FX

Smart contract security audit report

Audit Number：202205150060

2022-05-09

Smart Contract Name：Martis Token(Martis)

Smart Contract Address：0x501100d9a56a8e140cde43c8aef2abeb278a8f2f

Smart Contract Address Link：

https://bscscan.com/address/0x501100d9a56a8e140cde43c8aef2abeb278a8f2f#code

Start Date：20220507

Completion Date：20220509

Overall Result：Pass

Audit Team: FX Audit(Singapore) Technology Co. Ltd

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	BEP20 Token Standards	Pass
		Compiler Version Security	Pass
		Visibility Specifiers	Pass
		Gas Consumption	Pass
		SafeMath Features	Pass
		Fallback Usage	Pass
		tx.origin Usage	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		Overriding Variables	Pass
2	Function Call Audit	Authorization of Function Call	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		selfdestruct Function Security	Pass
3	Business Security	Access Control of Owner	Pass
		Business Logics	Pass
		Business Implementations	Pass
4	Integer Overflow/Underflow	-	Pass
5	Reentrancy	-	Pass
6	Exceptional Reachable State	-	Pass
7	Transaction-Ordering Dependence	-	Pass
8	Block Properties Dependence	-	Pass
9	Pseudo-random Number Generator (PRNG)	-	Pass
10	DoS (Denial of Service)	-	Pass
11	Token Vesting Implementation	-	N/A
12	Fake Deposit	-	Pass
13	event security	-	Pass

Note: Audit results and suggestions in code comments.

Disclaimer:

This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. FX Audit only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, FX Audit lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to FX Audit before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, FX Audit assumes no responsibility for the resulting loss or adverse effects. The audit report issued by FX Audit is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by FX Audit. Due to the technical limitations of any organization, this report conducted by FX Audit still has the possibility that the entire risk cannot be completely detected. FX disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to FX.

udit Results Explained:

FX audit has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract Martis,including Coding Standards,Security, and Business Logic. Martis contract passed all audit items. The overall result is Pass.The smart contract is able to function properly. Please find below the basic information of the smart contract.

1、Basic Token Information

Token name	Martis Life
Token symbol	Martis
decimals	6
totalSupply	280,000,000
Token type	BEP20

2、Token Vesting Information

N/A

Audited Source Code with Comments:

```
/**
 *Submitted for verification at BscScan.com on 2022-03-17
*/

// SPDX-License-Identifier: MIT

pragma solidity >=0.8.6;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
//FX Declare IERC20 contract standard token interface
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
```

```
26.      *
27.      * Returns a boolean value indicating whether the operation succe
    eded.
28.      *
29.      * Emits a {Transfer} event.
30.      */
31.     function transfer(address recipient, uint256 amount)
32.         external
33.         returns (bool);
34.
35.     /**
36.      * @dev Returns the remaining number of tokens that `spender` wil
    l be
37.      * allowed to spend on behalf of `owner` through {transferFrom}.
    This is
38.      * zero by default.
39.      *
40.      * This value changes when {approve} or {transferFrom} are called
    .
41.      */
42.     function allowance(address owner, address spender)
43.         external
44.         view
45.         returns (uint256);
46.
47.     /**
48.      * @dev Sets `amount` as the allowance of `spender` over the call
    er's tokens.
49.      *
50.      * Returns a boolean value indicating whether the operation succe
    eded.
51.      *
52.      * IMPORTANT: Beware that changing an allowance with this method
    brings the risk
53.      * that someone may use both the old and the new allowance by unf
    ortunate
54.      * transaction ordering. One possible solution to mitigate this r
    ace
55.      * condition is to first reduce the spender's allowance to 0 and
    set the
56.      * desired value afterwards:
57.      * https://github.com/ethereum/EIPs/issues/20#issuecomment-
    263524729
58.      *
```

```
59.      * Emits an {Approval} event.
60.      */
61.     function approve(address spender, uint256 amount) external return
    s (bool);
62.
63.     /**
64.      * @dev Moves `amount` tokens from `sender` to `recipient` using
    the
65.      * allowance mechanism. `amount` is then deducted from the caller
    's
66.      * allowance.
67.      *
68.      * Returns a boolean value indicating whether the operation succe
    eded.
69.      *
70.      * Emits a {Transfer} event.
71.      */
72.     function transferFrom(
73.         address sender,
74.         address recipient,
75.         uint256 amount
76.     ) external returns (bool);
77.
78.     /**
79.      * @dev Emitted when `value` tokens are moved from one account (`
    from`) to
80.      * another (`to`).
81.      *
82.      * Note that `value` may be zero.
83.      */
84.     event Transfer(address indexed from, address indexed to, uint256
    value);
85.
86.     /**
87.      * @dev Emitted when the allowance of a `spender` for an `owner`
    is set by
88.      * a call to {approve}. `value` is the new allowance.
89.      */
90.     event Approval(
91.         address indexed owner,
92.         address indexed spender,
93.         uint256 value
94.     );
95. }
```

```
96. //FX Declare the mathematical operation library to prevent overflow of mathematical operation
97. library SafeMath {
98.     /**
99.      * @dev Returns the addition of two unsigned integers, reverting on
100.       * overflow.
101.       *
102.       * Counterpart to Solidity's `+` operator.
103.       *
104.       * Requirements:
105.       *
106.       * - Addition cannot overflow.
107.       */
108.      function add(uint256 a, uint256 b) internal pure returns (uint256) {
109.          uint256 c = a + b;
110.          require(c >= a, "SafeMath: addition overflow");
111.
112.          return c;
113.      }
114.
115.      /**
116.       * @dev Returns the subtraction of two unsigned integers, reverting on
117.       * overflow (when the result is negative).
118.       *
119.       * Counterpart to Solidity's `-` operator.
120.       *
121.       * Requirements:
122.       *
123.       * - Subtraction cannot overflow.
124.       */
125.      function sub(uint256 a, uint256 b) internal pure returns (uint256) {
126.          return sub(a, b, "SafeMath: subtraction overflow");
127.      }
128.
129.      /**
130.       * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
131.       * overflow (when the result is negative).
132.       *
133.       * Counterpart to Solidity's `-` operator.
```

```
134.         *
135.         * Requirements:
136.         *
137.         * - Subtraction cannot overflow.
138.         */
139.       function sub(
140.           uint256 a,
141.           uint256 b,
142.           string memory errorMessage
143.       ) internal pure returns (uint256) {
144.           require(b <= a, errorMessage);
145.           uint256 c = a - b;
146.
147.           return c;
148.       }
149.
150.       /**
151.        * @dev Returns the multiplication of two unsigned integers, r
    everting on
152.        * overflow.
153.        *
154.        * Counterpart to Solidity's `*` operator.
155.        *
156.        * Requirements:
157.        *
158.        * - Multiplication cannot overflow.
159.        */
160.       function mul(uint256 a, uint256 b) internal pure returns (uint
    256) {
161.           // Gas optimization: this is cheaper than requiring 'a' no
    t being zero, but the
162.           // benefit is lost if 'b' is also tested.
163.           // See: https://github.com/OpenZeppelin/openzeppelin-
    contracts/pull/522
164.           if (a == 0) {
165.               return 0;
166.           }
167.
168.           uint256 c = a * b;
169.           require(c / a == b, "SafeMath: multiplication overflow");
170.
171.           return c;
172.       }
173.
```

```
174.      /**
175.       * @dev Returns the integer division of two unsigned integers.
    Reverts on
176.       * division by zero. The result is rounded towards zero.
177.       *
178.       * Counterpart to Solidity's `/` operator. Note: this function
    uses a
179.       * `revert` opcode (which leaves remaining gas untouched) whil
    e Solidity
180.       * uses an invalid opcode to revert (consuming all remaining g
    as).
181.       *
182.       * Requirements:
183.       *
184.       * - The divisor cannot be zero.
185.       */
186.      function div(uint256 a, uint256 b) internal pure returns (uint
    256) {
187.          return div(a, b, "SafeMath: division by zero");
188.      }
189.
190.      /**
191.       * @dev Returns the integer division of two unsigned integers.
    Reverts with custom message on
192.       * division by zero. The result is rounded towards zero.
193.       *
194.       * Counterpart to Solidity's `/` operator. Note: this function
    uses a
195.       * `revert` opcode (which leaves remaining gas untouched) whil
    e Solidity
196.       * uses an invalid opcode to revert (consuming all remaining g
    as).
197.       *
198.       * Requirements:
199.       *
200.       * - The divisor cannot be zero.
201.       */
202.      function div(
203.          uint256 a,
204.          uint256 b,
205.          string memory errorMessage
206.      ) internal pure returns (uint256) {
207.          require(b > 0, errorMessage);
208.          uint256 c = a / b;
```

```
209.           // assert(a == b * c + a % b); // There is no case in whic
    h this doesn't hold
210.
211.           return c;
212.       }
213.
214.       /**
215.        * @dev Returns the remainder of dividing two unsigned integer
    s. (unsigned integer modulo),
216.        * Reverts when dividing by zero.
217.        *
218.        * Counterpart to Solidity's `%` operator. This function uses
    a `revert`
219.        * opcode (which leaves remaining gas untouched) while Solidit
    y uses an
220.        * invalid opcode to revert (consuming all remaining gas).
221.        *
222.        * Requirements:
223.        *
224.        * - The divisor cannot be zero.
225.        */
226.       function mod(uint256 a, uint256 b) internal pure returns (uint
    256) {
227.           return mod(a, b, "SafeMath: modulo by zero");
228.       }
229.
230.       /**
231.        * @dev Returns the remainder of dividing two unsigned integer
    s. (unsigned integer modulo),
232.        * Reverts with custom message when dividing by zero.
233.        *
234.        * Counterpart to Solidity's `%` operator. This function uses
    a `revert`
235.        * opcode (which leaves remaining gas untouched) while Solidit
    y uses an
236.        * invalid opcode to revert (consuming all remaining gas).
237.        *
238.        * Requirements:
239.        *
240.        * - The divisor cannot be zero.
241.        */
242.       function mod(
243.           uint256 a,
244.           uint256 b,
```

```
245.             string memory errorMessage
246.         ) internal pure returns (uint256) {
247.             require(b != 0, errorMessage);
248.             return a % b;
249.         }
250.     }
251.
252.     /**
253.      * @dev Contract module which provides a basic access control mech
    anism, where
254.      * there is an account (an owner) that can be granted exclusive ac
    cess to
255.      * specific functions.
256.      *
257.      * By default, the owner account will be the one that deploys the
    contract. This
258.      * can later be changed with {transferOwnership}.
259.      *
260.      * This module is used through inheritance. It will make available
     the modifier
261.      * `onlyOwner`, which can be applied to your functions to restrict
     their use to
262.      * the owner.
263.      */
264.     // FX Declare a context contract for context control
265.     contract Ownable {
266.         address internal _owner;
267.
268.         /**
269.          * @dev Returns the address of the current owner.
270.          */
271.         function owner() public view returns (address) {
272.             return _owner;
273.         }
274.
275.         /**
276.          * @dev Throws if called by any account other than the owner.
277.          */
278.         modifier onlyOwner() {
279.             require(_owner == msg.sender, "Ownable: caller is not the
    owner");
280.             _;
281.         }
282.
```

```
283.       function changeOwner(address newOwner) public onlyOwner {
284.           _owner = newOwner;
285.       }
286.   }
287.   // FX Declare BEP20 token content
288.   contract Martis is IERC20, Ownable {
289.       using SafeMath for uint256;
290.
291.       mapping(address => uint256) private _balances;
292.
293.       mapping(address => mapping(address => uint256)) private _allow
    ances;
294.
295.       uint256 private _totalSupply;
296.
297.       string private _name;
298.       string private _symbol;
299.       uint8 private _decimals;
300.   // Burn address
301.       address private _deadAddress =
302.           address(0x000000000000000000000000000000000000dEaD);
303.   // pancakeswap address
304.       address public uniswapV2Pair;
305.   // isExcluded list
306.       mapping(address => bool) public isExcluded;
307.   // Invite relationship
308.       mapping(address => address) public inviter;
309.
310.       constructor(address tokenOwner) {
311.           _name = "Martis Life";
312.           _symbol = "Martis";
313.           _decimals = 6;
314.           _owner = tokenOwner;
315.
316.           isExcluded[tokenOwner] = true;
317.           isExcluded[_deadAddress] = true;
318.
319.           _totalSupply = _totalSupply.add( 28 * 10**7 * 10**_decimal
    s);
320.           _balances[tokenOwner] = _balances[tokenOwner].add(28 * 10*
    *7 * 10**_decimals);
321.       }
322.
323.       /**
```

```
324.         * @dev Returns the name of the token.
325.         */
326.       function name() public view returns (string memory) {
327.           return _name;
328.       }
329.
330.       function symbol() public view returns (string memory) {
331.           return _symbol;
332.       }
333.
334.       function decimals() public view returns (uint8) {
335.           return _decimals;
336.       }
337.
338.       /**
339.        * @dev See {IERC20-totalSupply}.
340.        */
341.       function totalSupply() public view override returns (uint256)
    {
342.           return _totalSupply;
343.       }
344.
345.       /**
346.        * @dev See {IERC20-balanceOf}.
347.        */
348.       function balanceOf(address account) public view override retur
    ns (uint256) {
349.           return _balances[account];
350.       }
351.
352.       function transfer(address recipient, uint256 amount)
353.           public
354.           virtual
355.           override
356.           returns (bool)
357.       {
358.           _transfer(msg.sender, recipient, amount);
359.           return true;
360.       }
361.
362.       /**
363.        * @dev See {IERC20-allowance}.
364.        */
365.       function allowance(address owner, address spender)
```

```
366.           public
367.           view
368.           virtual
369.           override
370.           returns (uint256)
371.       {
372.           return _allowances[owner][spender];
373.       }
374.
375.       function approve(address spender, uint256 amount)
376.           public
377.           virtual
378.           override
379.           returns (bool)
380.       {
381.           _approve(msg.sender, spender, amount);
382.           return true;
383.       }
384.
385.       function transferFrom(
386.           address sender,
387.           address recipient,
388.           uint256 amount
389.       ) public virtual override returns (bool) {
390.           _transfer(sender, recipient, amount);
391.           _approve(
392.               sender,
393.               msg.sender,
394.               _allowances[sender][msg.sender].sub(
395.                   amount,
396.                   "ERC20: transfer amount exceeds allowance"
397.               )
398.           );
399.           return true;
400.       }
401.
402.       function increaseAllowance(address spender, uint256 addedValue
    )
403.           public
404.           virtual
405.           returns (bool)
406.       {
407.           _approve(
408.               msg.sender,
```

```
409.                 spender,
410.                 _allowances[msg.sender][spender].add(addedValue)
411.             );
412.             return true;
413.         }
414.
415.         function decreaseAllowance(address spender, uint256 subtracted
    Value)
416.             public
417.             virtual
418.             returns (bool)
419.         {
420.             _approve(
421.                 msg.sender,
422.                 spender,
423.                 _allowances[msg.sender][spender].sub(
424.                     subtractedValue,
425.                     "ERC20: decreased allowance below zero"
426.                 )
427.             );
428.             return true;
429.         }
430.
431.         function setIsExcluded(address from, bool v) public onlyOwner
    {
432.             isExcluded[from] = v;
433.         }
434.
435.         function changeSwapAddress(address _addr) public onlyOwner {
436.             uniswapV2Pair = _addr;
437.         }
438.     // fee calculate
439.         function _takeInviterFee(
440.             address sender,
441.             address recipient,
442.             uint256 amount
443.         ) private returns (uint256) {
444.             address cur;
445.             if (sender == uniswapV2Pair) {
446.                 cur = recipient;
447.             } else {
448.                 cur = sender;
449.             }
450.             uint256 accurRate;
```

```
451.           uint256 rates = uint256(30);
452.           for (uint256 i = 0; i < 1; i++) {
453.               uint256 rate = rates;
454.               cur = inviter[cur];
455.               if (cur == address(0)) {
456.                   break;
457.               }
458.               if (balanceOf(cur) >= 1 * 10**_decimals) {
459.                   accurRate = accurRate.add(rate);
460.                   uint256 curAmount = amount.div(1000).mul(rate);
461.                   _balances[cur] = _balances[cur].add(curAmount);
462.                   emit Transfer(sender, cur, curAmount);
463.               }
464.           }
465.
466.           return accurRate;
467.       }
468.
469.       function _transfer(
470.           address sender,
471.           address recipient,
472.           uint256 amount
473.       ) internal virtual {
474.           require(sender != address(0), "ERC20: transfer from the zero address");
475.           require(recipient != address(0), "ERC20: transfer to the zero address");
476.
477.           _beforeTokenTransfer(sender, recipient, amount);
478.
479.           _balances[sender] = _balances[sender].sub(
480.               amount,
481.               "ERC20: transfer amount exceeds balance"
482.           );
483.
484.           bool shouldSetInviter = balanceOf(recipient) == 0 &&
485.               inviter[recipient] == address(0) &&
486.               sender != uniswapV2Pair;
487.   //free rate
488.           uint256 rate = 0;
489.           if (!isExcluded[sender] && !isExcluded[recipient]) {
490.               uint256 accurRate = _takeInviterFee(sender, recipient, amount);
491.
```

```
492.                 uint256 deadAmount = amount.div(100) +
493.                     amount.mul(50 - accurRate).div(1000);
494.                 _balances[_deadAddress] = _balances[_deadAddress].add(
    deadAmount);
495.                 emit Transfer(sender, _deadAddress, deadAmount);
496.
497.                 rate = 6;
498.             }
499.
500.             uint256 receiveAmount = amount.mul(100 - rate).div(100);
501.             _balances[recipient] = _balances[recipient].add(receiveAmo
    unt);
502.             emit Transfer(sender, recipient, receiveAmount);
503.
504.             if (shouldSetInviter) {
505.                 inviter[recipient] = sender;
506.             }
507.         }
508.
509.         function _burn(address account, uint256 amount) internal virtu
    al {
510.             require(account != address(0), "ERC20: burn from the zero
    address");
511.
512.             _beforeTokenTransfer(account, address(0), amount);
513.
514.             _balances[account] = _balances[account].sub(
515.                 amount,
516.                 "ERC20: burn amount exceeds balance"
517.             );
518.             _totalSupply = _totalSupply.sub(amount);
519.             emit Transfer(account, address(0), amount);
520.         }
521.
522.         function _approve(
523.             address owner,
524.             address spender,
525.             uint256 amount
526.         ) internal virtual {
527.             require(owner != address(0), "ERC20: approve from the zero
     address");
528.             require(spender != address(0), "ERC20: approve to the zero
     address");
529.
```

```
530.             _allowances[owner][spender] = amount;
531.             emit Approval(owner, spender, amount);
532.         }
533.
534.         function _beforeTokenTransfer(
535.             address from,
536.             address to,
537.             uint256 amount
538.         ) internal virtual {}
539.     }
```

Appendix:safety risk rating criteria:

Vulnerability rating	Vulnerability rating description
High risk	Loopholes that can directly cause the loss of token contracts or users' funds, such as value overflow loopholes that can cause the value of tokens to return to zero, false recharge loopholes that can cause the loss of tokens in the exchange, and reentry loopholes that can cause the loss of assets or tokens in the contract account; Vulnerabilities that can cause the loss of ownership of token contracts, such as access control defects of key functions, bypassing access control of key functions caused by call injection, etc; A vulnerability that can cause token contracts to fail to workproperly.
Medium risk	High risk vulnerabilities that require a specific address to trigger, such as value overflow vulnerabilities that can be triggered by token contract owners; Access control defects of non key functions, logic design defects that can not cause direct capital loss, etc.
Low risk	Vulnerabilities that are difficult to trigger, vulnerabilities that do limited harm after triggering, such as value overflow vulnerabilities that require a large number of tokens to trigger, vulnerabilities that the attacker cannot make direct profits after triggering value overflow, transaction sequence dependency risk triggered by specifying high mining fee, etc.

FX Contract safe

Email: support@vatin.xyz