Smart Contract Security Audit Report

The SlowMist Security Team received the PSC team's application for smart contract security audit of the Polestar on January 10, 2020. The following are the details and results of this smart contract security audit:

Token name :

PSC

The Contract address :

0x022302f2520aFDB08463F204cdeE4BDFC213A387

Link address :

https://etherscan.io/address/0x022302f2520aFDB08463F204cdeE4BDFC213A387

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive auditing authority	Passed
4	Safety Design Audit	Zeppelin module safe use	Passed
		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed
6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Passed
8	“False Deposit” vulnerability Audit	-	Passed

9	Malicious Event Log Audit	-	Passed
10	Scoping and Declarations Audit	-	Passed
11	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
12	Uninitialized Storage Pointers Audit	-	Passed
13	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : **Passed**

Audit Number : 0X002001150002

Audit Date : January 15, 2020

Audit Team : SlowMist Security Team

Summary: This is a token contract that does not contain the tokenVault section. SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue. The comprehensive evaluation contract is no risk.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2020-01-08
*/

//SlowMist// The contract does not have the Overflow and the Race Conditions issue

pragma solidity ^0.5.0;

/**
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * See https://github.com/ethereum/EIPs/issues/179
 */
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);
}

/**
  * @title Basic token
  * @dev Basic version of StandardToken, with no allowances.
  */
contract BasicToken is ERC20Basic {
    using SafeMath for uint256;

    mapping(address => uint256) balances;

    uint256 totalSupply_;

    /**
    * @dev Total number of tokens in existence
    */
    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.
    */
    function transfer(address _to, uint256 _value) public returns (bool) {

        require(_to != address(0), "Address must not be zero."); //SlowMist// This kind of check is very good,
avoiding user mistake leading to the loss of token during transfer

        require(_value <= balances[msg.sender], "There is no enough balance.");

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

        return true; //SlowMist// The return value conforms to the EIP20 specification

    }
```

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

}

/**
  * @title ERC20 interface
  * @dev see https://github.com/ethereum/EIPs/issues/20
  */
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
    );
}

/**
  * @title Standard ERC20 token
  *
  * @dev Implementation of the basic standard token.
  * https://github.com/ethereum/EIPs/issues/20
  * Based on code by FirstBlood: https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
  */
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
    */
    function transferFrom(
        address _from,
        address _to,
        uint256 _value
    )
        public
        returns (bool)
    {

        require(_to != address(0), "Address must not be zero."); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer

        require(_value <= balances[_from], "There is no enough balance.");
        require(_value <= allowed[_from][msg.sender], "There is no enough allowed balance.");

        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; //SlowMist// The return value conforms to the EIP20 specification

    }


    /**
    * @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.
    */
    function approve(address _spender, uint256 _value) public returns (bool) {
        allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);
```

```
        return true; //SlowMist// The return value conforms to the EIP20 specification
    }

    /**
    * @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.
    */
    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.
    */
    function increaseApproval(
        address _spender,
        uint256 _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.
    */
    function decreaseApproval(
        address _spender,
        uint256 _subtractedValue
    )
        public
        returns (bool)
    {
        uint256 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 PSCToken is StandardToken {
    string public name = "Polestar";
    string public symbol = "PSC";
    uint8 public decimals = 18;
    uint256 public init_Supply = 10000000000 * 10 ** 18;

    constructor() public {
        totalSupply_ = init_Supply;
        balances[msg.sender] = totalSupply_;
    }
}

/**
```

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

//SlowMist// SafeMath security Module is used, which is a recommend approach

```
library SafeMath {

    /**
    * @dev Multiplies two numbers, throws on overflow.
    */
    function mul(uint256 a, uint256 b) internal pure returns (uint256 c) {
        // Gas optimization: this is cheaper than asserting 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        c = a * b;
        assert(c / a == b); //SlowMist// It is recommended to replace "assert" with "require" to
```

optimize Gas

```
        return c;
    }

    /**
    * @dev Integer division of two numbers, truncating the quotient.
    */
    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).
    */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        assert(b <= a); //SlowMist// It is recommended to replace "assert" with "require" to
```

```
optimize Gas
        return a - b;
    }

    /**
    * @dev Adds two numbers, throws on overflow.
    */
    function add(uint256 a, uint256 b) internal pure returns (uint256 c) {
        c = a + b;
        assert(c >= a); //SlowMist// It is recommended to replace "assert" with "require" to
optimize Gas
        return c;
    }
}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account