Smart Contract Audit Report

Bonus Cloud Token (BxC)

Audit Result: Passed

Version Description

Reviser	Revisions	Revision Date	Version	Reviewer
Wang Hailong	Create document	13-9-2018	V1.0	Yang Wenyu

Document Information

Name	Version	Document ID	Classification Level
Bonus Cloud Token Smart Contract Audit Report	V1.0	0x53a94403	classified

Copyright Notice

Table of Contents

I、Review

1.1 Audit Background

Cheetah Mobile Security team conducted smart contract audit for **Bonus Cloud Token (BxC)** on **Sep. 13, 2018**. This report presents the audit details and results.

(***Disclaimer***: *Cheetah Mobile Security only issues the report based on the vulnerabilities existed before the issuance of the report, and bears corresponding responsibility. As for the facts that occur or exist after the issuance of the report, whose impacts on the project cannot be determined, Cheetah Mobile Security is not responsible for the consequence. The security audit analysis and other contents of this report are based only on the documents and information provided by the information provider to Cheetah Mobile Security as of the date of issuance of the report. The information provider is obliged to ensure that there is not missing, falsified, deletion or concealment of the information provided. If present, Cheetah Mobile Security shall not be liable for any loss or adverse effect caused by this situation.*)

1.2 Audit Result

Audit Result	Auditor	Reviewer
Passed	Wang Hailong	Yang Wenyu

1.3 Basic Information

- **Contract name：**

  BonusCloudToken

- **Contract address：**

  -

- **Etherscan URL：**

  -

- **Contract Source Code URL：**

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

- **Contract type：**

  Token contract, token name: Bonus Cloud Token（BxC）

- **On-chain Date：**

  -

II、 Contract Code Vulnerability Analysis

2.1 Contract Code Vulnerability Levels

The number of contract vulnerabilities is counted by level as follows：

High Risk	Medium Risk	Low Risk

0	0	0

2.2 Contract Code Vulnerability Distribution

Not found

2.3 Contract Code Audit Items and Results

(Other unknown vulnerabilities are not included in the scope of responsibility of this audit)

Audit Method	Audit Class	Audit Subclass	Audit Result
Offensive/ Defensive Audit	Arithmetic Safety	Integer Overflow	Passed
		Integer Underflow	Passed
		Operation Precision	Passed
	Competitive Competition	Reentrancy	Passed
		Transaction Ordering Dependence	Passed
	Access Control	Privilege Vulnerability	Passed
		Overprivileged Audit	Passed
	Security Design	Security Module Usage	Passed
		Compiler Version Security	Passed

		Hard Coded Address Security	Passed
		Sensitive Functions (fallback/call/tx.origin) Security	Passed
		Function Return Value	Passed
	Denial of Service	-	Passed
	Gas Optimazation	-	Passed
	Design Logic	-	Passed

III、 Audit Details

3.1 Arithmetic Safety Audit

The arithmetic security audit is divided into three parts: integer overflow audit, integer underflow audit and operation precision audit.

3.1.1 Integer Overflow Audit【Passed】

Solidity can handle 256 bits of data at most. When the maximum number increases, it will overflow. If the integer overflow occurs in the transfer logic, it will make the amount of transfer funds miscalculated, resulting in serious capital risk.

Audit Result：The code meets the specification.

Security Recommendation：No

3.1.2 Integer Underflow Audit【Passed】

Solidity can handle 256 bits of data at most. When the minimum number decreases, it will underflow. If the integer underflow occurs in the transfer logic, it will make the amount of transfer funds miscalculated and lead to serious capital risk.

Audit Result：The code meets the specification.

Security Recommendation：No

3.1.3 Operation Precision Audit【Passed】

Solidity performs type coercion in the process of multiplication and division. If the precision risk is included in the operation of capital variable, it will lead to user transfer logic error and capital loss.

Audit Result：The code meets the specification.

Security Recommendation：No

3.2 Competitive Competition Audit

The competitive competition audit is divided into two parts: reentrancy audit and transaction ordering dependence audit. With competitive vulnerabilities, an attacker can modify the output of a program by adjusting the execution process of transactions with a certain probability.

3.2.1 Reentrancy Audit【Passed】

Reentrancy occurs when external contract calls are allowed to make new calls to the calling contract before the initial execution is complete. For a function, this means that the contract state may change in the middle of its execution as a result of a call to an untrusted contract or the use of a low level function with an external address.

Audit Result：The code meets the specification.

Security Recommendation：No

3.2.2 Transaction Ordering Dependence Audit【Passed】

Since miners always get rewarded via gas fees for running code on behalf of externally owned addresses (EOA), users can specify higher fees to have their transactions mined more quickly. Since the Ethereum blockchain is public, everyone can see the contents of others' pending transactions. This means if a given user is revealing the solution to a puzzle or other valuable secret, a malicious user can steal the solution and copy their transaction with higher fees to preempt the original solution. If developers of smart contracts are not careful, this situation can lead to practical and devastating front-running attacks.

Audit Result：The code meets the specification.

Security Recommendation：No

3.3 Access Control Audit

Access control audit is divided into two parts: privilege vulnerability audit and overprivileged audit.

3.3.1 Privilege Vulnerability Audit【Passed】

Smart contracts with privilege vulnerability, attackers can weigh their own accounts

to gain higher execution privileges.

Audit Result：The code meets the specification.

Security Recommendation：No

3.3.2 Overprivileged Audit【Passed】

Overprivileged auditing focuses on whether there are special user privileges in audit contracts, such as allowing a user to unlimitly mine tokens.

Audit Result：The code meets the specification.

Security Recommendation：No

3.4 Security Design Audit

Security design audit is divided into five parts: security module usage, compiler version security, hard-coded address security, sensitive function usage security and function return value security.

3.4.1 Security Module Usage【Passed】

The security module usage audit whether the smart contract uses the SafeMath library function provided by OpenZepplin to avoid overflow vulnerabilities; if it does not, whether the transfer amount is strictly checked during the execution.

Audit Result：The code meets the specification.

Security Recommendation：No

3.4.2 Compiler Version Security Audit【Passed】

Compiler version security focuses on whether the smart contract explicitly indicates the compiler version and whether the compiler version used is too low to throw an exception.

Audit Result：The code meets the specification.

Security Recommendation：No

3.4.3 Hard Coded Address Security Audit【Passed】

Hard-coded address security audit static addressed in the smart contract to check whether there is an exception to the external contract, thus affecting the execution of this contract.

Audit Result：The code meets the specification.

Security Recommendation：No

3.4.4 Sensitive Functions Audit【Passed】

Sensitive functions audit checks whether the smart contract uses the unrecommandation functions such as fallback, call and tx.origin.

Audit Result：The code meets the specification.

Security Recommendation：No

3.4.5 Function Return Value Audit【Passed】

Function return value audit mainly analyzes whether the function correctly throws an exception, correctly returns to the state of the transaction.

Audit Result：The code meets the specification.

Security Recommendation：No

3.5 Denial of Service Audit【Passed】

Denial of service attack sometimes can make the smart contract offline forever by maliciously behaving when being the recipient of a transaction, artificially increasing the gas necessary to compute a function, abusing access controls to access private components of smart contracts, taking advantage of mixups and negligence and so on.

Audit Result：The code meets the specification.

Security Recommendation：No

3.6 Gas Optimization Audit【Passed】

If the computation of a function in a smart contract is too complex, such as the batch transfer to a variable-length array through a loop, it is very easy to cause the gas fee beyond the block's gas Limit resulting in transaction execution failure.

Audit Result：The code meets the specification.

Security Recommendation：NO

3.7 Design Logic Audit【Passed】

In addition to vulnerabilities, there are logic problems in the process of code implementation, resulting in abnormal execution results.

Audit Result：The code meets the specification.

Security Recommendation：No

Appendix I：Contract Code Audit Details

```
 1 pragma solidity ^0.4.24;   //Cheetah Mobile Security//Specifying the
compiled version conforming to the security coding specification
 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//realize SafeMath
library of Zeppelin, to avoid overflow attack
 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// Use contract constructors correctly to avoid permission
leakage
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//Check the transfer address to avoid the loss of capital caused
by misoperation.
```

```
151         require(_value <= balances[msg.sender]);  //Cheetah Mobile
Security//Check incoming parameters to avoid overflow vulnerabilities.
152         require(_value > 0);  //Cheetah Mobile Security//Check
incoming parameters to avoid overflow vulnerabilities.
153
154         balances[msg.sender] = balances[msg.sender].sub(_value);
//Cheetah Mobile Security//Transfers first increase and then decrease,
conforming to safety coding standard.
155         balances[_to] = balances[_to].add(_value);   //Cheetah
Mobile Security/using safemath function to avoid overflow
vulnerabilities.
156
157         emit Transfer(msg.sender, _to, _value);
158
159         return true;   //Cheetah Mobile Security//catch any exception
during transfer, to avoid "fake transfer" vulnerabilities.
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//Check incoming parameters to avoid overflow vulnerabilities.
173        require(_value <= allowed[_from][msg.sender]);  //Cheetah
Mobile Security//Check incoming parameters to avoid overflow
vulnerabilities.
174        require(_value > 0);  //Cheetah Mobile Security//Check
incoming parameters to avoid overflow vulnerabilities.
175        require(_to != address(0));   //Cheetah Mobile
Security//Check the transfer address to avoid the loss of capital caused
by misoperation.
176
177        balances[_from] = balances[_from].sub(_value);   //Cheetah
Mobile Security//Transfers first increase and then decrease, conforming
to safety coding standard.
178        balances[_to] = balances[_to].add(_value); //Cheetah Mobile
Security/using safemath function to avoid overflow vulnerabilities.
179        allowed[_from][msg.sender] =
allowed[_from][msg.sender].sub(_value); //Cheetah Mobile
Security/using safemath function to avoid overflow vulnerabilities.
180
181        emit Transfer(_from, _to, _value);
182
```

```
183         return true;  //Cheetah Mobile Security//catch any exception
during transfer, to avoid "fake transfer" vulnerabilities
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/using safemath function to avoid overflow vulnerabilities.
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/using
safemath function to avoid overflow vulnerabilities.
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// Use contract constructors correctly to avoid permission
leakage
387         name = NAME;
388         symbol = SYMBOL;
389         decimals = DECIMALS;
390         totalSupply = TOTAL_SUPPLY;
391         balances[owner] = TOTAL_SUPPLY;
392     }
393 }
```

Cheetah Mobile Security

audit@cmcm.com