慢雾科技 SLOWMIST ZONE TOKEN 智能合约审计实录

> 全文共 6500 字，15 张图片，预计阅读时间 2077s

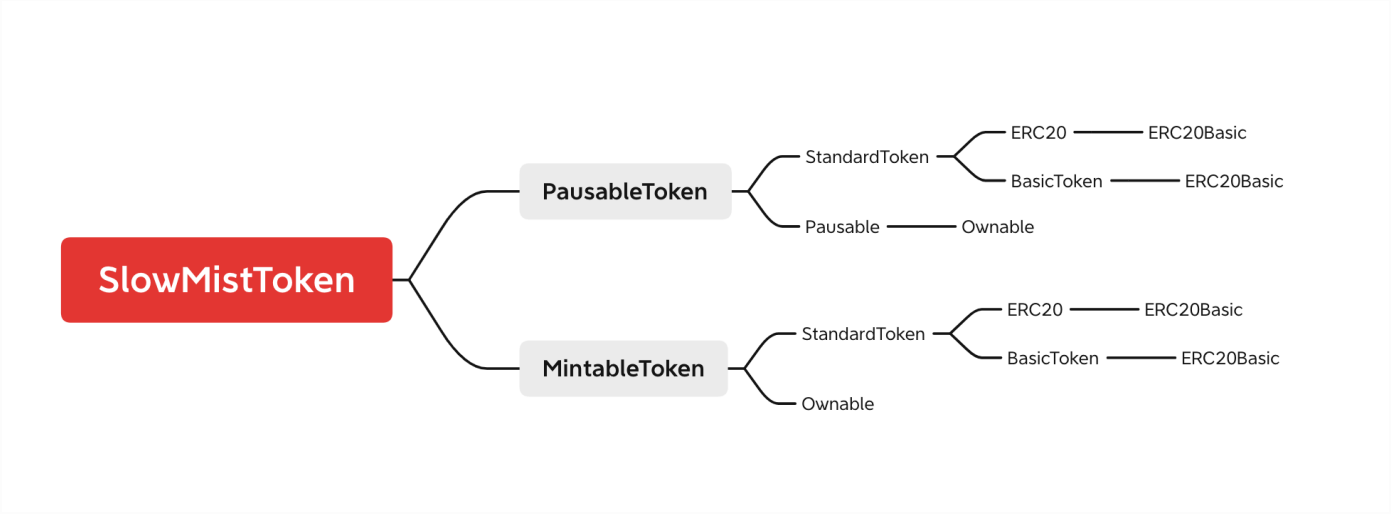
- 作者: xxxeyJ
- 作者博客: tricksongs.com
- 知识星球: 区块危机(BLOCKCRISIS)

前言

> 最近我搞了一个小蜜圈，星球主攻区块链安全方向，但是目前由于圈内的小伙伴扎根在传统安全行业久已，对于区块链安全没有进一步的了解(经过调研发现大多数小伙伴只停留在了晓得区块链，比特币等几个基础概念)，在此之前发在小蜜圈的文章也并没有很好的带动学习气氛，我想主要原因是由于小伙伴们对区块链安全不甚了解才导致了这幅局面，于是本文便应运而生，由于本文是出于教学的目的编写，故内容较为细化，适合对区块链安全感兴趣但目前处于新手村并且可以静下心来阅读学习的小伙伴食用，本文将会围绕着遵循 ***附录1*** `ERC20` 标准的 SlowMist Zone Token 展开审计，希望本文的诞生会对后续小蜜圈以及各位小伙伴在区块链这一领域的发展起到良好的铺垫作用

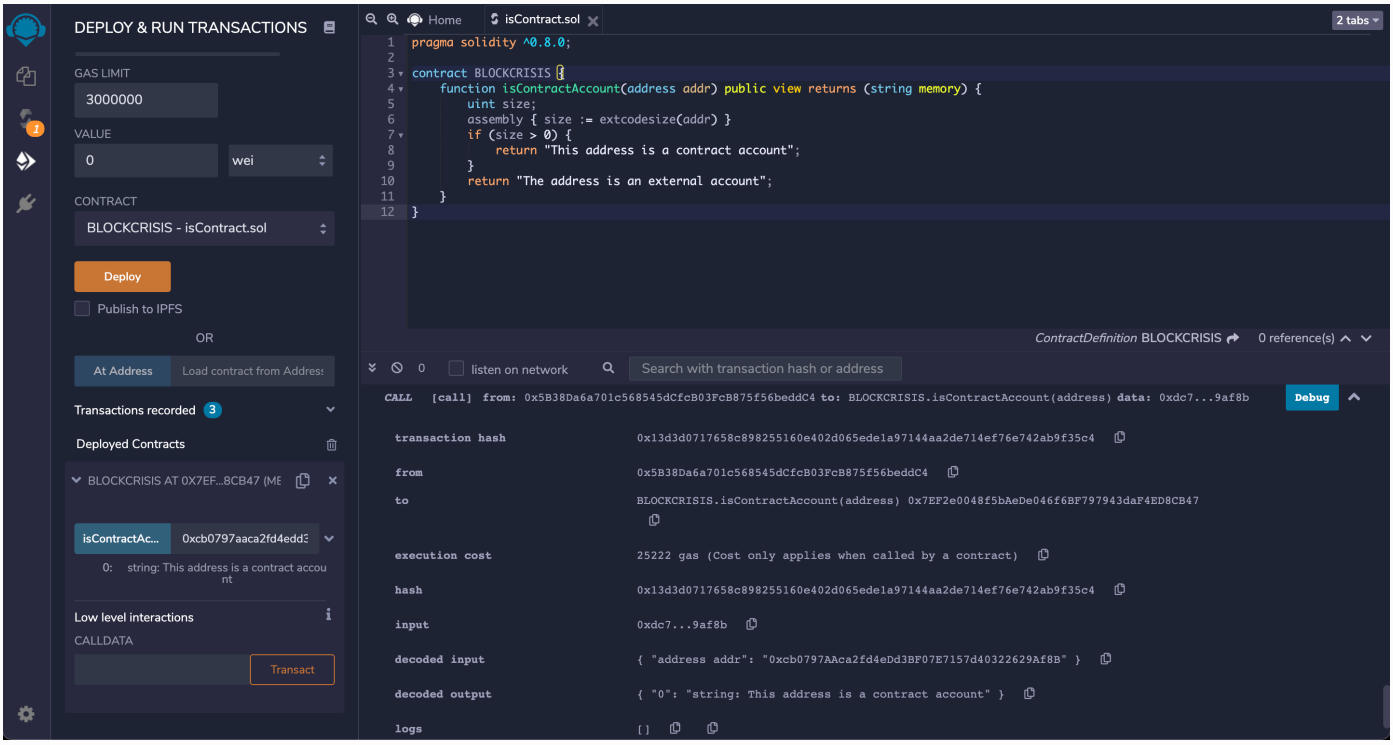
前期准备工作

> - **SlowMist Zone Token 智能合约流程图**



审计智能合约之前，需要先对审计对象有个基础了解，比如这里的慢雾区通证

Smart Contract: 0xcb0797aaca2fd4edd3bf07e7157d40322629af8b （Contract Account）



Holders: 99 (持币者数量)

Max Total Supply: 102,400,000 (最大发行总量)

Decimals: 18（精度）

Name: SlowMist Zone Token （Token 全称）

Symbol: SLOWMIST （Token 符号）

Owner Account: 0x06a3f099e75720fd7415f87edc8cd9953b36d171 (Ownership) ***附录2***

CODE REVIEW

`SlowMist Zone Token` 采用单源文件多合约的形式部署智能合约，`.sol` 源文件内部合计 **共 9 个 Smart Contract**，其在源文件内实现 `SafeMath Library`

Line 5 标识使用 Solidity 编译器版本号不低于 `0.4.23`，且不高于下个大版本(`0.5.0`)

```
1  /**
2   *Submitted for verification at Etherscan.io on 2019-06-11
3  */
4
5  pragma solidity ^0.4.23;
6
```

Line 12 - Line 52 在源文件内部实现了 `SafeMath Library`，将其作用于智能合约内部的数学运算以防止整型溢出漏洞的产生，在 SlowMist Zone Token 内部使用到的 `SafeMath Library` 只有 `sub()` 以及 `add()` 这两个 `function`，我想这也是为什么 `div()` 功能代码被注释掉的原因，这里值得一提的是，*附录3* `Solidity ^0.8.0` 版本之后编译器内部默认集成了 `SafeMath Library`，因此，在后续的版本直接使用 `Solidity 默认的数学运算符` 即可避免整型溢出漏洞的产生

```
12  library SafeMath {
13
14    /**
15    * @dev Multiplies two numbers, throws on overflow.
16    */
17    function mul(uint256 a, uint256 b) internal pure returns (uint256 c) {
18      if (a == 0) {
19        return 0;
20      }
21      c = a * b;
22      assert(c / a == b);
23      return c;
24    }
25
26    /**
27    * @dev Integer division of two numbers, truncating the quotient.
28    */
29    function div(uint256 a, uint256 b) internal pure returns (uint256) {
30      // assert(b > 0); // Solidity automatically throws when dividing by 0
31      // uint256 c = a / b;
32      // assert(a == b * c + a % b); // There is no case in which this doesn't hold
33      return a / b;
34    }
35
36    /**
37    * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than minuend).
38    */
39    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
40      assert(b <= a);
41      return a - b;
42    }
43
44    /**
45    * @dev Adds two numbers, throws on overflow.
46    */
47    function add(uint256 a, uint256 b) internal pure returns (uint256 c) {
48      c = a + b;
49      assert(c >= a);
50      return c;
51    }
52  }
```

Line 59 - Line 64 实现了一个更为精简的 `ERC20` 标准，其中声明了 **3 个 function**，以及定义了一个 `Transfer` 事件，Line 70 `BasicToken` 子合约继承 `ERC20Basic` 父合约，Line 71 表示将 `SafeMath Library` 中的 `function` 应用于 `uint256` 整数类型，Line 73 构造一个名为 `balances` 的 `映射(Mapping)`，在当前智能合约中给它的定位是将此映射表示为指定账户持有的 `Token` 数量，Line 75 定义了一个 `totalSupply_` 状态变量，在当前 Token 中，使用到该状态变量的功能共有三处，其一是 Line 80 的 `totalSupply()` function，可见性为 `public`，表示可公开调用，函数返回值为 `totalSupply_`，用以获取该 Token 中的发行总量，其二被作用在了 `MintableToken` 合约内的 `mint()` function，用来与 `tmpTotal` 变量联动判断当前已发行的代币数量，以确保其发行数量不超过最大发行总量，其三是这个变量

`(最大发行总量)` 会在后续主合约中的构造函数中定义下来，Line 89 - Line 97 实现了 `Transfer` 转账功能，检查 `to` 接收者地址不为 `0x00` 并检查转账金额小于等于发起人余额，通过两步检查后，使用 `SafeMath` 的 `sub()` function 给调用者 `uint256 balances(address(msg.sender)` 扣除转账金额，然后使用 `SafaMath` 的 `add` function 给 `to` 目标地址增加相应的转账金额(Token)，而后触发 `Transfer` 事件，返回 `True` 表明调用成功

```
53
54  /**
55   * @title ERC20Basic
56   * @dev Simpler version of ERC20 interface
57   * @dev see https://github.com/ethereum/EIPs/issues/179
58   */
59  contract ERC20Basic {
60    function totalSupply() public view returns (uint256);
61    function balanceOf(address who) public view returns (uint256);
62    function transfer(address to, uint256 value) public returns (bool);
63    event Transfer(address indexed from, address indexed to, uint256 value);
64  }
65
66  /**
67   * @title Basic token
68   * @dev Basic version of StandardToken, with no allowances.
69   */
70  contract BasicToken is ERC20Basic {
71    using SafeMath for uint256;
72
73    mapping(address => uint256) balances;
74
75    uint256 totalSupply_;
76
77    /**
78    * @dev total number of tokens in existence
79    */
80    function totalSupply() public view returns (uint256) {
81      return totalSupply_;
82    }
83
84    /**
85    * @dev transfer token for a specified address
86    * @param _to The address to transfer to.
87    * @param _value The amount to be transferred.
88    */
89    function transfer(address _to, uint256 _value) public returns (bool) {
90      require(_to != address(0));
91      require(_value <= balances[msg.sender]);
92
93      balances[msg.sender] = balances[msg.sender].sub(_value);
94      balances[_to] = balances[_to].add(_value);
95      emit Transfer(msg.sender, _to, _value);
96      return true;
97    }
```

通过 `balanceOf()` function 实现获取给定地址账户的余额(Token数量)

```
104     function balanceOf(address _owner) public view returns (uint256) {
105       return balances[_owner];
106     }
```

`ERC20` 合约继承自 `ERC20Basic` 合约，合约内部实现了三个方法，定义了一个 `Approval` 事件

```
109   /**
110    * @title ERC20 interface
111    * @dev see https://github.com/ethereum/EIPs/issues/20
112    */
113   contract ERC20 is ERC20Basic {
114     function allowance(address owner, address spender)
115       public view returns (uint256);
116
117     function transferFrom(address from, address to, uint256 value)
118       public returns (bool);
119
120     function approve(address spender, uint256 value) public returns (bool);
121     event Approval(
122       address indexed owner,
123       address indexed spender,
124       uint256 value
125     );
126   }
```

StandardToken 这一合约实现了标准化的 `ERC20` ，其继承自 **ERC20, BasicToken** 两个合约，合约内部定义了一个 `allowd` 映射(Mapping)，**allowd** 在此合约内表示 `_owner` 这一账户授权给 `_spender` 账户可使用的金额(Token)， 映射分为两部分，**mapping(keyType => valueType)**，其中 `keyType` 存在一定的限制，**keyType** 不能为 `映射` ，`变长数组` ，`合约` ，`枚举` ，`结构` ，而 `valueType` 则**无任何限制**，可以为任意类型，包括(Mapping)自身类型， `transferFrom()` 和 `approve()` 这两个方法需要组合使用，首先通过 **approve()** 授权指定账户(`_spender`)可以使用相应金额的 `Token` ， 然后被授权者通过 `transferFrom` 实现转账逻辑，而后给 `allowd[_from][msg.sender]` 减去相应已使用的金额，由于 `approve()` 在早期版本存在条件竞争的风险，OpenZeppelin 提出了一种解决方案 详情见 ***附录4***，慢雾在此沿用了这种方案，使其在 `Token` 内部实现了 `increaseApproval()` 用以追加支付授权额度，以及实现了 `decreaseApproval()` 用以撤销指定额度的支付授权，其中 `decreaseApproval()` 需要传递两个参数，被授权者地址以及想要撤销的金额，逻辑如下，将已授权的金额赋值给 `oldValue` 以便后续比对金额，如果想要削减的金额比被授权者持有

使用权的金额大的情况下，则将授权金额直接清零，否则，根据其(`_subtractedValue`)指定的撤销金额对被授权者减去相应的金额使用权，最后，触发 `ERC20` 合约内的 `Approval` 事件

```
128 /**
129  * @title Standard ERC20 token
130  *
131  * @dev Implementation of the basic standard token.
132  * @dev https://github.com/ethereum/EIPs/issues/20
133  * @dev Based on code by FirstBlood:
    https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
134  */
135 contract StandardToken is ERC20, BasicToken {
136
137   mapping (address => mapping (address => uint256)) internal allowed;
138
139   /**
140    * @dev Transfer tokens from one address to another
141    * @param _from address The address which you want to send tokens from
142    * @param _to address The address which you want to transfer to
143    * @param _value uint256 the amount of tokens to be transferred
144    */
145   function transferFrom(
146     address _from,
147     address _to,
148     uint256 _value
149   )
150     public
151     returns (bool)
152   {
153     require(_to != address(0));
154     require(_value <= balances[_from]);
155     require(_value <= allowed[_from][msg.sender]);
156
157     balances[_from] = balances[_from].sub(_value);
158     balances[_to] = balances[_to].add(_value);
159     allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
160     emit Transfer(_from, _to, _value);
161     return true;
162   }
163
164   /**
165    * @dev Approve the passed address to spend the specified amount of tokens on behalf of
    msg.sender.
166    *
167    * Beware that changing an allowance with this method brings the risk that someone may use both
    the old
168    * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate
    this
169    * race condition is to first reduce the spender's allowance to 0 and set the desired value
    afterwards:
170    * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
171    * @param _spender The address which will spend the funds.
172    * @param _value The amount of tokens to be spent.
173    */
174   function approve(address _spender, uint256 _value) public returns (bool) {
175     allowed[msg.sender][_spender] = _value;
176     emit Approval(msg.sender, _spender, _value);
177     return true;
178   }
179
```

```
  /**
   * @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;
  }

}
```

Line 255 实现了 **Ownable Library**，其中定义了一个名为 `owner` 的状态变量，表明合约所有者(Ownership)，定义了两个事件， `OwnershipRenounced` 以及 `OwnershipTransferred` ，尽管合约内并没有写关于放弃所有权的代码，紧接着是一个构造函数，用于初始化 `Ownable` 合约，将调用者 **address(msg.sender)** 赋值给 owner 这一状态变量，下面是一个修饰器(modifier)，在当前 Token 中用于校验调用者是否为所有者身份 (`Ownership`)， `transferOwnership()` 方法用于移交所有权

```
250 /**
251  * @title Ownable
252  * @dev The Ownable contract has an owner address, and provides basic authorization control
253  * functions, this simplifies the implementation of "user permissions".
254  */
255 contract Ownable {
256   address public owner;
257
258   event OwnershipRenounced(address indexed previousOwner);
259   event OwnershipTransferred(
260     address indexed previousOwner,
261     address indexed newOwner
262   );
263
264
265   /**
266    * @dev The Ownable constructor sets the original `owner` of the contract to the sender
267    * account.
268    */
269   constructor() public {
270     owner = msg.sender;
271   }
272
273   /**
274    * @dev Throws if called by any account other than the owner.
275    */
276   modifier onlyOwner() {
277     require(msg.sender == owner);
278     _;
279   }
280
281   /**
282    * @dev Allows the current owner to transfer control of the contract to a newOwner.
283    * @param newOwner The address to transfer ownership to.
284    */
285   function transferOwnership(address newOwner) public onlyOwner {
286     require(newOwner != address(0));
287     emit OwnershipTransferred(owner, newOwner);
288     owner = newOwner;
289   }
290 }
```

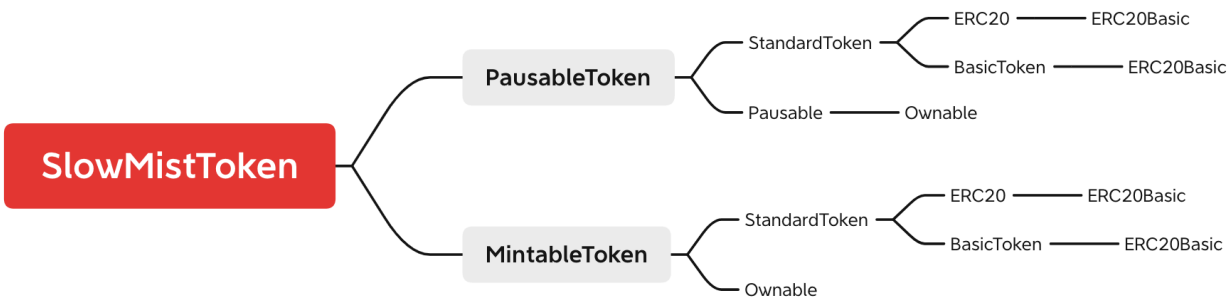
`MintableToken` 是一个专门用于铸币的合约，继承自 `StandardToken`， `Ownable` 这两个合约，继承 Ownable 合约的原因是为了使用其 `owner` 状态变量以便于确认所有者身份，**mint()** 使用了 `hasMintPermission` 和 `canMint` 修饰器加以修饰， `hasMintPermission` 用于确认铸币者是否为 `owner` 账户， `canMint` 用于检查是否允许铸币，Line 301 定义了一个名为 `mintingFinished` 的 bool 类型的状态变量，其值为 `false`，修饰器内进行了 `!` 取反操作，则通过检查，函数体内首先将当前已铸币量和待铸币数量相加并赋值给一个临时变量以便检验发行量是否小于 `1.024 亿` 这个数字（从这点可以看出此Token不可增发！），条件判断成立后给 `mintTotal` 变量增加发行的金额以及给指定账户铸造了指定 `Token` 数量，最后，触发两个事件以确认成功铸币

```
292 /**
293  * @title Mintable token
294  * @dev Simple ERC20 Token example, with mintable token creation
295  * @dev Issue: * https://github.com/OpenZeppelin/openzeppelin-solidity/issues/120
296  * Based on code by TokenMarketNet:
    https://github.com/TokenMarketNet/ico/blob/master/contracts/MintableToken.sol
297  */
298 contract MintableToken is StandardToken, Ownable {
299   event Mint(address indexed to, uint256 amount);
300
301   bool public mintingFinished = false;
302   uint public mintTotal = 0;
303
304   modifier canMint() {
305     require(!mintingFinished);
306     _;
307   }
308
309   modifier hasMintPermission() {
310     require(msg.sender == owner);
311     _;
312   }
313
314   /**
315    * @dev Function to mint tokens
316    * @param _to The address that will receive the minted tokens.
317    * @param _amount The amount of tokens to mint.
318    * @return A boolean that indicates if the operation was successful.
319    */
320   function mint(
321     address _to,
322     uint256 _amount
323   )
324     hasMintPermission
325     canMint
326     public
327     returns (bool)
328   {
329     uint tmpTotal = mintTotal.add(_amount);
330     require(tmpTotal <= totalSupply_);
331     mintTotal = mintTotal.add(_amount);
332     balances[_to] = balances[_to].add(_amount);
333     emit Mint(_to, _amount);
334     emit Transfer(address(0), _to, _amount);
335     return true;
336   }
337 }
```

Pausable 继承自 `Ownable` Contract，合约内部定义了两个用于暂停/取消暂停的事件，Line 348 定义了一个名为 `paused` 的 bool 类型状态变量，可见性为 `public`，此合约将该变量作用于 `whenNotPaused` 和 `whenPaused` 修饰器，`paused` 初始状态为 `true`，表示为暂停状态，这时只能够调用 `unpause()` 以取消其暂停状态，而后 `paused` 变量被赋值为 `false`，这时则可以通过由 `whenNotPaused` 修饰的 `pause()` 方法将其暂停

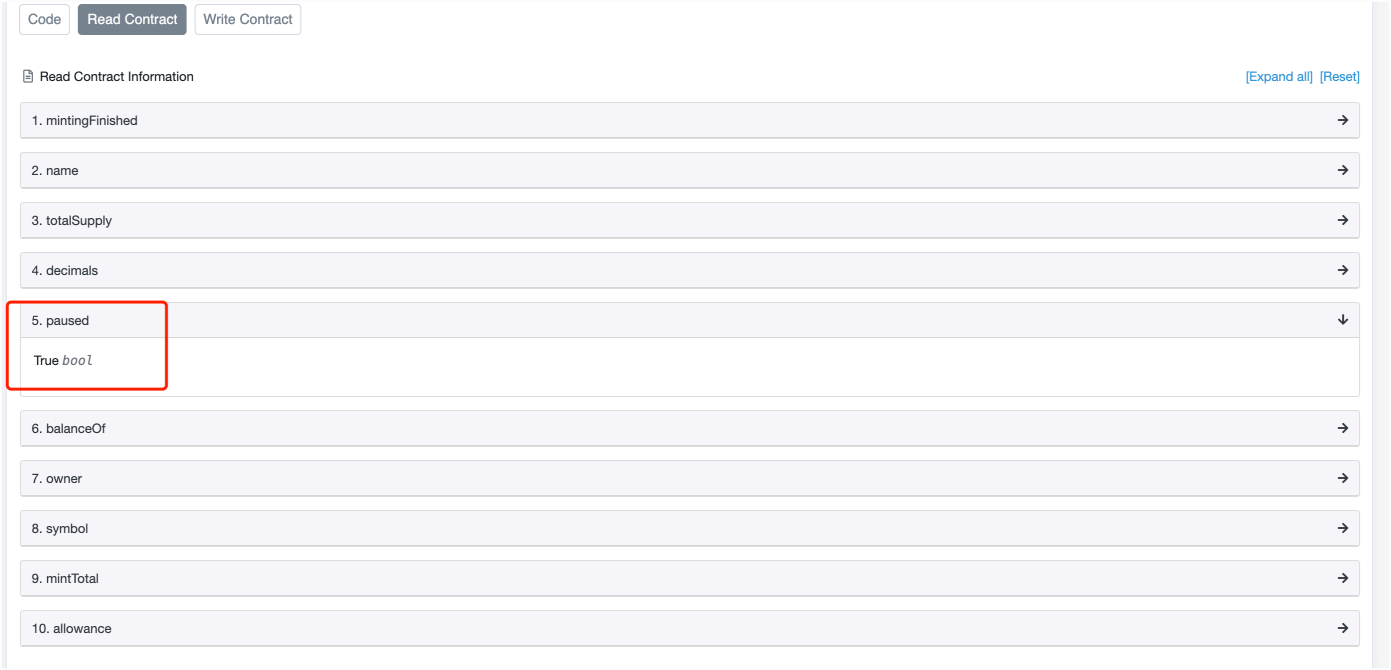
```
340 /**
341  * @title Pausable
342  * @dev Base contract which allows children to implement an emergency stop mechanism.
343  */
344 contract Pausable is Ownable {
345   event Pause();
346   event Unpause();
347
348   bool public paused = true;
349
350
351   /**
352    * @dev Modifier to make a function callable only when the contract is not paused.
353    */
354   modifier whenNotPaused() {
355     require(!paused);
356     _;
357   }
358
359   /**
360    * @dev Modifier to make a function callable only when the contract is paused.
361    */
362   modifier whenPaused() {
363     require(paused);
364     _;
365   }
366
367   /**
368    * @dev called by the owner to pause, triggers stopped state
369    */
370   function pause() onlyOwner whenNotPaused public {
371     paused = true;
372     emit Pause();
373   }
374
375   /**
376    * @dev called by the owner to unpause, returns to normal state
377    */
378   function unpause() onlyOwner whenPaused public {
379     paused = false;
380     emit Unpause();
381   }
382 }
```

PausableToken 继承自 `StandardToken, Pausable`，使用到了 `Pausable` 的修饰器 `whenNotPaused`，`whenNotPaused` 修饰器 `!paused` 执行取反操作则为 `false`，得出 Token 目前处于暂停状态，可以将 **super.function** 关键词理解为继承关系的最上游，比如说 Line 399 的 `return super.transfer(_to, _value)`，调用到的 `function()` 来自于最后一个实现此方法的合约，继承关系如下：




```
/**
 * @title Pausable token
 * @dev StandardToken modified with pausable transfers.
 **/
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);
  }
}
```



最后是 **SlowMistToken** 这一主合约，定义了三个用于标识 `Token` 信息的状态变量，以及 Line 455 的构造函数设定最大发行总量为 **102,400,000** 枚 `Token`，Line 458 的 `fallback()` 回退函数不可用于接收 ETH，强行向该合约账户转账将会导致状态回滚

```
448 contract SlowMistToken is PausableToken, MintableToken {
449     // public variables
450     string public name = "SlowMist Zone Token";
451     string public symbol = "SLOWMIST";
452     uint8 public decimals = 18;
453
454     constructor() public {
455         totalSupply_ = 102400000 * (10 ** uint256(decimals));
456     }
457
458     function () public payable {
459         revert();
460     }
461 }
```

后记

总体看下来，慢雾科技的 `SlowMist Zone Token` 在合约代码满足了其业务需求的同时尽可能将攻击面降至最低，就目前市面上公开的攻击手法来讲，慢雾区 `Token` 的安全性毋庸置疑，回归审计文章原题，在我看来，代码层面没有漏洞，不代表其它层面没有问题，漫雾科技给 `SlowMist Zone Token` 的定位是**社区激励代币**，它没有在二级市场流动所以不具备金融属性，即便劫持了该 Token 的 `Ownership` 账户 ，在我看来也做不了什么大的文章，目前来讲，该 Token 直面的威胁主要有两点，一个是防范 `Ownership` 账户沦陷的风险(比如说通过钓鱼等手段劫持其 `Owner` 账户)，另一个则是目前藏匿于区块链生态中那片光亮照不到的 `0day`

总结

近年来，区块链生态安全形势愈演愈烈，区块链自身的金融属性是其它行业所不具备的，据统计加密货币市值已超过 1.66 万亿美金$，区块链中的链上安全形势尤为严峻，且伴随着 `DApp`(**去中心化应用**)， `DeFi`(**去中心化金融**)， `NFT`(**非同质化代币**) 等底层架构依附于区块链技术的新兴概念诞生，区块链也将会给一众从业者们带来前所未有的机遇，如果你对区块链安全感兴趣，并且想要加入知识星球 **区块危机** 共同学习，可以扫描下方二维码添加我的微信，星球主体内容为当下较为冷门的区块链安全(Blockchain Security)，相信加入本星球的小伙伴们能够有所收获

知识星球

xxxeyJ

送你一张星球优惠券

立减

¥ 50

可用于

区块危机

2021/08/05 12:00 至 2022/01/01 00:00

前 233 名加入可用 ▸

长按二维码立抢优惠

附录

- 1) ERC20 Token Standard - ERC20 是当前最为常见的 Token 标准
- 2) Ownership - OpenZeppelin 的 Ownable Library 是当下最为流行的实现访问控制的 Library
- 3) Solidity v0.8.0 Changes - Solidity v0.8.0 更新日志
- 4) 条件竞争概述 - Approve() 存在的事务顺序依赖性问题解决方案