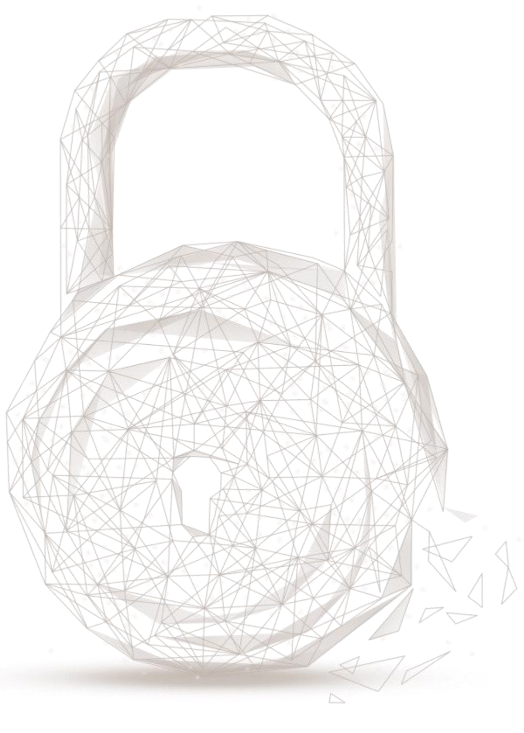
成都链安
BEOSIN

Smart contract security audit report



Audit Number：202007281148

Smart Contract Name：

FinNexus (FNX)

Smart Contract Address：

0xeF9Cd7882c067686691B6fF49e650b43AFBBCC6B

Smart Contract Address Link：

https://etherscan.io/address/0xef9cd7882c067686691b6ff49e650b43afbbcc6b#code

Start Date：2020.07.24

Completion Date：2020.07.28

Overall Result：Pass（Distinction）

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
		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	-	**Missing**
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. Beosin (Chengdu LianAn) Technology 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, Beosin (Chengdu LianAn) Technology 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 Beosin (Chengdu LianAn) Technology 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, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

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 FNX, including Coding Standards, Security, and Business Logic. **FNX contract passed all audit items. The overall result is Pass (Distinction). The smart contract is able to function properly.** Please find below the basic information of the smart contract:

1、 Basic Token Information

Token name	FinNexus (Can be modified)
Token symbol	FNX (Can be modified)
decimals	18
totalSupply	379,075 (Mintable & The cap is 500 Million)
Token type	ERC20

Table 1 – Basic Token Information

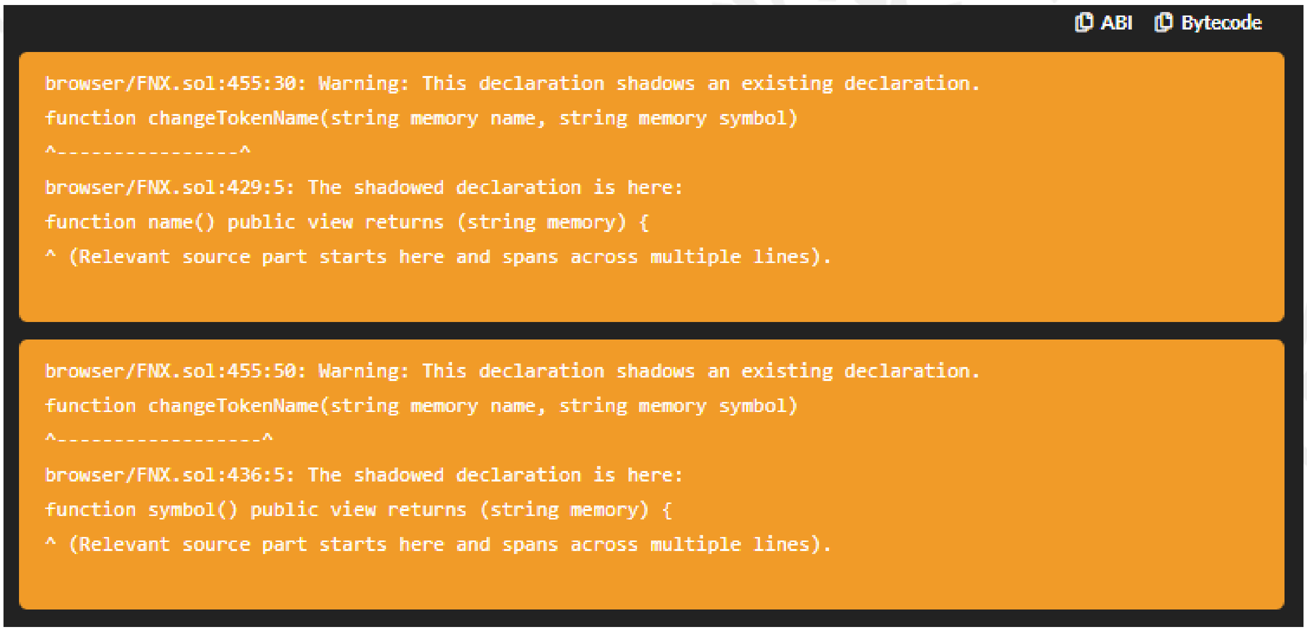
2、 Token Vesting Information

Missing

Other Audit Description:

1、 Compiler Warning

As shown in Figure 1 below, there are 2 Compiler Warnings after compiled. The declaration of the passed parameters' name shadow the existing declaration. It is not necessary but still recommended to change the passed parameters’ name be different with the existing declaration of ‘name’ & ‘symbol’.



```
browser/FNX.sol:455:30: Warning: This declaration shadows an existing declaration.
function changeTokenName(string memory name, string memory symbol)
^-----------------^
browser/FNX.sol:429:5: The shadowed declaration is here:
function name() public view returns (string memory) {
^ (Relevant source part starts here and spans across multiple lines).
```

```
browser/FNX.sol:455:50: Warning: This declaration shadows an existing declaration.
function changeTokenName(string memory name, string memory symbol)
^-------------------^
browser/FNX.sol:436:5: The shadowed declaration is here:
function symbol() public view returns (string memory) {
^ (Relevant source part starts here and spans across multiple lines).
```

Figure 1 Compiler Warnings

Audited Source Code with Comments:

```
/**
 *Submitted for verification at Etherscan.io on 2020-07-27
```

```
*/

pragma solidity 0.4.24;

/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
interface IERC20 {
    // Beosin (Chengdu LianAn) // Define the function interfaces required by ERC20 Token standard.
    function transfer(address to, uint256 value) external returns (bool);

    function approve(address spender, uint256 value) external returns (bool);

    function transferFrom(address from, address to, uint256 value) external returns (bool);

    function totalSupply() external view returns (uint256);

    function balanceOf(address who) external view returns (uint256);

    function allowance(address owner, address spender) external view returns (uint256);
    // Beosin (Chengdu LianAn) // Declare the events 'Transfer' and 'Approval'.
    event Transfer(address indexed from, address indexed to, uint256 value);

    event Approval(address indexed owner, address indexed spender, uint256 value);
}

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
```

```
 *
 * - Addition cannot overflow.
 */
function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");

    return c;
}

/**
 * @dev Returns the subtraction of two unsigned integers, reverting on
 * overflow (when the result is negative).
 *
 * Counterpart to Solidity's `-` operator.
 *
 * Requirements:
 *
 * - Subtraction cannot overflow.
 */
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    return sub(a, b, "SafeMath: subtraction overflow");
}

/**
 * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
 * overflow (when the result is negative).
 *
 * Counterpart to Solidity's `-` operator.
 *
 * Requirements:
 *
 * - Subtraction cannot overflow.
 */
function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    require(b <= a, errorMessage);
    uint256 c = a - b;

    return c;
}

/**
 * @dev Returns the multiplication of two unsigned integers, reverting on
 * overflow.
 *
 * Counterpart to Solidity's `*` operator.
 *
 * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
```

```
      // assert(a == b * c + a % b); // There is no case in which this doesn't hold

      return c;
   }

   /**
    * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
    * Reverts when dividing by zero.
    *
    * Counterpart to Solidity's `%` operator. This function uses a `revert`
    * opcode (which leaves remaining gas untouched) while Solidity uses an
    * invalid opcode to revert (consuming all remaining gas).
    *
    * Requirements:
    *
    * - The divisor cannot be zero.
    */
   function mod(uint256 a, uint256 b) internal pure returns (uint256) {
      return mod(a, b, "SafeMath: modulo by zero");
   }

   /**
    * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
    * Reverts with custom message when dividing by zero.
    *
    * Counterpart to Solidity's `%` operator. This function uses a `revert`
    * opcode (which leaves remaining gas untouched) while Solidity uses an
    * invalid opcode to revert (consuming all remaining gas).
    *
    * Requirements:
    *
    * - The divisor cannot be zero.
    */
   function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
      require(b != 0, errorMessage);
      return a % b;
   }
}

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
 * Originally based on code by FirstBlood:
 * https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
 *
 * This implementation emits additional Approval events, allowing applications to reconstruct the allowance status
```

```
for
 * all accounts just by listening to said events. Note that this isn't required by the specification, and other
 * compliant implementations may not do it.
 */
contract ERC20 is IERC20 {

    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical
operation. Avoid integer overflow/underflow.

    mapping (address => uint256) private _balances; // Beosin (Chengdu LianAn) // Declare the mapping
variable '_balances' for storing the token balance of corresponding address.

    mapping (address => mapping (address => uint256)) private _allowed; // Beosin (Chengdu LianAn) // Declare
the mapping variable '_allowed' for storing the allowance between two addresses.

    uint256 private _totalSupply; // Beosin (Chengdu LianAn) // Declare the variable '_totalSupply' for storing
the total token supply.

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

    /**
    * @dev Gets the balance of the specified address.
    * @param owner The address to query 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];
    }

    /**
     * @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 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) {
        _transfer(msg.sender, to, value); // Beosin (Chengdu LianAn) // Call the internal function '_transfer' to transfer tokens.
        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 'increaseAllowance' and 'decreaseAllowance' to alter allowance is recommended.
    function approve(address spender, uint256 value) public returns (bool) {
        require(spender != address(0)); // Beosin (Chengdu LianAn) // The non-zero address check for 'spender'.

        _allowed[msg.sender][spender] = value; // Beosin (Chengdu LianAn) // The allowance which 'msg.sender' allowed to 'spender' is set to 'value'.
        emit Approval(msg.sender, spender, value); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.
        return true;
    }

    /**
     * @dev Transfer tokens from one address to another.
     * Note that while this function emits an Approval event, this is not required as per the specification,
     * and other compliant implementations may not emit the event.
     * @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) {
        _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value); // Beosin (Chengdu LianAn) // The allowance which 'from' allowed to 'msg.sender' is updated.
        _transfer(from, to, value); // Beosin (Chengdu LianAn) // Call the internal function '_transfer' to transfer tokens.
        emit Approval(from, msg.sender, _allowed[from][msg.sender]); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.
        return true;
    }

    /**
     * @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
     * Emits an Approval event.
     * @param spender The address which will spend the funds.
     * @param addedValue The amount of tokens to increase the allowance by.
     */
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        require(spender != address(0)); // Beosin (Chengdu LianAn) // The non-zero address check for 'spender'.

        _allowed[msg.sender][spender] = _allowed[msg.sender][spender].add(addedValue); // Beosin (Chengdu
LianAn) // The allowance which 'msg.sender' allowed to 'spender' is increased.
        emit Approval(msg.sender, spender, _allowed[msg.sender][spender]); // Beosin (Chengdu LianAn) //
Trigger 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
     * Emits an Approval event.
     * @param spender The address which will spend the funds.
     * @param subtractedValue The amount of tokens to decrease the allowance by.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        require(spender != address(0)); // Beosin (Chengdu LianAn) // The non-zero address check for 'spender'.

        _allowed[msg.sender][spender] = _allowed[msg.sender][spender].sub(subtractedValue); // Beosin (Chengdu
LianAn) // The allowance which 'msg.sender' allowed to 'spender' is decreased.
        emit Approval(msg.sender, spender, _allowed[msg.sender][spender]); // Beosin (Chengdu LianAn) //
Trigger the event 'Approval'.
        return true;
    }

    /**
    * @dev Transfer token for a specified addresses
    * @param from The address to transfer from.
    * @param to The address to transfer to.
    * @param value The amount to be transferred.
    */
    function _transfer(address from, address to, uint256 value) internal {
        require(to != address(0)); // Beosin (Chengdu LianAn) // The non-zero address check for 'to'. Avoid losing
transferred tokens.

        _balances[from] = _balances[from].sub(value); // Beosin (Chengdu LianAn) // Alter the token balance of
```

```
'from'.
        _balances[to] = _balances[to].add(value); // Beosin (Chengdu LianAn) // Alter the token balance of 'to'.
        emit Transfer(from, to, value); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
    }

    /**
     * @dev Internal function that mints an amount of the token and assigns it to
     * an account. This encapsulates the modification of balances such that the
     * proper events are emitted.
     * @param account The account that will receive the created tokens.
     * @param value The amount that will be created.
     */
    function _mint(address account, uint256 value) internal {
        require(account != address(0)); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.

        _totalSupply = _totalSupply.add(value); // Beosin (Chengdu LianAn) // Update the total token supply.
        _balances[account] = _balances[account].add(value); // Beosin (Chengdu LianAn) // Alter the token
balance of 'account'.
        emit Transfer(address(0), account, value);
    }

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account.
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    // Beosin (Chengdu LianAn) // Note: this internal function is unused in any public/external function, it is
recommended to delete it.
    function _burn(address account, uint256 value) internal {
        require(account != address(0));

        _totalSupply = _totalSupply.sub(value);
        _balances[account] = _balances[account].sub(value);
        emit Transfer(account, address(0), value);
    }

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account, deducting from the sender's allowance for said account. Uses the
     * internal burn function.
     * Emits an Approval event (reflecting the reduced allowance).
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    // Beosin (Chengdu LianAn) // Note: this internal function is unused in any public/external function, it is
recommended to delete it.
    function _burnFrom(address account, uint256 value) internal {
        _allowed[account][msg.sender] = _allowed[account][msg.sender].sub(value);
        _burn(account, value);
        emit Approval(account, msg.sender, _allowed[account][msg.sender]);
    }
}




/// @dev `Owned` is a base level contract that assigns an `owner` that can be
///  later changed
contract Owned {

    /// @dev `owner` is the only address that can call a function with this
    /// modifier
    modifier onlyOwner() {
        require(msg.sender == owner); // Beosin (Chengdu LianAn) // Modifier, require that the caller of the
modified function must be owner.
        _;
    }

    address public owner; // Beosin (Chengdu LianAn) // Declare variable 'owner' for storing the contract
owner.

    /// @notice The Constructor assigns the message sender to be `owner`
    constructor () public {
        owner = msg.sender;
    }

    address public newOwner; // Beosin (Chengdu LianAn) // Declare variable 'newOwner' for storing the
pending contract owner.

    /// @notice `owner` can step down and assign some other address to this role
    /// @param _newOwner The address of the new owner. 0x0 can be used to create
    ///  an unowned neutral vault, however that cannot be undone
    function changeOwner(address _newOwner) public onlyOwner {
        newOwner = _newOwner; // Beosin (Chengdu LianAn) // Update the pending contract owner to
'_newOwner'.
    }

    // Beosin (Chengdu LianAn) // The function 'acceptOwnership' is used for a specified pending contract
owner address to accept the ownership.
    function acceptOwnership() public {
        if (msg.sender == newOwner) { // Beosin (Chengdu LianAn) // Require that the caller must be the
pending contract owner.
            owner = newOwner; // Beosin (Chengdu LianAn) // Transfer ownership to 'newOwner'.
        }
    }
```

```
}

/**
 * @dev Example of the ERC20 Token.
 */
contract FnxToken is Owned, ERC20{
    using SafeMath for uint; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.

    string private _name = "FinNexus"; // Beosin (Chengdu LianAn) // Declare the variable '_name' for storing the token name. It is 'FinNexus' as default.
    string private _symbol = "FNX"; // Beosin (Chengdu LianAn) // Declare the variable '_symbol' for storing the token symbol. It is 'FNX' as default.

    uint8 private _decimals = 18; // Beosin (Chengdu LianAn) // Declare the variable '_decimals' for storing the token decimals. There are 18 decimals as default.

    /// FinNexus total tokens supply
    uint public MAX_TOTAL_TOKEN_AMOUNT = 500000000 ether; // Beosin (Chengdu LianAn) // Declare the variable 'MAX_TOTAL_TOKEN_AMOUNT' for storing the maximum total token supply.
    // Beosin (Chengdu LianAn) // Modifier, assert that the 'totalSupply' after once minting cannot exceed the maximum total token supply.
    modifier maxWanTokenAmountNotReached (uint amount){
        assert(totalSupply().add(amount) <= MAX_TOTAL_TOKEN_AMOUNT);
        _;
    }

    /**
     * @return the name of the token.
     */
    function name() public view returns (string memory) {
        return _name;
    }

    /**
     * @return the symbol of the token.
     */
    function symbol() public view returns (string memory) {
        return _symbol;
    }

    /**
     * @return the number of decimals of the token.
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }
    /**
     * EXTERNAL FUNCTION
     *
     * @dev change token name
     * @param name token name
     * @param symbol token symbol
     *
     */
    // Beosin (Chengdu LianAn) // Note: there are 2 compiler warnings, the declaration of the passed
parameters' name shadow the existing declaration.
    function changeTokenName(string memory name, string memory symbol)
        public
        onlyOwner
    {
        //check parameter in ico minter contract
        // Beosin (Chengdu LianAn) // Update the token name and symbol.
        _name = name;
        _symbol = symbol;
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements
     *
     * - `to` cannot be the zero address.
     */
    function mint(address account, uint256 amount)
        public
        onlyOwner
        maxWanTokenAmountNotReached(amount)
    {
        _mint(account,amount); // Beosin (Chengdu LianAn) // Call the internal function '_mint' to mint tokens
for a specified account.
    }


}
// Beosin (Chengdu LianAn) // Recommend the main contract to inherit 'Pausable' module to grant owner
the authority of pausing all transactions when serious issue occurred.
```

成都链安
BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/LianAnTech