猎豹区块链安全

智能合约审计报告

Bonus Cloud Token（BxC）

审计结果：通过

版本说明

修订人	修订内容	修订时间	版本号	审阅人
王海龙	编写文档	2018 年 9 月 13 日	V1.0	杨文玉

文档信息

文档名称	文档版本号	文档编号	保密级别
Bonus Cloud Token 智能合约审计报告	V1.0	0x53a94403	保密

版权声明

目录

一、 综述

1.1 审计背景

Cheetah Mobile Security 团队于**2018年9月13日** 对 **Bonus Cloud Token（BxC）项目**的智能合约进行了安全审计。本报告详细阐述了审计细节与结果。（**声明**：Cheetah Mobile Security 仅根据本报告出具前已经发生或存在的风险漏洞实情出具报告，并就此承担相应责任。对于报告出具以后发生或存在的事实，无法判断其对该项目造成的影响，Cheetah Mobile Security 并不对此承担责任。本报告所做的安全审计分析及其他内容，仅基于信息提供者截止本报告出具时向 Cheetah Mobile Security 提供的文件和资料出具报告，资料提供方有义务保证已提供资料不存在缺失、篡改、删减或隐瞒的情况。如果存在，Cheetah Mobile Security 对此而导致的损失和不利影响不承担任何责任。）

1.2 审计结果

审计结果	审计人	审阅人
通过	王海龙	杨文玉

1.3 项目基本信息

- **合约名称：**

BonusCloudToken

- **合约地址：**

  -

- **Etherscan 链接：**

  -

- **合约源码链接：**

  https://github.com/BonusCloud/bonuscloud-contracts/blob/master/BonusCloudToken.sol

- **合约类型：**

  代币型合约，代币名称为：Bonus Cloud Token（BxC）

- **合约上链日期：**

  -

二、 合约代码漏洞分析

2.1 合约代码漏洞等级

合约漏洞数量按等级统计如下：

高危漏洞	中危漏洞	低危漏洞
0	0	0

2.2 合约代码漏洞分布

合约漏洞按类型分布如下：

未发现漏洞

2.3 合约代码审计项及结果

（其他未知安全漏洞不包含在本次审计责任范围）

审计方法	审计大类	审计子类	审计结果
攻防类审计	算数安全审计	整数上溢审计	通过
		整数下溢审计	通过
		运算精度审计	通过
	竞态审计	重入攻击审计	通过
		事物顺序依赖审计	通过
	权限控制审计	权限漏洞审计	通过
		权限过大审计	通过
	安全设计审计	安全模块使用安全	通过
		编译器版本安全	通过
		硬编码地址安全	通过
		敏感函数（fallback/call/tx.origin）使用安全	通过
		函数返回值安全	通过

	拒绝服务审计	-	通过
	Gas 优化审计	-	通过
	设计逻辑审计	-	通过

三、 代码审计结果分析

3.1 算数安全审计

智能合约中的算数安全审计分为三部分：整数上溢审计、整数下溢审计以及运算精度设计。

3.1.1 整数上溢出审计【通过】

Solidity 最多能处理 256 位的数据。当最大数字增加会上溢。整数上溢如果发生在转账逻辑中，会使得转账资金数额运算错误，导致严重的资金风险。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无

3.1.2 整数下溢出审计【通过】

Solidity 最多能处理 256 位的数据。当最小数字减小会下溢。整数下溢如果发生在转账逻辑中，会使得转账资金数额运算错误，导致严重的资金风险。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无

3.1.3 运算精度审计【通过】

solidity 在做乘法、除法的运算的过程中，会进行类型强制转换。例如整数相除，若有余数，则会丢失精度。如果资金变量运算中包含了精度风险，则会导致用户转账逻辑错误，资金损失。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无

3.2 竞态竞争审计

智能合约中的竞态审计分为两部分：重入攻击审计以及事物顺序依赖审计。具有竞态竞争漏洞的合约中 ,攻击者可以在一定概率下通过调整程序或交易的执行过程，修改程序的输出结果。

3.2.1 重入攻击审计【通过】

重入攻击指当调用 call.value()函数发送 Ether 时候，攻击者使得函数发送 Ether 的操作发生在实际减少发送者账户的余额之前，从而导致异常转账。如经典的 THE DAO 攻击。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无

3.2.2 事物顺序依赖审计【通过】

事物顺序依赖指，在以太坊中，矿工在打包交易的时候，会根据交易的手续 gas 费用的高低进行排序。攻击者可以故意提高其攻击交易的 gas 费，从而使得攻击交易打包在合法交易之前，从而导致异常转账。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无

3.3 权限控制审计

智能合约中的权限控制审计分为两部分：权限漏洞审计以及权限过大审计。

3.3.1 权限漏洞审计【通过】

在有权限漏洞的智能合约中，攻击者可以对自身的账户进行提权，从而获得更高的执行权限，实施越权攻击。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无

3.3.2 权限过大审计【通过】

权限过大审计则着眼于审计合约中是否存在特殊用户权限过大，如导致无限增发代币等情况。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无

3.4 安全设计审计

智能合约中的安全设计审计分为五部分：安全模块使用、编译器版本安全、硬编码地址安全、敏感函数使用安全以及函数返回值安全。

3.4.1 安全模块使用【通过】

安全模块使用审计会审计目标代码是否使用如 OpenZepplin 提供的 SafeMath 库函数，以避免溢出漏洞的产生；如未使用此类库，则审计代码执行过程中是否严格校验转账金额大小，以避免溢出漏洞的产生。

检测结果：经检测，智能合约代码中使用了 SafeMath 库，避免了转账过程中出现溢出漏洞。

安全建议：无。

3.4.2 编译器版本安全审计【通过】

编译器版本安全着眼于审计代码是否明确地指出编译器版本，并审计是否使用的编译器版本过低，导致异常状况无法抛出。

检测结果：经检测，智能合约代码编译器版本> 0.4.0，当转入 Ether 时，不会抛出异常。

安全建议：无

3.4.3 硬编码地址安全审计【通过】

硬编码地址安全分析代码中硬编码的地址，可能是外部合约调用地址、个人

地址，审计外部合约是否存在异常从而影响本合约的执行。

检测结果：经检测，智能合约代码未采用硬编码。

安全建议：无

3.4.4 敏感函数使用安全审计【通过】

敏感函数使用主要分析合约代码中是否使用了 fallback、call、tx.origin 等不推荐的函数。

检测结果：经检测，智能合约代码未引用敏感函数。

安全建议：无

3.4.5 函数返回值安全审计【通过】

函数返回值设计主要分析函数是否正确地抛出了异常、是否正确地返回了交易的状态，避免“假充值”漏洞。

检测结果：经检测，智能合约代码能够正确抛出交易异常，返回交易状态，返回值安全。

安全建议：无

3.5 拒绝服务审计【通过】

拒绝服务供给可能导致智能合约永远无法恢复正常工作状态。攻击手段分为：交易接收方的恶意行为、人为增加计算功能、滥用合约 private 组件等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无

3.6 Gas 优化审计【通过】

如果智能合约中某个函数的计算量过大，如通过循环向变长数组进行批量转账，则极易造成交易的 gas 费过高，甚至达到区块的 gasLimit 导致交易执行失败。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

3.7 设计逻辑审计【通过】

除攻击漏洞外，代码在实现业务的过程中，逻辑存在问题，从而导致执行结果异常。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

附录一：合约源码审计详情

```
 1 pragma solidity ^0.4.24;   //Cheetah Mobile Security//明确编译版本，
符合安全推荐做法
 2
 3 /**
 4  * @dev Library that helps prevent integer overflows and underflows,
 5  * inspired by https://github.com/OpenZeppelin/zeppelin-solidity
 6  */
 7 library SafeMath {    //Cheetah Mobile Security//实现 zeppelin 的
safemath 库，避免溢出漏洞
 8     function add(uint256 a, uint256 b) internal pure returns (uint256)
{
 9         uint256 c = a + b;
10         require(c >= a);
11
12         return c;
13     }
14
15     function sub(uint256 a, uint256 b) internal pure returns (uint256)
{
16         require(b <= a);
17         uint256 c = a - b;
18
19         return c;
20     }
21
```

```
22    function mul(uint256 a, uint256 b) internal pure returns (uint256)
{
23        if (a == 0) {
24            return 0;
25        }
26        uint256 c = a * b;
27        require(c / a == b);
28
29        return c;
30    }
31
32    function div(uint256 a, uint256 b) internal pure returns (uint256)
{
33        require(b > 0);
34        uint256 c = a / b;
35
36        return c;
37    }
38 }
39
40 /**
41  * @title HasOwner
42  *
43  * @dev Allows for exclusive access to certain functionality.
44  */
45 contract HasOwner {
```

```
46    // Current owner.
47    address public owner;
48
49    // Conditionally the new owner.
50    address public newOwner;
51
52    /**
53     * @dev The constructor.
54     *
55     * @param _owner The address of the owner.
56     */
57    constructor(address _owner) internal {   //Cheetah Mobile
Security//正确调用构造函数，避免权限泄露
58        owner = _owner;
59    }
60
61    /**
62     * @dev Access control modifier that allows only the current owner
to call the function.
63     */
64    modifier onlyOwner {
65        require(msg.sender == owner);
66        _;
67    }
68
69    /**
```

```
70      * @dev The event is fired when the current owner is changed.
71      *
72      * @param _oldOwner The address of the previous owner.
73      * @param _newOwner The address of the new owner.
74      */
75     event OwnershipTransfer(address indexed _oldOwner, address
indexed _newOwner);
76
77     /**
78      * @dev Transfering the ownership is a two-step process, as we
prepare
79      * for the transfer by setting `newOwner` and requiring `newOwner`
to accept
80      * the transfer. This prevents accidental lock-out if something
goes wrong
81      * when passing the `newOwner` address.
82      *
83      * @param _newOwner The address of the proposed new owner.
84      */
85     function transferOwnership(address _newOwner) public onlyOwner {
86         newOwner = _newOwner;
87     }
88
89     /**
90      * @dev The `newOwner` finishes the ownership transfer process by
accepting the
```

```
91      * ownership.
92      */
93     function acceptOwnership() public {
94         require(msg.sender == newOwner);
95
96         emit OwnershipTransfer(owner, newOwner);
97
98         owner = newOwner;
99     }
100 }
101
102 /**
103  * @dev The standard ERC20 Token interface.
104  */
105 contract ERC20TokenInterface {
106     uint256 public totalSupply;  /* shorthand for public function and
a property */
107     event Transfer(address indexed _from, address indexed _to, uint256
_value);
108     event Approval(address indexed _owner, address indexed _spender,
uint256 _value);
109     function balanceOf(address _owner) public constant returns
(uint256 balance);
110     function transfer(address _to, uint256 _value) public returns
(bool success);
```

```
111     function transferFrom(address _from, address _to, uint256 _value)
public returns (bool success);
112     function approve(address _spender, uint256 _value) public returns
(bool success);
113     function allowance(address _owner, address _spender) public
constant returns (uint256 remaining);
114 }
115
116 /**
117  * @title ERC20Token
118  *
119  * @dev Implements the operations declared in the
`ERC20TokenInterface`.
120  */
121 contract ERC20Token is ERC20TokenInterface {
122     using SafeMath for uint256;
123
124     // Token account balances.
125     mapping (address => uint256) balances;
126
127     // Delegated number of tokens to transfer.
128     mapping (address => mapping (address => uint256)) allowed;
129
130     /**
131      * @dev Checks the balance of a certain address.
132      *
```

```
133      * @param _account The address which's balance will be checked.
134      *
135      * @return Returns the balance of the `_account` address.
136      */
137     function balanceOf(address _account) public constant returns
(uint256 balance) {
138         return balances[_account];
139     }
140
141     /**
142      * @dev Transfers tokens from one address to another.
143      *
144      * @param _to The target address to which the `_value` number of
tokens will be sent.
145      * @param _value The number of tokens to send.
146      *
147      * @return Whether the transfer was successful or not.
148      */
149     function transfer(address _to, uint256 _value) public returns
(bool success) {
150         require(_to != address(0));    //Cheetah Mobile Security//对
传入地址参数进行校验，避免误操作引起资金损失
151         require(_value <= balances[msg.sender]);  //Cheetah Mobile
Security//对传入参数和余额进行校验，避免溢出漏洞
152         require(_value > 0);  //Cheetah Mobile Security//对传入参数进
行校验，避免溢出漏洞
```

```
153
154         balances[msg.sender] = balances[msg.sender].sub(_value);
//Cheetah Mobile Security//转账操作先减后增，符合安全推荐做法
155         balances[_to] = balances[_to].add(_value);   //Cheetah
Mobile Security//使用 safemath 函数，避免溢出漏洞
156
157         emit Transfer(msg.sender, _to, _value);  //Cheetah Mobile
Security//对转账操作进行记录，符合安全推荐做法
158
159         return true;   //Cheetah Mobile Security//合理抛出转账的结果，
避免假充值漏洞
160     }
161
162     /**
163      * @dev Send `_value` tokens to `_to` from `_from` if `_from` has
approved the process.
164      *
165      * @param _from The address of the sender.
166      * @param _to The address of the recipient.
167      * @param _value The number of tokens to be transferred.
168      *
169      * @return Whether the transfer was successful or not.
170      */
171     function transferFrom(address _from, address _to, uint256 _value)
public returns (bool success) {
```

```
172         require(_value <= balances[_from]); //Cheetah Mobile
Security//对传入参数和余额进行校验，避免溢出漏洞
173         require(_value <= allowed[_from][msg.sender]);  //Cheetah
Mobile Security//对传入参数和余额进行校验，避免溢出漏洞
174         require(_value > 0);  //Cheetah Mobile Security//对传入参数进
行校验，避免溢出漏洞
175         require(_to != address(0));   //Cheetah Mobile Security//对
传入地址参数进行校验，避免误操作引起资金损失
176
177         balances[_from] = balances[_from].sub(_value);   //Cheetah
Mobile Security//转账操作先减后增，符合安全推荐做法
178         balances[_to] = balances[_to].add(_value);  //Cheetah Mobile
Security//使用 safemath 函数，避免溢出漏洞
179         allowed[_from][msg.sender] =
allowed[_from][msg.sender].sub(_value);  //Cheetah Mobile Security//使
用 safemath 函数，避免溢出漏洞
180
181         emit Transfer(_from, _to, _value);
182
183         return true;  //Cheetah Mobile Security//合理抛出转账的结果，避
免假充值漏洞
184     }
185
186     /**
187      * @dev Allows another contract to spend some tokens on your behalf.
188      *
```

```
189     * @param _spender The address of the account which will be approved
for transfer of tokens.
190     * @param _value The number of tokens to be approved for transfer.
191     *
192     * @return Whether the approval was successful or not.
193     */
194    function approve(address _spender, uint256 _value) public returns
(bool success) {
195        allowed[msg.sender][_spender] = _value;
196
197        emit Approval(msg.sender, _spender, _value);
198
199        return true;
200    }
201
202    /**
203     * @dev Increase the amount of tokens that an owner allowed to
a spender.
204     * approve should be called when allowed[_spender] == 0. To
increment
205     * allowed value is better to use this function to avoid 2 calls
(and wait until
206     * the first transaction is mined)
207     * From MonolithDAO Token.sol
208     *
209     * @param _spender The address which will spend the funds.
```

```
210      * @param _addedValue The amount of tokens to increase the
allowance by.
211      */
212     function increaseApproval(address _spender, uint256 _addedValue)
public returns (bool) {
213             allowed[msg.sender][_spender] =
(allowed[msg.sender][_spender].add(_addedValue));  //Cheetah Mobile
Security//使用 safemath 函数，避免溢出漏洞
214
215             emit Approval(msg.sender, _spender,
allowed[msg.sender][_spender]);
216
217             return true;
218     }
219
220     /**
221      * @dev Decrease the amount of tokens that an owner allowed to
a spender.
222      * approve should be called when allowed[_spender] == 0. To
decrement
223      * allowed value is better to use this function to avoid 2 calls
(and wait until
224      * the first transaction is mined)
225      * From MonolithDAO Token.sol
226      *
227      * @param _spender The address which will spend the funds.
```

```
228      * @param _subtractedValue The amount of tokens to decrease the
allowance by.
229      */
230     function decreaseApproval(address _spender, uint256
_subtractedValue) public returns (bool) {
231             uint256 oldValue = allowed[msg.sender][_spender];
232             if (_subtractedValue >= oldValue) {
233                     allowed[msg.sender][_spender] = 0;
234             } else {
235                     allowed[msg.sender][_spender] =
oldValue.sub(_subtractedValue);  //Cheetah Mobile Security//使用
safemath 函数，避免溢出漏洞
236             }
237
238             emit Approval(msg.sender, _spender,
allowed[msg.sender][_spender]);
239             return true;
240     }
241
242     /**
243      * @dev Shows the number of tokens approved by `_owner` that are
allowed to be transferred by `_spender`.
244      *
245      * @param _owner The account which allowed the transfer.
246      * @param _spender The account which will spend the tokens.
247      *
```

```
248      * @return The number of tokens to be transferred.
249      */
250     function allowance(address _owner, address _spender) public
constant returns (uint256 remaining) {
251         return allowed[_owner][_spender];
252     }
253
254     /**
255      * Don't accept ETH
256      */
257     function () public payable {
258         revert();
259     }
260 }
261
262 /**
263  * @title Freezable
264  * @dev This trait allows to freeze the transactions in a Token
265  */
266 contract Freezable is HasOwner {
267     bool public frozen = false;
268
269     /**
270      * @dev Modifier makes methods callable only when the contract is
not frozen.
271      */
```

```
272     modifier requireNotFrozen() {
273         require(!frozen);
274         _;
275     }
276 
277     /**
278      * @dev Allows the owner to "freeze" the contract.
279      */
280     function freeze() onlyOwner public {
281         frozen = true;
282     }
283 
284     /**
285      * @dev Allows the owner to "unfreeze" the contract.
286      */
287     function unfreeze() onlyOwner public {
288         frozen = false;
289     }
290 }
291 
292 /**
293  * @title FreezableERC20Token
294  *
295  * @dev Extends ERC20Token and adds ability to freeze all transfers
of tokens.
296  */
```

```
297 contract FreezableERC20Token is ERC20Token, Freezable {
298     /**
299      * @dev Overrides the original ERC20Token implementation by adding
whenNotFrozen modifier.
300      *
301      * @param _to The target address to which the `_value` number of
tokens will be sent.
302      * @param _value The number of tokens to send.
303      *
304      * @return Whether the transfer was successful or not.
305      */
306     function transfer(address _to, uint _value) public
requireNotFrozen returns (bool success) {
307         return super.transfer(_to, _value);
308     }
309
310     /**
311      * @dev Send `_value` tokens to `_to` from `_from` if `_from` has
approved the process.
312      *
313      * @param _from The address of the sender.
314      * @param _to The address of the recipient.
315      * @param _value The number of tokens to be transferred.
316      *
317      * @return Whether the transfer was successful or not.
318      */
```

```
319     function transferFrom(address _from, address _to, uint _value)
public requireNotFrozen returns (bool success) {
320         return super.transferFrom(_from, _to, _value);
321     }
322
323     /**
324      * @dev Allows another contract to spend some tokens on your behalf.
325      *
326      * @param _spender The address of the account which will be approved
for transfer of tokens.
327      * @param _value The number of tokens to be approved for transfer.
328      *
329      * @return Whether the approval was successful or not.
330      */
331     function approve(address _spender, uint _value) public
requireNotFrozen returns (bool success) {
332         return super.approve(_spender, _value);
333     }
334
335     function increaseApproval(address _spender, uint256 _addedValue)
public requireNotFrozen returns (bool) {
336         return super.increaseApproval(_spender, _addedValue);
337     }
338
339     function decreaseApproval(address _spender, uint256
_subtractedValue) public requireNotFrozen returns (bool) {
```

```
340         return super.decreaseApproval(_spender, _subtractedValue);
341     }
342 }
343
344 /**
345  * @title BonusCloudTokenConfig
346  *
347  * @dev The static configuration for the Bonus Cloud Token.
348  */
349 contract BonusCloudTokenConfig {
350     // The name of the token.
351     string constant NAME = "Bonus Cloud Token";
352
353     // The symbol of the token.
354     string constant SYMBOL = "BxC";
355
356     // The number of decimals for the token.
357     uint8 constant DECIMALS = 18;
358
359     // Decimal factor for multiplication purposes.
360     uint256 constant DECIMALS_FACTOR = 10 ** uint(DECIMALS);
361
362     // TotalSupply
363     uint256 constant TOTAL_SUPPLY = 7000000000 * DECIMALS_FACTOR;
364 }
365
```

```
366 /**
367  * @title Bonus Cloud Token
368  *
369  * @dev A standard token implementation of the ERC20 token standard
with added
370  *      HasOwner trait and initialized using the configuration
constants.
371  */
372 contract BonusCloudToken is BonusCloudTokenConfig, HasOwner,
FreezableERC20Token {
373     // The name of the token.
374     string public name;
375
376     // The symbol for the token.
377     string public symbol;
378
379     // The decimals of the token.
380     uint8 public decimals;
381
382     /**
383      * @dev The constructor.
384      *
385      */
386     constructor() public HasOwner(msg.sender) {  //Cheetah Mobile
Security//正确调用构造函数，避免权限泄露
387         name = NAME;
```

```
388         symbol = SYMBOL;
389         decimals = DECIMALS;
390         totalSupply = TOTAL_SUPPLY;
391         balances[owner] = TOTAL_SUPPLY;
392     }
393 }
```

猎豹区块链安全

audit@cmcm.com