    /*@CTK _mint
      @tag assume_completion
      @post account != 0
      @post __post._totalSupply == _totalSupply + value
      @post __post._balances[account] == _balances[account] + value
     */
    function _mint(address account, uint256 value) internal {
        require(account != address(0));

        _totalSupply = _totalSupply.add(value);
        _balances[account] = _balances[account].add(value);
        emit Transfer(address(0), account, value);
```

```
    }

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account.
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    /*@CTK _burn
      @tag assume_completion
      @post account != 0
      @post value <= _balances[account]
      @post __post._totalSupply == _totalSupply - value
      @post __post._balances[account] == _balances[account] - value
     */
    function _burn(address account, uint256 value) internal {
        require(account != address(0));

        _totalSupply = _totalSupply.sub(value);
        _balances[account] = _balances[account].sub(value);
        emit Transfer(account, address(0), value);
    }

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account, deducting from the sender's allowance for said account. Uses the
     * internal burn function.
     * Emits an Approval event (reflecting the reduced allowance).
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    /*@CTK _burnFrom
      @tag assume_completion
      @post value <= _allowed[account][msg.sender]
      @post __post._allowed[account][msg.sender] == _allowed[account][msg.sender] -
          value
      @post __post._totalSupply == _totalSupply - value
      @post __post._balances[account] == _balances[account] - value
     */
    function _burnFrom(address account, uint256 value) internal {
        _allowed[account][msg.sender] = _allowed[account][msg.sender].sub(value);
        _burn(account, value);
        emit Approval(account, msg.sender, _allowed[account][msg.sender]);
    }
}
```

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

Formal Verification Request 1

totalSupply

📅 07, Apr 2019
⏱ 6.31 ms

Line 30-32 in File ERC20.sol

```
30  /*@CTK totalSupply
31    @post __return == _totalSupply
32   */
```

Line 33-35 in File ERC20.sol

```
33  function totalSupply() public view returns (uint256) {
34      return _totalSupply;
35  }
```

✅ The code meets the specification

Formal Verification Request 2

balanceOf

📅 07, Apr 2019
⏱ 5.96 ms

Line 42-44 in File ERC20.sol

```
42  /*@CTK balanceOf
43    @post __return == _balances[owner]
44   */
```

Line 45-47 in File ERC20.sol

```
45  function balanceOf(address owner) public view returns (uint256) {
46      return _balances[owner];
47  }
```

✅ The code meets the specification

Formal Verification Request 3

allowance

📅 07, Apr 2019
⏱ 5.99 ms

Line 55-57 in File ERC20.sol

```
55  /*@CTK allowance
56    @post __return == _allowed[owner][spender]
57   */
```

Line 58-60 in File ERC20.sol

```
58    function allowance(address owner, address spender) public view returns (uint256) {
59        return _allowed[owner][spender];
60    }
```

The code meets the specification

Formal Verification Request 4

transfer

07, Apr 2019
224.23 ms

Line 67-74 in File ERC20.sol

```
67    /*@CTK transfer
68      @tag assume_completion
69      @pre msg.sender != to
70      @post to != address(0)
71      @post value <= _balances[msg.sender]
72      @post __post._balances[to] == _balances[to] + value
73      @post __post._balances[msg.sender] == _balances[msg.sender] - value
74     */
```

Line 75-78 in File ERC20.sol

```
75    function transfer(address to, uint256 value) public returns (bool) {
76        _transfer(msg.sender, to, value);
77        return true;
78    }
```

The code meets the specification

Formal Verification Request 5

approve

07, Apr 2019
18.22 ms

Line 89-93 in File ERC20.sol

```
89    /*@CTK approve
90      @tag assume_completion
91      @post spender != address(0)
92      @post __post._allowed[msg.sender][spender] == value
93     */
```

Line 94-100 in File ERC20.sol

```
94    function approve(address spender, uint256 value) public returns (bool) {
95        require(spender != address(0));
96
97        _allowed[msg.sender][spender] = value;
98        emit Approval(msg.sender, spender, value);
99        return true;
100   }
```

The code meets the specification

Formal Verification Request 6

transfer_from

07, Apr 2019
221.76 ms

Line 110-119 in File ERC20.sol

```
110   /*@CTK transfer_from
111     @tag assume_completion
112     @pre from != to
113     @post to != address(0)
114     @post value <= _allowed[from][msg.sender]
115     @post __post._balances[from] == _balances[from] - value
116     @post __post._balances[to] == _balances[to] + value
117     @post __post._allowed[from][msg.sender] ==
118     _allowed[from][msg.sender] - value
119    */
```

Line 120-125 in File ERC20.sol

```
120   function transferFrom(address from, address to, uint256 value) public returns (
          bool) {
121       _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
122       _transfer(from, to, value);
123       emit Approval(from, msg.sender, _allowed[from][msg.sender]);
124       return true;
125   }
```

The code meets the specification

Formal Verification Request 7

increaseAllowance

07, Apr 2019
53.19 ms

Line 137-142 in File ERC20.sol

```
137   /*@CTK increaseAllowance
138     @tag assume_completion
139     @post spender != address(0)
140     @post __post._allowed[msg.sender][spender] ==
141         _allowed[msg.sender][spender] + addedValue
142    */
```

Line 143-149 in File ERC20.sol

```
143   function increaseAllowance(address spender, uint256 addedValue) public returns (
          bool) {
144       require(spender != address(0));
```

```
        _allowed[msg.sender][spender] = _allowed[msg.sender][spender].add(addedValue);
        emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
        return true;
    }
```

The code meets the specification

Formal Verification Request 8

decreaseAllowance

07, Apr 2019
50.33 ms

Line 161-166 in File ERC20.sol

```
    /*@CTK decreaseAllowance
      @tag assume_completion
      @post spender != address(0)
      @post __post._allowed[msg.sender][spender] ==
          _allowed[msg.sender][spender] - subtractedValue
     */
```

Line 167-173 in File ERC20.sol

```
    function decreaseAllowance(address spender, uint256 subtractedValue) public
        returns (bool) {
        require(spender != address(0));

        _allowed[msg.sender][spender] = _allowed[msg.sender][spender].sub(
            subtractedValue);
        emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
        return true;
    }
```

The code meets the specification

Formal Verification Request 9

_transfer

07, Apr 2019
46.37 ms

Line 181-187 in File ERC20.sol

```
    /*@CTK _transfer
      @tag assume_completion
      @pre from != to
      @post to != address(0)
      @post __post._balances[from] == _balances[from] - value
      @post __post._balances[to] == _balances[to] + value
     */
```

Line 188-194 in File ERC20.sol

```
188 function _transfer(address from, address to, uint256 value) internal {
189     require(to != address(0));
190
191     _balances[from] = _balances[from].sub(value);
192     _balances[to] = _balances[to].add(value);
193     emit Transfer(from, to, value);
194 }
```

The code meets the specification

Formal Verification Request 10

_mint

07, Apr 2019
99.77 ms

Line 203-208 in File ERC20.sol

```
203 /*@CTK _mint
204   @tag assume_completion
205   @post account != 0
206   @post __post._totalSupply == _totalSupply + value
207   @post __post._balances[account] == _balances[account] + value
208  */
```

Line 209-215 in File ERC20.sol

```
209 function _mint(address account, uint256 value) internal {
210     require(account != address(0));
211
212     _totalSupply = _totalSupply.add(value);
213     _balances[account] = _balances[account].add(value);
214     emit Transfer(address(0), account, value);
215 }
```

The code meets the specification

Formal Verification Request 11

_burn

07, Apr 2019
117.44 ms

Line 223-229 in File ERC20.sol

```
223 /*@CTK _burn
224   @tag assume_completion
225   @post account != 0
226   @post value <= _balances[account]
227   @post __post._totalSupply == _totalSupply - value
228   @post __post._balances[account] == _balances[account] - value
229  */
```

Line 230-236 in File ERC20.sol

```
230    function _burn(address account, uint256 value) internal {
231        require(account != address(0));
232
233        _totalSupply = _totalSupply.sub(value);
234        _balances[account] = _balances[account].sub(value);
235        emit Transfer(account, address(0), value);
236    }
```

The code meets the specification

Formal Verification Request 12

_burnFrom

07, Apr 2019
256.25 ms

Line 246-252 in File ERC20.sol

```
246    /*@CTK _burnFrom
247      @tag assume_completion
248      @post value <= _allowed[account][msg.sender]
249      @post __post._allowed[account][msg.sender] == _allowed[account][msg.sender] -
             value
250      @post __post._totalSupply == _totalSupply - value
251      @post __post._balances[account] == _balances[account] - value
252     */
```

Line 253-257 in File ERC20.sol

```
253    function _burnFrom(address account, uint256 value) internal {
254        _allowed[account][msg.sender] = _allowed[account][msg.sender].sub(value);
255        _burn(account, value);
256        emit Approval(account, msg.sender, _allowed[account][msg.sender]);
257    }
```

The code meets the specification