Smart Contract Security Audit Report

The SlowMist Security Team received the Vivi team's application for smart contract security audit of the Vivi token on Jan. 18, 2021. The following are the details and results of this smart contract security audit:

Token name :

Vivi

The Contract address :

0xACF420F57F9231F3EAcaA2407eA2ce459B4CFE25

Link address :

https://etherscan.io/address/0xACF420F57F9231F3EAcaA2407eA2ce459B4CFE25

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 authority audit	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 top-up” 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 : 0X002101200001

Audit Date : Jan. 20, 2021

Audit Team : SlowMist Security Team

Summary: This is a token contract that does not contain the tokenVault section. The total amount of contract tokens can be changed, users can burn their tokens through the burn function. OpenZeppelin's SafeMath security module is used, which is a recommend approach. The contract does not have the Overflow and the Race Conditions issue.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2021-01-11
*/

/**
 *Submitted for verification at Etherscan.io on 2019-02-19
*/

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

pragma solidity ^0.4.21 ;
```

```
// produced by the Solidity File Flattener (c) David Appleton 2018
// contact : dave@akomba.com
// released under Apache 2.0 licence

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);
}
```

//SlowMist// OpenZeppelin's 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) {
    if (a == 0) {
      return 0;
    }
    c = a * b;
    assert(c / a == b);
    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);
    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);
    return c;
  }
}

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)); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer
    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; //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];
  }

}
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)); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer
    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; //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 MonolithDAC 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, 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.
   */
  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 BurnableToken is BasicToken {

  event Burn(address indexed burner, uint256 value);

  /**
   * @dev Burns a specific amount of tokens.
   * @param _value The amount of token to be burned.
   */
  function burn(uint256 _value) public {
    _burn(msg.sender, _value);
  }
```

```
  function _burn(address _who, uint256 _value) internal {
    require(_value <= balances[_who]);
    // no need to require value <= totalSupply, since that would imply the
    // sender's balance is greater than the totalSupply, which *should* be an assertion failure

    balances[_who] = balances[_who].sub(_value);
    totalSupply_ = totalSupply_.sub(_value);
    emit Burn(_who, _value);
    emit Transfer(_who, address(0), _value);
  }
}
contract StandardBurnableToken is BurnableToken, StandardToken {

  /**
   * @dev Burns a specific amount of tokens from the target address and decrements allowance
   * @param _from address The address which you want to send tokens from
   * @param _value uint256 The amount of token to be burned
   */
  //SlowMist// Because burnFrom() and transferFrom() share the allowed amount of approve(), if the agent be evil, there is the possibility of malicious burn
  function burnFrom(address _from, uint256 _value) public {
    require(_value <= allowed[_from][msg.sender]);
    // Should https://github.com/OpenZeppelin/zeppelin-solidity/issues/707 be accepted,
    // this function needs to emit an event with the updated approval.
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
    _burn(_from, _value);
  }
}
contract ViviToken is StandardBurnableToken{
    string public name = "Vivi Token";
    string public symbol = "Vivi";
    uint8 public decimals = 8;
    uint256 public INITIAL_SUPPLY = 4000000000000000;
    constructor() public {
        totalSupply_ = INITIAL_SUPPLY;
        balances[msg.sender] = INITIAL_SUPPLY;
    }
}
```

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist