Smart Contract Security Audit Report

The SlowMist Security Team received the GET team's application for smart contract security audit of the Guaranteed Entrance Token on January 06, 2020. The following are the details and results of this smart contract security audit:

Token name :

GET

The Contract address :

0x8a854288a5976036a725879164ca3e91d30c6a1b

Link address :

https://etherscan.io/address/0x8a854288a5976036a725879164ca3e91d30c6a1b

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 : 0X002001100002

Audit Date : January 10, 2020

Audit Team : SlowMist Security Team

(**Statement :** SlowMist only issues this report based on the fact that has occurred or existed before the report is issued, and bears the corresponding responsibility in this regard. For the facts occur or exist later after the report, SlowMist cannot judge the security status of its smart contract. SlowMist is not responsible for it. The security audit analysis and other contents of this report are based on the documents and materials provided by the information provider to SlowMist as of the date of this report (referred to as "the provided information"). SlowMist assumes that: there has been no information missing, tampered, deleted, or concealed. If the information provided has been missed, modified, deleted, concealed or reflected and is inconsistent with the actual situation, SlowMist will not bear any responsibility for the resulting loss and adverse effects. SlowMist will not bear any responsibility for the background or other circumstances of the project.)

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. mintAgent can mint tokens without limit through the mint function. SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2017-11-13
*/
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
//SlowMist// Contract deployment compiler version is too low and no risk has been found
pragma solidity ^0.4.11;
/**
```

```
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/179
 */
contract ERC20Basic {
  uint256 public totalSupply;
  function balanceOf(address who) constant returns (uint256);
  function transfer(address to, uint256 value) returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}
/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
  address public owner;
  /**
   * @dev The Ownable constructor sets the original `owner` of the contract to the sender
   * account.
   */
  function Ownable() {
    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 transfer control of the contract to a newOwner.
   * @param newOwner The address to transfer ownership to.
   */
  function transferOwnership(address newOwner) onlyOwner {
    if (newOwner != address(0)) { //SlowMist// This check is quite good in avoiding losing control of
the contract caused by user mistakes
      owner = newOwner;
    }
```

```
  }
}
// Temporarily have SafeMath here until all contracts have been migrated to SafeMathLib version from OpenZeppelin
/**
 * Math operations with safety checks
 */
```

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

```
contract SafeMath {
  function safeMul(uint a, uint b) internal returns (uint) {
    uint c = a * b;
```

assert(a == 0 || c / a == b); //SlowMist// It is recommended to replace "assert" with "require" to optimize Gas

```
    return c;
  }
  function safeDiv(uint a, uint b) internal returns (uint) {
    assert(b > 0);
    uint c = a / b;
```

assert(a == b * c + a % b); //SlowMist// It is recommended to replace "assert" with "require" to optimize Gas

```
    return c;
  }
  function safeSub(uint a, uint b) internal returns (uint) {
```

assert(b <= a); //SlowMist// It is recommended to replace "assert" with "require" to optimize Gas

```
    return a - b;
  }
  function safeAdd(uint a, uint b) internal returns (uint) {
    uint c = a + b;
```

assert(c>=a && c>=b); //SlowMist// It is recommended to replace "assert" with "require" to optimize Gas

```
    return c;
  }
  function max64(uint64 a, uint64 b) internal constant returns (uint64) {
```

```
    return a >= b ? a : b;
  }
  function min64(uint64 a, uint64 b) internal constant returns (uint64) {
    return a < b ? a : b;
  }
  function max256(uint256 a, uint256 b) internal constant returns (uint256) {
    return a >= b ? a : b;
  }
  function min256(uint256 a, uint256 b) internal constant returns (uint256) {
    return a < b ? a : b;
  }
}
/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
contract ERC20 is ERC20Basic {
  function allowance(address owner, address spender) constant returns (uint256);
  function transferFrom(address from, address to, uint256 value) returns (bool);
  function approve(address spender, uint256 value) returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}
/**
 * Standard ERC20 token with Short Hand Attack and approve() race condition mitigation.
 *
 * Based on code by FirstBlood:
 * https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
 */
contract StandardToken is ERC20, SafeMath {
  /* Token supply got increased and a new owner received these tokens */
  event Minted(address receiver, uint amount);
  /* Actual balances of token holders */
```

```
mapping(address => uint) balances;
/* approve() allowances */
mapping (address => mapping (address => uint)) allowed;
/* Interface declaration */
function isToken() public constant returns (bool weAre) {
  return true;
}
function transfer(address _to, uint _value) returns (bool success) {
  //SlowMist// require(_to != address(0));

  //SlowMist// It is recommended to add to code above, avoiding user mistake leading to the
loss of token during transfer
  balances[msg.sender] = safeSub(balances[msg.sender], _value);
  balances[_to] = safeAdd(balances[_to], _value);
  Transfer(msg.sender, _to, _value);
  return true; //SlowMist// The return value conforms to the EIP20 specification
}
function transferFrom(address _from, address _to, uint _value) returns (bool success) {
  //SlowMist// require(_value <= _allowance[_from][msg.sender]);

  //SlowMist// It is recommended to add to code above, can optimize Gas
  uint _allowance = allowed[_from][msg.sender];
  balances[_to] = safeAdd(balances[_to], _value);
  balances[_from] = safeSub(balances[_from], _value);
  allowed[_from][msg.sender] = safeSub(_allowance, _value);
  Transfer(_from, _to, _value);
  return true; //SlowMist// The return value conforms to the EIP20 specification
}
function balanceOf(address _owner) constant returns (uint balance) {
  return balances[_owner];
}
function approve(address _spender, uint _value) returns (bool success) {
  // To change the approve amount you first have to reduce the addresses`
  //  allowance to zero by calling `approve(_spender, 0)` if it is not
  //  already 0 to mitigate the race condition described here:
  //  https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
  if ((_value != 0) && (allowed[msg.sender][_spender] != 0)) throw;
```

```
    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);

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

  }
  function allowance(address _owner, address _spender) constant returns (uint remaining) {
    return allowed[_owner][_spender];
  }
}
/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
/**
 * Upgrade agent interface inspired by Lunyr.
 *
 * Upgrade agent transfers tokens to a new contract.
 * Upgrade agent itself can be the token contract, or just a middle man contract doing the heavy lifting.
 */
contract UpgradeAgent {
  uint public originalSupply;
  /** Interface marker */
  function isUpgradeAgent() public constant returns (bool) {
    return true;
  }
  function upgradeFrom(address _from, uint256 _value) public;
}
/**
 * A token upgrade mechanism where users can opt-in amount of tokens to the next smart contract revision.
 *
 * First envisioned by Golem and Lunyr projects.
 */
contract UpgradeableToken is StandardToken {
  /** Contract / person who can set the upgrade path. This can be the same as team multisig wallet, as what it is with its
default value. */
  address public upgradeMaster;
```

```
/** The next contract where the tokens will be migrated. */
UpgradeAgent public upgradeAgent;
/** How many tokens we have upgraded by now. */
uint256 public totalUpgraded;
/**
 * Upgrade states.
 *
 * - NotAllowed: The child contract has not reached a condition where the upgrade can bgun
 * - WaitingForAgent: Token allows upgrade, but we don't have a new agent yet
 * - ReadyToUpgrade: The agent is set, but not a single token has been upgraded yet
 * - Upgrading: Upgrade agent is set and the balance holders can upgrade their tokens
 *
 */
enum UpgradeState {Unknown, NotAllowed, WaitingForAgent, ReadyToUpgrade, Upgrading}
/**
 * Somebody has upgraded some of his tokens.
 */
event Upgrade(address indexed _from, address indexed _to, uint256 _value);
/**
 * New upgrade agent available.
 */
event UpgradeAgentSet(address agent);
/**
 * Do not allow construction without upgrade master set.
 */
function UpgradeableToken(address _upgradeMaster) {
   upgradeMaster = _upgradeMaster;
}
/**
 * Allow the token holder to upgrade some of their tokens to a new contract.
 */
function upgrade(uint256 value) public {
    UpgradeState state = getUpgradeState();
    if(!(state == UpgradeState.ReadyToUpgrade || state == UpgradeState.Upgrading)) {
      // Called in a bad state
      throw;
    }
    // Validate input value.
    if (value == 0) throw;
    balances[msg.sender] = safeSub(balances[msg.sender], value);
    // Take tokens out from circulation
    totalSupply = safeSub(totalSupply, value);
```

```
        totalUpgraded = safeAdd(totalUpgraded, value);
        // Upgrade agent reissues the tokens
        upgradeAgent.upgradeFrom(msg.sender, value);
        Upgrade(msg.sender, upgradeAgent, value);
    }
    /**
     * Set an upgrade agent that handles
     */
    function setUpgradeAgent(address agent) external {
        if(!canUpgrade()) {
            // The token is not yet in a state that we could think upgrading
            throw;
        }
        if (agent == 0x0) throw;
        // Only a master can designate the next agent
        if (msg.sender != upgradeMaster) throw;
        // Upgrade has already begun for an agent
        if (getUpgradeState() == UpgradeState.Upgrading) throw;
        upgradeAgent = UpgradeAgent(agent);
        // Bad interface
        if(!upgradeAgent.isUpgradeAgent()) throw;
        // Make sure that token supplies match in source and target
        if (upgradeAgent.originalSupply() != totalSupply) throw;
        UpgradeAgentSet(upgradeAgent);
    }
    /**
     * Get the state of the token upgrade.
     */
    function getUpgradeState() public constant returns(UpgradeState) {
      if(!canUpgrade()) return UpgradeState.NotAllowed;
      else if(address(upgradeAgent) == 0x00) return UpgradeState.WaitingForAgent;
      else if(totalUpgraded == 0) return UpgradeState.ReadyToUpgrade;
      else return UpgradeState.Upgrading;
    }
    /**
     * Change the upgrade master.
     *
     * This allows us to set a new owner for the upgrade mechanism.
     */
    function setUpgradeMaster(address master) public {
        if (master == 0x0) throw;
        if (msg.sender != upgradeMaster) throw;
```

```
        upgradeMaster = master;
    }
    /**
     * Child contract can enable to provide the condition when the upgrade can begun.
     */
    function canUpgrade() public constant returns(bool) {
        return true;
    }
}
/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
/**
 * Define interface for releasing the token transfer after a successful crowdsale.
 */
contract ReleasableToken is ERC20, Ownable {
    /* The finalizer contract that allows unlift the transfer limits on this token */
    address public releaseAgent;
    /** A crowdsale contract can release us to the wild if ICO success. If false we are are in transfer lock up period.*/
    bool public released = false;
    /** Map of agents that are allowed to transfer tokens regardless of the lock down period. These are crowdsale contracts and possible the team multisig itself. */
    mapping (address => bool) public transferAgents;
    /**
     * Limit token transfer until the crowdsale is over.
     *
     */
    modifier canTransfer(address _sender) {
        if(!released) {
            if(!transferAgents[_sender]) {
                throw;
            }
        }
        _;
    }
    /**
     * Set the contract that can call release and make the token transferable.
     *
     * Design choice. Allow reset the release agent to fix fat finger mistakes.
     */
```

```
  function setReleaseAgent(address addr) onlyOwner inReleaseState(false) public {
    // We don't do interface check here as we might want to a normal wallet address to act as a release agent
    releaseAgent = addr;
  }
  /**
   * Owner can allow a particular address (a crowdsale contract) to transfer tokens despite the lock up period.
   */
  function setTransferAgent(address addr, bool state) onlyOwner inReleaseState(false) public {
    transferAgents[addr] = state;
  }
  /**
   * One way function to release the tokens to the wild.
   *
   * Can be called only from the release agent that is the final ICO contract. It is only called if the crowdsale has been success
(first milestone reached).
   */
  function releaseTokenTransfer() public onlyReleaseAgent {
    released = true;
  }
  /** The function can be called only before or after the tokens have been releasesd */
  modifier inReleaseState(bool releaseState) {
    if(releaseState != released) {
        throw;
    }
    _;
  }
  /** The function can be called only by a whitelisted release agent. */
  modifier onlyReleaseAgent() {
    if(msg.sender != releaseAgent) {
        throw;
    }
    _;
  }
  function transfer(address _to, uint _value) canTransfer(msg.sender) returns (bool success) {
    // Call StandardToken.transfer()
   return super.transfer(_to, _value);
  }
  function transferFrom(address _from, address _to, uint _value) canTransfer(_from) returns (bool success) {
    // Call StandardToken.transferForm()
    return super.transferFrom(_from, _to, _value);
  }
}
```

```
/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
/**
 * Safe unsigned safe math.
 *
 * https://blog.aragon.one/library-driven-development-in-solidity-2bebcaf88736#.750gwtwli
 *
 * Originally from https://raw.githubusercontent.com/AragonOne/zeppelin-solidity/master/contracts/SafeMathLib.sol
 *
 * Maintained here until merged to mainline zeppelin-solidity.
 *
 */
library SafeMathLib {
  function times(uint a, uint b) returns (uint) {
    uint c = a * b;

    assert(a == 0 || c / a == b); //SlowMist// It is recommended to replace "assert" with "require" to optimize Gas

    return c;
  }
  function minus(uint a, uint b) returns (uint) {

    assert(b <= a); //SlowMist// It is recommended to replace "assert" with "require" to optimize Gas

    return a - b;
  }
  function plus(uint a, uint b) returns (uint) {

    uint c = a + b; //SlowMist// It is recommended to replace "assert" with "require" to optimize Gas

    assert(c>=a);
```

```
    return c;
  }
}
/**
 * A token that can increase its supply by another contract.
 *
 * This allows uncapped crowdsale by dynamically increasing the supply when money pours in.
 * Only mint agents, contracts whitelisted by owner, can mint new tokens.
 *
 */
contract MintableToken is StandardToken, Ownable {
  using SafeMathLib for uint;
  bool public mintingFinished = false;
  /** List of agents that are allowed to create new tokens */
  mapping (address => bool) public mintAgents;
  event MintingAgentChanged(address addr, bool state  );
  /**
   * Create new tokens and allocate them to an address..
   *
   * Only callably by a crowdsale contract (mint agent).
   */
  function mint(address receiver, uint amount) onlyMintAgent canMint public {
    totalSupply = totalSupply.plus(amount);
    balances[receiver] = balances[receiver].plus(amount);
    // This will make the mint transaction apper in EtherScan.io
    // We can remove this after there is a standardized minting event
    Transfer(0, receiver, amount);
  }
  /**
   * Owner can allow a crowdsale contract to mint new tokens.
   */
  function setMintAgent(address addr, bool state) onlyOwner canMint public {
    mintAgents[addr] = state;
    MintingAgentChanged(addr, state);
  }
  modifier onlyMintAgent() {
    // Only crowdsale contracts are allowed to mint new tokens
    if(!mintAgents[msg.sender]) {
        throw;
    }
    _;
  }
```

```
  /** Make sure we are not done yet. */
  modifier canMint() {
    if(mintingFinished) throw;
    _;
  }
}
/**
 * A crowdsaled token.
 *
 * An ERC-20 token designed specifically for crowdsales with investor protection and further development path.
 *
 * - The token transfer() is disabled until the crowdsale is over
 * - The token contract gives an opt-in upgrade path to a new contract
 * - The same token can be part of several crowdsales through approve() mechanism
 * - The token can be capped (supply set in the constructor) or uncapped (crowdsale contract can mint new tokens)
 *
 */
contract CrowdsaleToken is ReleasableToken, MintableToken, UpgradeableToken {
  /** Name and symbol were updated. */
  event UpdatedTokenInformation(string newName, string newSymbol);
  string public name;
  string public symbol;
  uint public decimals;
  /**
   * Construct the token.
   *
   * This token must be created through a team multisig wallet, so that it is owned by that wallet.
   *
   * @param _name Token name
   * @param _symbol Token symbol - should be all caps
   * @param _initialSupply How many tokens we start with
   * @param _decimals Number of decimal places
   * @param _mintable Are new tokens created over the crowdsale or do we distribute only the initial supply? Note that
when the token becomes transferable the minting always ends.
   */
  function CrowdsaleToken(string _name, string _symbol, uint _initialSupply, uint _decimals, bool _mintable)
    UpgradeableToken(msg.sender) {
    // Create any address, can be transferred
    // to team multisig via changeOwner(),
    // also remember to call setUpgradeMaster()
    owner = msg.sender;
    name = _name;
```

```
    symbol = _symbol;
    totalSupply = _initialSupply;
    decimals = _decimals;
    // Create initially all balance on the team multisig
    balances[owner] = totalSupply;
    if(totalSupply > 0) {
      Minted(owner, totalSupply);
    }
    // No more new supply allowed after the token creation
    if(!_mintable) {
      mintingFinished = true;
      if(totalSupply == 0) {
        throw; // Cannot create a token without supply and no minting
      }
    }
}
/**
  * When token is released to be transferable, enforce no new tokens can be created.
  */
function releaseTokenTransfer() public onlyReleaseAgent {
    mintingFinished = true;
    super.releaseTokenTransfer();
}
/**
  * Allow upgrade agent functionality kick in only if the crowdsale was success.
  */
function canUpgrade() public constant returns(bool) {
    return released && super.canUpgrade();
}
/**
  * Owner can update token information here.
  *
  * It is often useful to conceal the actual token association, until
  * the token operations, like central issuance or reissuance have been completed.
  *
  * This function allows the token owner to rename the token after the operations
  * have been completed and then point the audience to use the token contract.
  */
function setTokenInformation(string _name, string _symbol) onlyOwner {
    name = _name;
    symbol = _symbol;
    UpdatedTokenInformation(name, symbol);
```

```
  }
}
/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
contract BurnableToken is StandardToken {
  address public constant BURN_ADDRESS = 0;
  /** How many tokens we burned */
  event Burned(address burner, uint burnedAmount);
  /**
   * Burn extra tokens from a balance.
   *
   */
  function burn(uint burnAmount) {
    address burner = msg.sender;
    balances[burner] = safeSub(balances[burner], burnAmount);
    totalSupply = safeSub(totalSupply, burnAmount);
    Burned(burner, burnAmount);
  }
}
contract GetToken is CrowdsaleToken, BurnableToken {
    function GetToken() CrowdsaleToken(
            "Guaranteed Entrance Token",
            "GET",
            0,   // We don't want to have initial supply
            18,
            true // Mintable
        )
    {}
}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account