CertiK-Tunwu Audit Report For SportX

SPORTX

Request Date: 2019-05-28
Revision Date: 2019-05-29
Platform Name: Ethereum

TUNWU

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-Tunwu and SportX(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-Tunwu’s prior written consent.

Exective Summary

This report has been prepared as product of the Smart Contract Audit request by SportX. This audit was conducted to discover issues and vulnerabilities in the source code of SportX's Smart Contracts. Utilizing CertiK-Tunwu's Formal Verification Platform, Static Analysis and Manual Review, a comprehensive examination has been performed. The auditing process pays special attention to the following considerations.

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessment of the codebase for best practice and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line by line manual review of the entire codebase by industry experts.

Vulnerability Classification

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

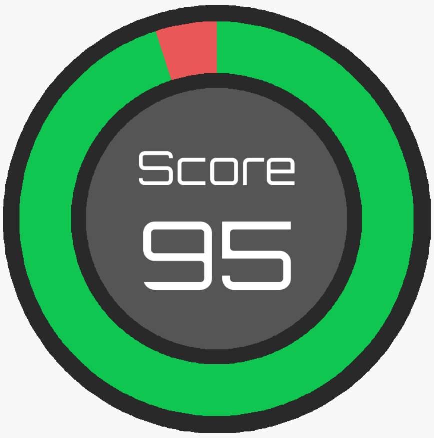
- Critical: The code implementation does not match the specification, or it could result in loss of funds for contract owner or users.
- Medium: The code implementation does not match the specification at certain condition, or it could affect the security standard by lost of access control.
- Low: The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

Testing Summary

PASS

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

May 29, 2019



Type of Issues

CertiK-Tunwu 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.	7	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.

Manual Review Notes

Review Details

Source Code SHA-256 Checksum

- **sportx.sol** `fbf975c4506694e078e1cf456c29c82c32d46879fcbaf38e7ff9d94898411f5d`

Summary

CertiK team is invited by The SportX team to audit the design and implementations of its to be released ERC20 based smart contract, and the source code has been analyzed under different perspectives and with different tools such as CertiK formal verification checking as well as manual reviews by smart contract experts. At this point the SportX team didn't provide other repositories sources as testing and documentation reference. We recommend to have more unit tests coverage together with documentation to simulate potential use cases and walk through the functionalities to token holders, especially those super admin privileges that may impact the decentralized nature.

With the final update of source code and delivery of the audit report, we conclude that the contract is not vulnerable to any classically known anti-patterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend seeking multiple opinions, more test coverage and sandbox deployments before the mainnet release.

Recommendations

Items in this section are low impact to the overall aspects of the smart contracts, thus will let client to decide whether to have those reflected in the final deployed version of source codes.

sportx.sol

- **mul(), sub(), add() in SafeMath** – The Solidity `assert()` function is meant to assert invariants. Properly functioning code should never reach a failing assert statement. Recommend using `require()` to replace `assert()`.

Source Code with CertiK-Tunwu Labels

File sportx.sol

```
/**
 * Source Code first verified at https://etherscan.io on Thursday, April 25, 2019
 (UTC) */

pragma solidity ^0.4.25;

library SafeMath {

  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  //@CTK FAIL NO_ASF
  /*@CTK "SafeMath mul"
    @post (((a) > (0)) && ((((a) * (b)) / (a)) != (b))) == (__reverted)
    @post !__reverted -> c == a * b
    @post !__reverted == !__has_overflow
    @post !(__has_buf_overflow)
   */
  function mul(uint256 a, uint256 b) internal pure returns (uint256 c) {
    if (a == 0) {
      return 0;
    }
    c = a * b;
    assert(c / a == b);
    return c;
  }

  /**
  * @dev Integer division of two numbers, truncating the quotient.
  */
  //@CTK FAIL NO_ASF
  /*@CTK "SafeMath div"
    @post b != 0 -> !__reverted
    @post !__reverted -> __return == a / b
    @post !__reverted -> !__has_overflow
    @post !(__has_buf_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 a / b;
  }

  /**
  * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than minuend).
  */
  //@CTK FAIL NO_ASF
  /*@CTK "SafeMath sub"
    @post (a < b) == __reverted
    @post !__reverted -> __return == a - b
    @post !__reverted -> !__has_overflow
    @post !(__has_buf_overflow)
   */
  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  /**
  * @dev Adds two numbers, throws on overflow.
  */
  //@CTK FAIL NO_ASF
  /*@CTK "SafeMath add"
    @post (a + b < a || a + b < b) == __reverted
    @post !__reverted -> c == a + b
    @post !__reverted -> !__has_overflow
    @post !(__has_buf_overflow)
   */
  function add(uint256 a, uint256 b) internal pure returns (uint256 c) {
    c = a + b;
    assert(c >= a);
    return c;
  }
}

contract ERC20Basic {
  function totalSupply() public view returns (uint256);
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}

contract ERC20 is ERC20Basic {
  function allowance(address owner, address spender) public view returns (uint256);
  function transferFrom(address from, address to, uint256 value) public returns (bool);
  function approve(address spender, uint256 value) public returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}

contract BasicToken is ERC20Basic {
  using SafeMath for uint256;

  mapping(address => uint256) balances;

  uint256 totalSupply_;

  /**
  * @dev total number of tokens in existence
  */
  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK "totalSupply correctness"
    @post __return == totalSupply_
   */
  function totalSupply() public view returns (uint256) {
    return totalSupply_;
  }

  /**
  * @dev transfer token for a specified address
  * @param _to The address to transfer to.
  * @param _value The amount to be transferred.
  */
  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK FAIL NO_ASF
  /*@CTK "transfer correctness"
    @tag assume_completion
    @post _to != 0x0
    @post _value <= balances[msg.sender]
    @post _to != msg.sender -> __post.balances[msg.sender] == balances[msg.sender] - _value
    @post _to != msg.sender -> __post.balances[_to] == balances[_to] + _value
    @post _to == msg.sender -> __post.balances[msg.sender] == balances[msg.sender]
   */
  function transfer(address _to, uint256 _value) public returns (bool) {
    require(_to != address(0));
    require(_value <= balances[msg.sender]);

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

  /**
  * @dev Gets the balance of the specified address.
  * @param _owner The address to query the the balance of.
  * @return An uint256 representing the amount owned by the passed address.
  */
  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK "balanceOf correctness"
    @post balance == balances[_owner]
   */
  function balanceOf(address _owner) public view returns (uint256 balance) {
    return balances[_owner];
  }

}

contract StandardToken is ERC20, BasicToken {

  mapping (address => mapping (address => uint256)) internal allowed;


  /**
   * @dev Transfer tokens from one address to another
   * @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 NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK FAIL NO_ASF
```

```
/*@CTK "transferFrom correctness"
  @tag assume_completion
  @post _to != 0x0
  @post _value <= balances[_from] && _value <= allowed[_from][msg.sender]
  @post _to != _from -> __post.balances[_from] == balances[_from] - _value
  @post _to != _from -> __post.balances[_to] == balances[_to] + _value
  @post _to == _from -> __post.balances[_from] == balances[_from]
  @post __post.allowed[_from][msg.sender] == allowed[_from][msg.sender] - _value
 */
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);
  emit Transfer(_from, _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 NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK "approve correctness"
  @post __post.allowed[msg.sender][_spender] == _value
 */
function approve(address _spender, uint256 _value) public returns (bool) {
  allowed[msg.sender][_spender] = _value;
  emit Approval(msg.sender, _spender, _value);
  return true;
}

/**
 * @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 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];
}

/**
 * @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
 * @param _spender The address which will spend the funds.
 * @param _addedValue The amount of tokens to increase the allowance by.
 */
//@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
 */
function increaseApproval(address _spender, uint _addedValue) public returns (bool) {
  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
 * @param _spender The address which will spend the funds.
 * @param _subtractedValue The amount of tokens to decrease the allowance by.
 */
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK decreaseApproval0
  @pre __return == true
  @pre allowed[msg.sender][_spender] <= _subtractedValue
  @post __post.allowed[msg.sender][_spender] == 0
*/
/*@CTK decreaseApproval
  @pre __return == true
  @pre allowed[msg.sender][_spender] > _subtractedValue
  @post __post.allowed[msg.sender][_spender] ==
       allowed[msg.sender][_spender] - _subtractedValue
*/
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);
    }
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }

}


contract SportX is StandardToken {

  string public constant name = "SPORTX"; // solium-disable-line uppercase
  string public constant symbol = "SOX"; // solium-disable-line uppercase
  uint8 public constant decimals = 4; // solium-disable-line uppercase

  uint256 public constant INITIAL_SUPPLY = 1000000000 * (10 ** uint256(decimals));

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK SportX
    @post __post.balances[msg.sender] == __post.totalSupply_
  */
  constructor() public {
    totalSupply_ = INITIAL_SUPPLY;
    balances[msg.sender] = INITIAL_SUPPLY;
    emit Transfer(0x0, msg.sender, INITIAL_SUPPLY);
  }

}
```