Smart Contract Security Audit Report

The SlowMist Security Team received the Origin team's application for smart contract security audit of the OGN on February 15, 2020. The following are the details and results of this smart contract security audit:

Token name :

OGN

The Contract address :

0x8207c1ffc5b6804f6024322ccf34f29c3541ae26

Link address :

https://etherscan.io/address/0x8207c1ffc5b6804f6024322ccf34f29c3541ae26

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 : 0X002002170001

Audit Date : February 17, 2020

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. SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue. It is recommended to limit the amount of gas available when transfer tokens through the approveAndCallWithSender function, to avoid Reentrancy risk.

During the audit we found some issues:

1. **The owner can burn tokens of any user through the burn function.**
2. **When mintingFinished is false, owner can use the mint function to unlimited mint tokens.**
3. **When the WhitelistExpiration expires, it cannot be reset.**

But Origin team confirms the contract is owned by 5 of 8 multisig and the whitelist expiration functionality isn't needed anymore.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2018-10-04
*/
```

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

```
pragma solidity ^0.4.24;
// produced by the Solididy 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 Ownable {
  address public owner;


  event OwnershipRenounced(address indexed previousOwner);
  event OwnershipTransferred(
    address indexed previousOwner,
    address indexed newOwner
  );


  /**
   * @dev The Ownable constructor sets the original `owner` of the contract to the sender
   * account.
   */
  constructor() public {
    owner = msg.sender;
  }

  /**
   * @dev Throws if called by any account other than the owner.
```

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

  /**
   * @dev Allows the current owner to relinquish control of the contract.
   */
  function renounceOwnership() public onlyOwner {
    emit OwnershipRenounced(owner);
    owner = address(0);
  }

  /**
   * @dev Allows the current owner to transfer control of the contract to a newOwner.
   * @param _newOwner The address to transfer ownership to.
   */
  function transferOwnership(address _newOwner) public onlyOwner {
    _transferOwnership(_newOwner);
  }

  /**
   * @dev Transfers control of the contract to a newOwner.
   * @param _newOwner The address to transfer ownership to.
   */
  function _transferOwnership(address _newOwner) internal {
    require(_newOwner != address(0)); //SlowMist// This check is quite good in avoiding losing control
of the contract caused by user mistakes
    emit OwnershipTransferred(owner, _newOwner);
    owner = _newOwner;
  }
}

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;
    }
}

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 Pausable is Ownable {
    event Pause();
    event Unpause();

    bool public paused = false;


    /**
     * @dev Modifier to make a function callable only when the contract is not paused.
     */
    modifier whenNotPaused() {
        require(!paused);
        _;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is paused.
     */
    modifier whenPaused() {
        require(paused);
        _;
```

```
  }

  /**
   * @dev called by the owner to pause, triggers stopped state
   */
  //SlowMist// Suspending all transactions upon major abnormalities is a recommended approach
  function pause() onlyOwner whenNotPaused public {
    paused = true;
    emit Pause();
  }

  /**
   * @dev called by the owner to unpause, returns to normal state
   */
  function unpause() onlyOwner whenPaused public {
    paused = false;
    emit Unpause();
  }
}

contract DetailedERC20 is ERC20 {
  string public name;
  string public symbol;
  uint8 public decimals;

  constructor(string _name, string _symbol, uint8 _decimals) public {
    name = _name;
    symbol = _symbol;
    decimals = _decimals;
  }
}

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 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 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 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,
    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 PausableToken is StandardToken, Pausable {

  function transfer(
    address _to,
    uint256 _value
  )
    public
    whenNotPaused
    returns (bool)
  {
    return super.transfer(_to, _value);
  }

  function transferFrom(
    address _from,
    address _to,
    uint256 _value
  )
    public
    whenNotPaused
    returns (bool)
  {
    return super.transferFrom(_from, _to, _value);
  }

  function approve(
    address _spender,
    uint256 _value
  )
```

```
    public
    whenNotPaused
    returns (bool)
  {
    return super.approve(_spender, _value);
  }

  function increaseApproval(
    address _spender,
    uint _addedValue
  )
    public
    whenNotPaused
    returns (bool success)
  {
    return super.increaseApproval(_spender, _addedValue);
  }

  function decreaseApproval(
    address _spender,
    uint _subtractedValue
  )
    public
    whenNotPaused
    returns (bool success)
  {
    return super.decreaseApproval(_spender, _subtractedValue);
  }
}

contract MintableToken is StandardToken, Ownable {
  event Mint(address indexed to, uint256 amount);
  event MintFinished();

  bool public mintingFinished = false;


  modifier canMint() {
    require(!mintingFinished);
    _;
  }
```

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

  /**
   * @dev Function to mint tokens
   * @param _to The address that will receive the minted tokens.
   * @param _amount The amount of tokens to mint.
   * @return A boolean that indicates if the operation was successful.
   */
  function mint(
    address _to,
    uint256 _amount
  )
    hasMintPermission
    canMint
    public
    returns (bool)
  {
    totalSupply_ = totalSupply_.add(_amount);
    balances[_to] = balances[_to].add(_amount);
    emit Mint(_to, _amount);
    emit Transfer(address(0), _to, _amount);
    return true;
  }

  /**
   * @dev Function to stop minting new tokens.
   * @return True if the operation was successful.
   */
  function finishMinting() onlyOwner canMint public returns (bool) {
    mintingFinished = true;
    emit MintFinished();
    return true;
  }
}

contract WhitelistedPausableToken is PausableToken {
    // UNIX timestamp (in seconds) after which this whitelist no longer applies
    uint256 public whitelistExpiration;
    // While the whitelist is active, either the sender or recipient must be
```

```
// in allowedTransactors.
mapping (address => bool) public allowedTransactors;

event SetWhitelistExpiration(uint256 expiration);
event AllowedTransactorAdded(address sender);
event AllowedTransactorRemoved(address sender);

//
// Functions for maintaining whitelist
//

modifier allowedTransfer(address _from, address _to) {
    require(
        // solium-disable-next-line operator-whitespace
        !whitelistActive() ||
        allowedTransactors[_from] ||
        allowedTransactors[_to],
        "neither sender nor recipient are allowed"
    );
    _;
}

function whitelistActive() public view returns (bool) {
    return block.timestamp < whitelistExpiration;
}

function addAllowedTransactor(address _transactor) public onlyOwner {
    emit AllowedTransactorAdded(_transactor);
    allowedTransactors[_transactor] = true;
}

function removeAllowedTransactor(address _transactor) public onlyOwner {
    emit AllowedTransactorRemoved(_transactor);
    delete allowedTransactors[_transactor];
}

/**
* @dev Set the whitelist expiration, after which the whitelist no longer
* applies.
*/
```

//SlowMist// After confirming with the project party, the whitelist expiration functionality

isn't needed anymore

```
function setWhitelistExpiration(uint256 _expiration) public onlyOwner {
    // allow only if whitelist expiration hasn't yet been set, or if the
    // whitelist expiration hasn't passed yet
    require(
        whitelistExpiration == 0 || whitelistActive(),
        "an expired whitelist cannot be extended"
    );
    // prevent possible mistakes in calling this function
    require(
        _expiration >= block.timestamp + 1 days,
        "whitelist expiration not far enough into the future"
    );
    emit SetWhitelistExpiration(_expiration);
    whitelistExpiration = _expiration;
}

//
// ERC20 transfer functions that have been overridden to enforce the
// whitelist.
//

function transfer(
    address _to,
    uint256 _value
)
    public
    allowedTransfer(msg.sender, _to)
    returns (bool)
{
    return super.transfer(_to, _value);
}

function transferFrom(
    address _from,
    address _to,
    uint256 _value
)
public
    allowedTransfer(_from, _to)
returns (bool)
```

```
    {
        return super.transferFrom(_from, _to, _value);
    }
}

contract OriginToken is BurnableToken, MintableToken, WhitelistedPausableToken, DetailedERC20 {
    event AddCallSpenderWhitelist(address enabler, address spender);
    event RemoveCallSpenderWhitelist(address disabler, address spender);

    mapping (address => bool) public callSpenderWhitelist;

    // @dev Constructor that gives msg.sender all initial tokens.
    constructor(uint256 _initialSupply) DetailedERC20("OriginToken", "OGN", 18) public {
        owner = msg.sender;
        mint(owner, _initialSupply);
    }

    //
    // Burn methods
    //

    // @dev Burns tokens belonging to the sender
    // @param _value Amount of token to be burned
    function burn(uint256 _value) public onlyOwner {
        // TODO: add a function & modifier to enable for all accounts without doing
        // a contract migration?
        super.burn(_value);
    }

    // @dev Burns tokens belonging to the specified address
    // @param _who The account whose tokens we're burning
    // @param _value Amount of token to be burned
    function burn(address _who, uint256 _value) public onlyOwner {
        _burn(_who, _value);
    }

    //
    // approveAndCall methods
    //

    // @dev Add spender to whitelist of spenders for approveAndCall
    // @param _spender Address to add
```

```
function addCallSpenderWhitelist(address _spender) public onlyOwner {
    callSpenderWhitelist[_spender] = true;
    emit AddCallSpenderWhitelist(msg.sender, _spender);
}

// @dev Remove spender from whitelist of spenders for approveAndCall
// @param _spender Address to remove
function removeCallSpenderWhitelist(address _spender) public onlyOwner {
    delete callSpenderWhitelist[_spender];
    emit RemoveCallSpenderWhitelist(msg.sender, _spender);
}

// @dev Approve transfer of tokens and make a contract call in a single
// @dev transaction. This allows a DApp to avoid requiring two MetaMask
// @dev approvals for a single logical action, such as creating a listing,
// @dev which requires the seller to approve a token transfer and the
// @dev marketplace contract to transfer tokens from the seller.
//
// @dev This is based on the ERC827 function approveAndCall and avoids
// @dev security issues by only working with a whitelisted set of _spender
// @dev addresses. The other difference is that the combination of this
// @dev function ensures that the proxied function call receives the
// @dev msg.sender for this function as its first parameter.
//
// @param _spender The address that will spend the funds.
// @param _value The amount of tokens to be spent.
// @param _selector Function selector for function to be called.
// @param _callParams Packed, encoded parameters, omitting the first parameter which is always msg.sender
function approveAndCallWithSender(
    address _spender,
    uint256 _value,
    bytes4 _selector,
    bytes _callParams
)
    public
    payable
    returns (bool)
{
    require(_spender != address(this), "token contract can't be approved");
    require(callSpenderWhitelist[_spender], "spender not in whitelist");

    require(super.approve(_spender, _value), "approve failed");
```

```
        bytes memory callData = abi.encodePacked(_selector, uint256(msg.sender), _callParams);
        // solium-disable-next-line security/no-call-value
        require(_spender.call.value(msg.value)(callData), "proxied call failed");

        //SlowMist//  require(_spender.call.value(msg.value).gas(gas_value)(callData),  "proxied call failed");

        //SlowMist// It is recommended to modify the above code, to avoid Reentrancy risk

        return true;
    }
}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account