Smart contract security audit report

Audit Number：201903151136

Smart Contract Name：

ETERNAL TOKEN (XET)

Smart Contract Address：

0x054C64741dBafDC19784505494029823D89c3b13

Smart Contract Address Link：

https://etherscan.io/address/0x054c64741dbafdc19784505494029823d89c3b13#code

Start Date：2019.03.14

Completion Date：2019.03.15

Overall Result：Pass（Merit）

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	ERC20 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
		Self-destruct 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	-	**Missing**
12	Fake Deposit	-	Pass
13	Event security	-	Pass

Note: Audit results and suggestions in code comments

Disclaimer：This audit is only for the subitems and the range of audit categories given in the audit result table. Other unknown security vulnerabilities not listed in the table are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology issues this report only based on the vulnerabilities that have already occurred or existed and takes corresponding responsibility in this regard. As for the new attacks or vulnerabilities that occur or exist in the future, Beosin (Chengdu LianAn) Technology cannot predict the security status of its smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are only based on the documents and materials provided by the contract provider to Beosin (Chengdu LianAn) Technology as of the issued time of this report, and no missing, falsified, deleted, or concealed documents and materials will be accepted. Contract provider should be aware that if the documents and materials provided are missing, falsified, deleted, concealed or inconsistent with the actual situation, Beosin (Chengdu LianAn) Technology disclaims any liability for the losses and negative effects caused by this reason.

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract XET, including Coding Standards, Security, and Business Logic. **XET contract passed all audit items. The overall result is Pass (Merit). The smart contract is able to function properly.** Please find below the basic information of the smart contract:

1、Basic Token Information

Token name	ETERNAL TOKEN
Token symbol	XET
Decimals	8
Total supply	200 million (Total supply is constant)
Token type	ERC20

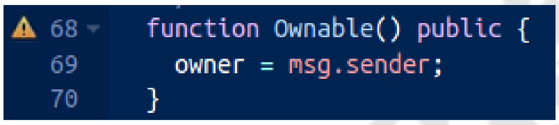
Table 1 – Basic Token Information

2、Token Vesting Information

Missing

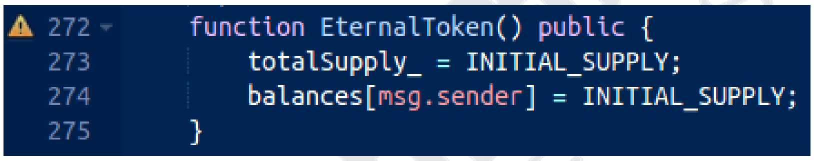
3、Other Audit Suggestion

The contract specifies that the minimum compiler version is 0.4.23, but when compiled with it, there are two warnings as shown in Figure 1 and 2.



```
function Ownable() public {
  owner = msg.sender;
}
```

Figure 1 – The constructor source code of contract 'Ownable'



```
function EternalToken() public {
    totalSupply_ = INITIAL_SUPPLY;
    balances[msg.sender] = INITIAL_SUPPLY;
}
```

Figure 2 – The constructor source code of contract 'EternalToken'

Safety Suggestion: It is recommended to use style like 'constructor(...) public {...}' to replace the constructor.

Audited Source Code with Comments:

```
pragma solidity ^0.4.23; // Beosin (Chengdu LianAn) // It is recommended to fix the compiler version and eliminate compiler warnings.

// ----------------------------------------------------------------------------
// 'ETERNAL TOKEN' token contract
//
// Symbol      : XET
// Name        : ETERNAL TOKEN
// Total supply: 200000000
// Decimals    : 8
//
//
// (c) by atom-solutions 2018. The MIT Licence.
// ----------------------------------------------------------------------------

// ----------------------------------------------------------------------------
// Safe maths
// ----------------------------------------------------------------------------

/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }

  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}

/**
 * @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; // Beosin (Chengdu LianAn) // Declares the owner.
  // Beosin (Chengdu LianAn) // Declares the event 'OwnershipTransferred'.
  event OwnershipTransferred(address indexed previousOwner, address indexed
newOwner);


  /**
   * @dev The Ownable constructor sets the original `owner` of the contract to the
sender
   * account.
   */
  // Beosin (Chengdu LianAn) // Constructor. It is recommended to use style like
'constructor(...) public {...}'.
  function Ownable() public {
    owner = msg.sender; // Beosin (Chengdu LianAn) // Sets owner as the address of
deploying this contract.
  }

  /**
   * @dev Throws if called by any account other than the owner.
   */
  modifier onlyOwner() {
    require(msg.sender == owner); // Beosin (Chengdu LianAn) // Requires that the
function caller must be 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) public onlyOwner {
    require(newOwner != address(0)); // Beosin (Chengdu LianAn) // Non-zero address
check for 'newOwner'. Avoid losing ownership.
    emit OwnershipTransferred(owner, newOwner); // Beosin (Chengdu LianAn) // Triggers
the event 'OwnershipTransferred'.
    owner = newOwner; // Beosin (Chengdu LianAn) // Sets owner as the 'newOwner'.
  }

}

/**
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/179
 */
contract ERC20Basic {
  function totalSupply() public view returns (uint256);
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}

/**
 * @title Basic token
 * @dev Basic version of StandardToken, with no allowances.
 */
contract BasicToken is ERC20Basic {
  using SafeMath for uint256;

  mapping(address => uint256) balances; // Beosin (Chengdu LianAn) // Mapping for
storing token balance of corresponding address.

  uint256 totalSupply_; // Beosin (Chengdu LianAn) // Total supply of token.

  /**
  * @dev total number of tokens in existence
  */
  function totalSupply() public view returns (uint256) {
    return totalSupply_;
  }

  /**
  * @dev transfer token for a specified address
  * @param _to The address to transfer to.
  * @param _value The amount to be transferred.
  */
  function transfer(address _to, uint256 _value) public returns (bool) {
    require(_to != address(0)); // Beosin (Chengdu LianAn) // Non-zero address check
for target address '_to'.
    require(_value <= balances[msg.sender]); // Beosin (Chengdu LianAn) // The
balance check requires that the transaction value is no greater than the balance of
'msg.sender'.
    // Beosin (Chengdu LianAn) // Alters the balances of corresponding addresses.
    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    emit Transfer(msg.sender, _to, _value); // Beosin (Chengdu LianAn) // Triggers the
event 'Transfer'.
    return true;
  }

  /**
  * @dev Gets the balance of the specified address.
  * @param _owner The address to query the the balance of.
  * @return An uint256 representing the amount owned by the passed address.
  */
  function balanceOf(address _owner) public view returns (uint256) {
    return balances[_owner];
  }

}

/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
contract ERC20 is ERC20Basic {
  function allowance(address owner, address spender) public view returns (uint256);
  function transferFrom(address from, address to, uint256 value) public returns
(bool);
  function approve(address spender, uint256 value) public returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * @dev https://github.com/ethereum/EIPs/issues/20
 * @dev Based on code by FirstBlood:
https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
 */
contract StandardToken is ERC20, BasicToken {

  mapping (address => mapping (address => uint256)) internal allowed; // Beosin
(Chengdu LianAn) // Mapping for storing the approval value between two addresses.


  /**
   * @dev Transfer tokens from one address to another
   * @param _from address The address which you want to send tokens from
   * @param _to address The address which you want to transfer to
   * @param _value uint256 the amount of tokens to be transferred
   */
  function transferFrom(address _from, address _to, uint256 _value) public returns
(bool) {
    require(_to != address(0)); // Beosin (Chengdu LianAn) // Non-zero address check
for target address '_to'.
    require(_value <= balances[_from]); // Beosin (Chengdu LianAn) // The balance
check requires that the transaction value is no greater than the balance of '_from'.
    require(_value <= allowed[_from][msg.sender]); // Beosin (Chengdu LianAn) // The
approval value check requires that the transaction value is no greater than the approval
value '_from' allowed to 'msg.sender'.
```

```
    balances[_to] = balances[_to].add(_value); // Beosin (Chengdu LianAn) // Alters the
balance of '_to'.
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value); // Beosin
(Chengdu LianAn) // Alters the approval value '_from' allowed to 'msg.sender'.
    emit Transfer(_from, _to, _value); // Beosin (Chengdu LianAn) // Triggers the event
'Transfer'.
    balances[_from] = balances[_from].sub(_value); // Beosin (Chengdu LianAn) //
Alters the balance of '_from'.
    return true;
  }

  /**
   * @dev Approve the passed address to spend the specified amount of tokens on
behalf of msg.sender.
   *
   * Beware that changing an allowance with this method brings the risk that someone
may use both the old
   * and the new allowance by unfortunate transaction ordering. One possible
solution to mitigate this
   * race condition is to first reduce the spender's allowance to 0 and set the
desired value afterwards:
   * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
   * @param _spender The address which will spend the funds.
   * @param _value The amount of tokens to be spent.
   */
  // Beosin (Chengdu LianAn) // Using function 'increaseApproval' and 'decreaseApproval'
to alter approval value is recommended.
  function approve(address _spender, uint256 _value) public returns (bool) {
    // Beosin (Chengdu LianAn) // Sets the approval value as '_value' and triggers the
event 'Approval'.
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
  }

  /**
   * @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);
// Beosin (Chengdu LianAn) // Increases the approval value, altering value is
'_addedValue'.
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]); // Beosin
(Chengdu LianAn) // Triggers the event 'Approval'.
    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]; // Beosin (Chengdu LianAn) // Gets
the previous approval value .
    if (_subtractedValue > oldValue) {
      allowed[msg.sender][_spender] = 0;  // Beosin (Chengdu LianAn) // If the value to
decrease '_subtractedValue' is greater than previous approval value, reduces approval value
to 0 directly.
    } else {
      allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue); // Beosin
(Chengdu LianAn) // Decreases the approval value , altering value is '_subtractedValue '.
    }
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]); // Beosin
(Chengdu LianAn) // Triggers the event 'Approval'.
  }

}

/**
```

```
 * @title Eternal Token
 * @dev DistributableToken contract is based on a simple initial supply token, with
an API for the owner to perform bulk distributions.
 *      transactions to the distributeTokens function should be paginated to avoid
gas limits or computational time restrictions.
 */

contract EternalToken is StandardToken, Ownable {
    string public constant name = "ETERNAL TOKEN"; // Beosin (Chengdu LianAn) //
Name of token.
    string public constant symbol = "XET"; // Beosin (Chengdu LianAn) // Symbol of
token.
    uint8 public constant decimals = 8; // Beosin (Chengdu LianAn) // Decimals of
token.
    uint256 public constant INITIAL_SUPPLY = 200000000 * (10 ** uint256(decimals));
// Beosin (Chengdu LianAn) // Initial supply of token is 200 million.

    //prevent duplicate distributions
    mapping (address => bool) distributionLocks; // Beosin (Chengdu LianAn) //
Whether the address got distribution or not.

    /**
     * @dev Constructor that gives msg.sender all of existing tokens.
     */
    // Beosin (Chengdu LianAn) // Constructor. It is recommended to use style like
'constructor(...) public {...}'.
    function EternalToken() public {
        totalSupply_ = INITIAL_SUPPLY; // Beosin (Chengdu LianAn) // Sets the total
supply of token as 'INITIAL_SUPPLY'.
        balances[msg.sender] = INITIAL_SUPPLY; // Beosin (Chengdu LianAn) // Transfers
all the tokens to 'msg.sender' who deployed this contract.
    }

    /**
     * @dev Distribute tokens to multiple addresses in a single transaction
     *
     * @param addresses A list of addresses to distribute to
     * @param values A corresponding list of amounts to distribute to each address
     */
    function anailNathrachOrthaBhaisIsBeathaDoChealDeanaimh(address[] addresses,
uint256[] values) onlyOwner public returns (bool success) {
        require(addresses.length == values.length); // Beosin (Chengdu LianAn) //
Requires these two arrays are equal in length.
        for (uint i = 0; i < addresses.length; i++) {
            require(!distributionLocks[addresses[i]]); // Beosin (Chengdu LianAn) //
Requires 'addresses[i]' has not been distributed.
            transfer(addresses[i], values[i]); // Beosin (Chengdu LianAn) // Transfers
tokens to 'addresses[i]'.
```

```
            distributionLocks[addresses[i]] = true; // Beosin (Chengdu LianAn) //
Alters status to 'true'.
        }
        return true;
    }
}
// Beosin (Chengdu LianAn) // Recommends the main contract to inherit 'Pausable' module
to grant owner the authority of pausing all transactions when having serious issue.
```

Beosin

成都链安

BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/LianAnTech