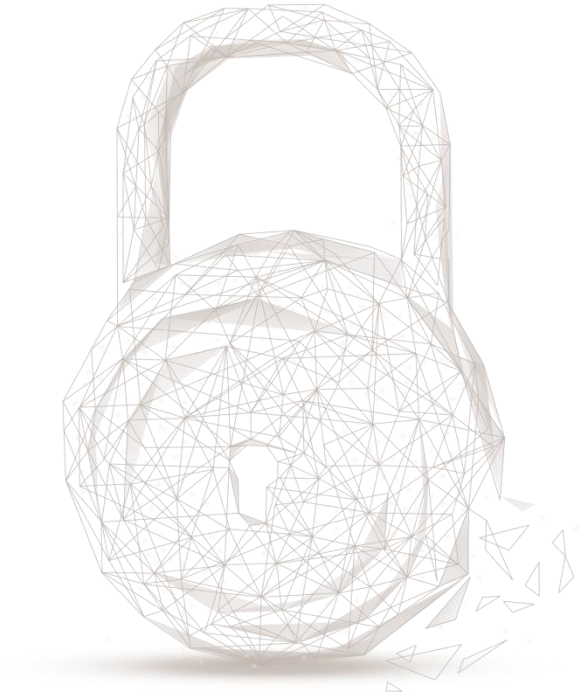
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202012241418

Smart Contract Name：

ZEN Exchange Token (ZCX)

Smart Contract Information：

Contract name	Smart Contract Address	Smart Contract Address Link
AdminUpgradeabilityProxy	0xC52C326331E9Ce41F04484d3B5E5648158028804	https://etherscan.io/address/0xc52c326331e9ce41f04484d3b5e5648158028804#code
ZCX	0x50d44bCfc10Bc0A752388F173F482FaAB87B3445	https://etherscan.io/address/0x50d44bcfc10bc0a752388f173f482faab87b3445#code

Start Date：2020.12.17

Completion Date：2020.12.24

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	-	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. 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 ZCX,

including Coding Standards, Security, and Business Logic. **ZCX 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	ZEN Exchange Token
Token symbol	ZCX
decimals	18
totalSupply	Initial supply is 1 Billion (Burnable)
Token type	ERC20

Table 1 – Basic Token Information

2、Token Vesting Information

N/A

3、Token Proxy Description

ZCX contract set-up a proxy contract architecture that will allow to use new deployed contracts.A proxy architecture mode is such that all message calls go through a Proxy contract that will redirect them to the latest deployed contract logic. If the logic contract needs to be upgraded, a new contract can be deployed, and the Proxy updates to reference the new contract address.

Audited Source Code with Comments

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.6.2; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.


library Address {
    /**
     * @dev Returns true if ‘account’ is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, ‘isContract’ will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
```

```
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) } // Beosin (Chengdu LianAn) // Get the code length of the specified address.
        return size > 0;
    }

    /**
     * @dev Replacement for Solidity's 'transfer': sends 'amount' wei to
     * 'recipient', forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by 'transfer', making them unable to receive funds via
     * 'transfer'. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to 'recipient', care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    /**
     * @dev Performs a Solidity function call using a low level 'call'. A
     * plain'call' is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If 'target' reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-
```

```
encoding-and-decoding-functions[‘abi.decode’].
     *
     * Requirements:
     *
     * - ‘target’ must be a contract.
     * - calling ‘target’ with ‘data’ must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[‘functionCall’], but with
     * ‘errorMessage’ as a fallback revert reason when ‘target’ reverts.
     *
     *  Available since v3.1.
     */
    function functionCall(address target, bytes memory data, string memory errorMessage) internal returns
(bytes memory) {
        return _functionCallWithValue(target, data, 0, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[‘functionCall’],
     * but also transferring ‘value’ wei to ‘target’.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least ‘value’.
     * - the called Solidity function must be ‘payable’.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes
memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-
}[‘functionCallWithValue’], but
     * with ‘errorMessage’ as a fallback revert reason when ‘target’ reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory
```

```
errorMessage) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        return _functionCallWithValue(target, data, value, errorMessage);
    }

    function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory
errorMessage) private returns (bytes memory) {
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}




abstract contract Proxy {
    /**
     * @dev Fallback function.
     * Implemented entirely in `_fallback`.
     */
    fallback() external payable {
        _fallback();
    }

    /**
     * @dev Receive function.
     * Implemented entirely in `_fallback`.
     */
    receive() external payable {
```

```
        _fallback(); // Beosin (Chengdu LianAn) // Call the internal function '_fallback'.
    }

    /**
     * @return The Address of the implementation.
     */
    function  implementation() internal virtual view returns (address);

    /**
     * @dev Delegates execution to an implementation contract.
     * This is a low level function that doesn't return to its internal call site.
     * It will return to the external caller whatever the implementation returns.
     * @param implementation Address to delegate.
     */
    function  delegate(address implementation) internal {
        assembly {
            // Copy msg.data. We take full control of memory in this inline assembly
            // block because it will not return to Solidity code. We overwrite the
            // Solidity scratch pad at memory position 0.
            calldatacopy(0, 0, calldatasize())

            // Call the implementation.
            // out and outsize are 0 because we don't know the size yet.
            let result := delegatecall(
                gas(),
                implementation,
                0,
                calldatasize(),
                0,
                0
            )

            // Copy the returned data.
            returndatacopy(0, 0, returndatasize())

            switch result
                // delegatecall returns 0 on error.
                case 0 {
                    revert(0, returndatasize()) // Beosin (Chengdu LianAn) // Revert when the
returned call result is 0.
                }
                default {
                    return(0, returndatasize())
                }
        }
    }

    /**
```

```
     * @dev Function that is run as the first thing in the fallback function.
     * Can be redefined in derived contracts to add functionality.
     * Redefinitions must call super._willFallback().
     */
    function _willFallback() internal virtual {}

    /**
     * @dev fallback implementation.
     * Extracted to enable manual triggering.
     */
    function  fallback() internal {
        _willFallback();
        _delegate(_implementation());
    }
}



contract UpgradeabilityProxy is Proxy {
  /**
   * @dev Contract constructor.
   * @param _logic Address of the initial implementation.
   * @param _data Data to send as msg.data to the implementation to initialize the proxied contract.
   * It should include the signature and the parameters of the function to be called, as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding.
   * This parameter is optional, if no data is given the initialization call to proxied contract will be skipped.
   */
  constructor(address _logic, bytes memory _data) public payable {
    assert(IMPLEMENTATION_SLOT == bytes32(uint256(keccak256('eip1967.proxy.implementation')) -
1)); // Beosin (Chengdu LianAn) // Using require to check is recommended.
    _setImplementation(_logic); // Beosin (Chengdu LianAn) // Call function '_setImplementation' to set
up implementation contract.
    if(_data.length > 0) {
      (bool success,) = _logic.delegatecall(_data); // Beosin (Chengdu LianAn) // Delegate calls the
specified implementation contract's function in the environment of this contract.
      require(success); // Beosin (Chengdu LianAn) // Check the call result.
    }
  }

  /**
   * @dev Emitted when the implementation is upgraded.
   * @param implementation Address of the new implementation.
   */
  event Upgraded(address indexed implementation);

  /**
   * @dev Storage slot with the address of the current implementation.
```

```
    * This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1, and is
    * validated in the constructor.
    */
   bytes32 internal constant IMPLEMENTATION_SLOT =
0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc;

   /**
    * @dev Returns the current implementation.
    * @return impl Address of the current implementation
    */
   function   implementation() internal override view returns (address impl) {
      bytes32 slot = IMPLEMENTATION_SLOT;
      assembly {
         impl := sload(slot) // Beosin (Chengdu LianAn) // Return the current implementation contract
address.
      }
   }

   /**
    * @dev Upgrades the proxy to a new implementation.
    * @param newImplementation Address of the new implementation.
    */
   function _upgradeTo(address newImplementation) internal {
      _setImplementation(newImplementation); // Beosin (Chengdu LianAn) // Call the function
'_setImplementation' to set the new implementation contract address.
      emit Upgraded(newImplementation); // Beosin (Chengdu LianAn) // Trigger the implementation
contract address upgrade event.
   }

   /**
    * @dev Sets the implementation address of the proxy.
    * @param newImplementation Address of the new implementation.
    */
   function _setImplementation(address newImplementation) internal {
      require(Address.isContract(newImplementation), "Cannot set a proxy implementation to a non-contract
address"); // Beosin (Chengdu LianAn) // Require 'newImplementation' to be a contract address.

      bytes32 slot = IMPLEMENTATION_SLOT;

      assembly {
         sstore(slot, newImplementation) // Beosin (Chengdu LianAn) // Update the storage record of the
implementation contract address.
      }
   }
}
```

```
contract AdminUpgradeabilityProxy is UpgradeabilityProxy {

    constructor(
        address _logic,
        address   admin,
        bytes memory _data
    ) public payable UpgradeabilityProxy(_logic, _data) {
        assert(
            ADMIN  SLOT == bytes32(uint256(keccak256("eip1967.proxy.admin")) - 1) // Beosin
(Chengdu LianAn) // Check the default admin.
        );
          setAdmin(  admin); // Beosin (Chengdu LianAn) // Call function '_setAdmin' to set admin.
    }

    /**
     * @dev Emitted when the administration has been transferred.
     * @param previousAdmin Address of the previous admin.
     * @param newAdmin Address of the new admin.
     */
    event AdminChanged(address previousAdmin, address newAdmin);

    /**
     * @dev Storage slot with the admin of the contract.
     * This is the keccak-256 hash of "eip1967.proxy.admin" subtracted by 1, and is
     * validated in the constructor.
     */

    bytes32
        internal constant ADMIN_SLOT =
0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103;

    /**
     * @dev Modifier to check whether the ‘msg.sender’ is the admin.
     * If it is, it will run the function. Otherwise, it will delegate the call
     * to the implementation.
     */
    modifier ifAdmin() {
        if (msg.sender == _admin()) {
            _;
        } else {
            _fallback(); // Beosin (Chengdu LianAn) // Call function '_fallback' to execute the
specified function operation.
        }
    }
```

```
    /**
     * @return The address of the proxy admin.
     */
    function admin() external ifAdmin returns (address) {
        return _admin(); // Beosin (Chengdu LianAn) // Return the current admin address.
    }

    /**
     * @return The address of the implementation.
     */
    function implementation() external ifAdmin returns (address) {
        return _implementation(); // Beosin (Chengdu LianAn) // Return the current implementation contract address.
    }

    /**
     * @dev Changes the admin of the proxy.
     * Only the current admin can call this function.
     * @param newAdmin Address to transfer proxy administration to.
     */
    function changeAdmin(address newAdmin) external ifAdmin {
        require(
            newAdmin != address(0),
            "Cannot change the admin of a proxy to the zero address"
        );
        emit AdminChanged(_admin(), newAdmin);
        _setAdmin(newAdmin); // Beosin (Chengdu LianAn) // Call function '_setAdmin' to set admin address.
    }

    /**
     * @dev Upgrade the backing implementation of the proxy.
     * Only the admin can call this function.
     * @param newImplementation Address of the new implementation.
     */
    function upgradeTo(address newImplementation) external ifAdmin {
        _upgradeTo(newImplementation); // Beosin (Chengdu LianAn) // Call function '_upgradeTo' to update the implementation contract address.
    }

    /**
     * @dev Upgrade the backing implementation of the proxy and call a function
     * on the new implementation.
     * This is useful to initialize the proxied contract.
     * @param newImplementation Address of the new implementation.
     * @param data Data to send as msg.data in the low level call.
     * It should include the signature and the parameters of the function to be called, as described in
     * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding.
```

```
     */

    function upgradeToAndCall(address newImplementation, bytes calldata data)
        external
        payable
        ifAdmin
    {
        _upgradeTo(newImplementation); // Beosin (Chengdu LianAn) // Call function '_upgradeTo' to update the implementation contract address.
        (bool success, ) = newImplementation.delegatecall(data); // Beosin (Chengdu LianAn) // Delegate calls implementation contract function with '_data' as transaction data.
        require(success); // Beosin (Chengdu LianAn) // Check the call result.
    }

    /**
     * @return adm The admin slot.
     */
    function   admin() internal view returns (address adm) {
        bytes32 slot = ADMIN  SLOT;
        assembly {
            adm := sload(slot) // Beosin (Chengdu LianAn) // Return the current admin address.
        }
    }

    /**
     * @dev Sets the address of the proxy admin.
     * @param newAdmin Address of the new proxy admin.
     */
    function _setAdmin(address newAdmin) internal {
        bytes32 slot = ADMIN_SLOT;

        assembly {
            sstore(slot, newAdmin) // Beosin (Chengdu LianAn) // Update the current admin address.
        }
    }

    /**
     * @dev Only fall back when the sender is not the admin.
     */
    function _willFallback() internal virtual override {
        require(
            msg.sender != _admin(),
            "Cannot call fallback function from the proxy admin"
        );
        super._willFallback(); // Beosin (Chengdu LianAn) // Call the parent contract function '_willFallback'.
    }
}
```

```
// contracts/ZCX.sol
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.8.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.


abstract contract Initializable {

    /**
     * @dev Indicates that the contract has been initialized.
     */
    bool private   initialized;

    /**
     * @dev Indicates that the contract is in the process of being initialized.
     */
    bool private _initializing;

    /**
     * @dev Modifier to protect an initializer function from being invoked twice.
     */
    // Beosin (Chengdu LianAn) // Require the modified function should not be initialized.
    modifier initializer() {
        require(_initializing || _isConstructor() || !_initialized, "Initializable: contract is already initialized");

        bool isTopLevelCall = !_initializing;
        if (isTopLevelCall) {
            _initializing = true;
            _initialized = true;
        }

        _;

        if (isTopLevelCall) {
            _initializing = false;
        }
    }

    /// @dev Returns true if and only if the function is running in the constructor
    function _isConstructor() private view returns (bool) {
        // extcodesize checks the size of the code stored in an address, and
        // address returns the current address. Since the code is still not
        // deployed when running a constructor, any checks on its code size will
        // yield zero, making it an effective way to detect if a contract is
        // under construction or not.
```

```
        address self = address(this);
        uint256 cs;
        // solhint-disable-next-line no-inline-assembly
        assembly { cs := extcodesize(self) }
        return cs == 0;
    }
}



abstract contract ContextUpgradeable is Initializable {
    function __Context_init() internal initializer {
        __Context_init_unchained();
    }

    function __Context_init_unchained() internal initializer {
    }
    // Beosin (Chengdu LianAn) // Return the sender of the message.
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }
    // Beosin (Chengdu LianAn) // Return the data of the message.
    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
    uint256[50] private __gap;
}


// Beosin (Chengdu LianAn) // Define the interface functions required by the ERC20 Token standard.
interface IERC20Upgradeable {
    /**
     * @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'.
     *
     * Returns a boolean value indicating whether the operation succeeded.
```

```
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: 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
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    // Beosin (Chengdu LianAn) // Declare the events required by the ERC20 Token standard.
```

```
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}


// Beosin (Chengdu LianAn) // The SafeMathUpgradeable library declares functions for safe mathematical operation.
library SafeMathUpgradeable {
    /**
     * @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;
    }
}



contract ERC20Upgradeable is Initializable, ContextUpgradeable, IERC20Upgradeable {
    using SafeMathUpgradeable for uint256; // Beosin (Chengdu LianAn) // Use the SafeMathUpgradeable 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 _allowances; // Beosin (Chengdu LianAn) // Declare the mapping variable '_allowances' for storing the allowance between two addresses.

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

    string private _name; // Beosin (Chengdu LianAn) // Declare the variable '_name' for storing the token name.
    string private _symbol; // Beosin (Chengdu LianAn) // Declare the variable '_symbol' for storing the token symbol.
    uint8 private _decimals; // Beosin (Chengdu LianAn) // Declare the variable '_decimals' for storing the token decimals.

    /**
     * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
     * a default value of 18.
     *
     * To select a different value for {decimals}, use {_setupDecimals}.
     *
     * All three of these values are immutable: they can only be set once during
     * construction.
     */
    function __ERC20_init(string memory name_, string memory symbol_) internal initializer {
        __Context_init_unchained();
        __ERC20_init_unchained(name_, symbol_);
    }
    // Beosin (Chengdu LianAn) // Initialize token information.
    function __ERC20_init_unchained(string memory name_, string memory symbol_) internal initializer {
        _name = name_; // Beosin (Chengdu LianAn) // Initialize the token name.
        _symbol = symbol_; // Beosin (Chengdu LianAn) // Initialize the token symbol.
        _decimals = 18; // Beosin (Chengdu LianAn) // Initialize the token decimals to 18 as default.
    }
```

```
/**
 * @dev Returns the name of the token.
 */
// Beosin (Chengdu LianAn) // Return the token name.
function name() public view returns (string memory) {
    return  name;
}

/**
 * @dev Returns the symbol of the token, usually a shorter version of the
 * name.
 */
// Beosin (Chengdu LianAn) // Return the token symbol.
function symbol() public view returns (string memory) {
    return _symbol;
}

/**
 * @dev Returns the number of decimals used to get its user representation.
 * For example, if ‘decimals’ equals ‘2’, a balance of ‘505’ tokens should
 * be displayed to a user as ‘5,05’ (‘505 / 10 ** 2’).
 *
 * Tokens usually opt for a value of 18, imitating the relationship between
 * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
 * called.
 *
 * NOTE: This information is only used for _display_ purposes: it in
 * no way affects any of the arithmetic of the contract, including
 * {IERC20-balanceOf} and {IERC20-transfer}.
 */
// Beosin (Chengdu LianAn) // Return the token decimals.
function decimals() public view returns (uint8) {
    return _decimals;
}

/**
 * @dev See {IERC20-totalSupply}.
 */
// Beosin (Chengdu LianAn) // Return the total supply of token.
function totalSupply() public view override returns (uint256) {
    return _totalSupply;
}

/**
 * @dev See {IERC20-balanceOf}.
 */
// Beosin (Chengdu LianAn) // Return the balance of the account.
```

```
    function balanceOf(address account) public view override returns (uint256) {
        return _balances[account];
    }

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - 'recipient' cannot be the zero address.
     * - the caller must have a balance of at least 'amount'.
     */
    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        transfer(  msgSender(), recipient, amount); // Beosin (Chengdu LianAn) // Call the internal function '_transfer' to transfer tokens.
        return true;
    }

    /**
     * @dev See {IERC20-allowance}.
     */
    // Beosin (Chengdu LianAn) // Return the allowance which 'owner' allowed to 'spender'.
    function allowance(address owner, address spender) public view virtual override returns (uint256) {
        return _allowances[owner][spender];
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - 'spender' cannot be the zero address.
     */
    // Beosin (Chengdu LianAn) // 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.
    // Beosin (Chengdu LianAn) // Using function 'increaseAllowance' and 'decreaseAllowance' to alter allowance is recommended.
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to set the allowance which 'msg.sender' allowed to 'spender'.
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20}.
```

```
     *
     * Requirements:
     *
     * - ‘sender’ and ‘recipient’ cannot be the zero address.
     * - ‘sender’ must have a balance of at least ‘amount’.
     * - the caller must have allowance for ‘‘sender’’'s tokens of at least
     * ‘amount’.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
        transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Call the internal function '_transfer' to transfer tokens.
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance")); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to alter the allowance between two addresses.
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to ‘spender’ by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - ‘spender’ cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue)); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to increase the allowance between two addresses.
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to ‘spender’ by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - ‘spender’ cannot be the zero address.
```

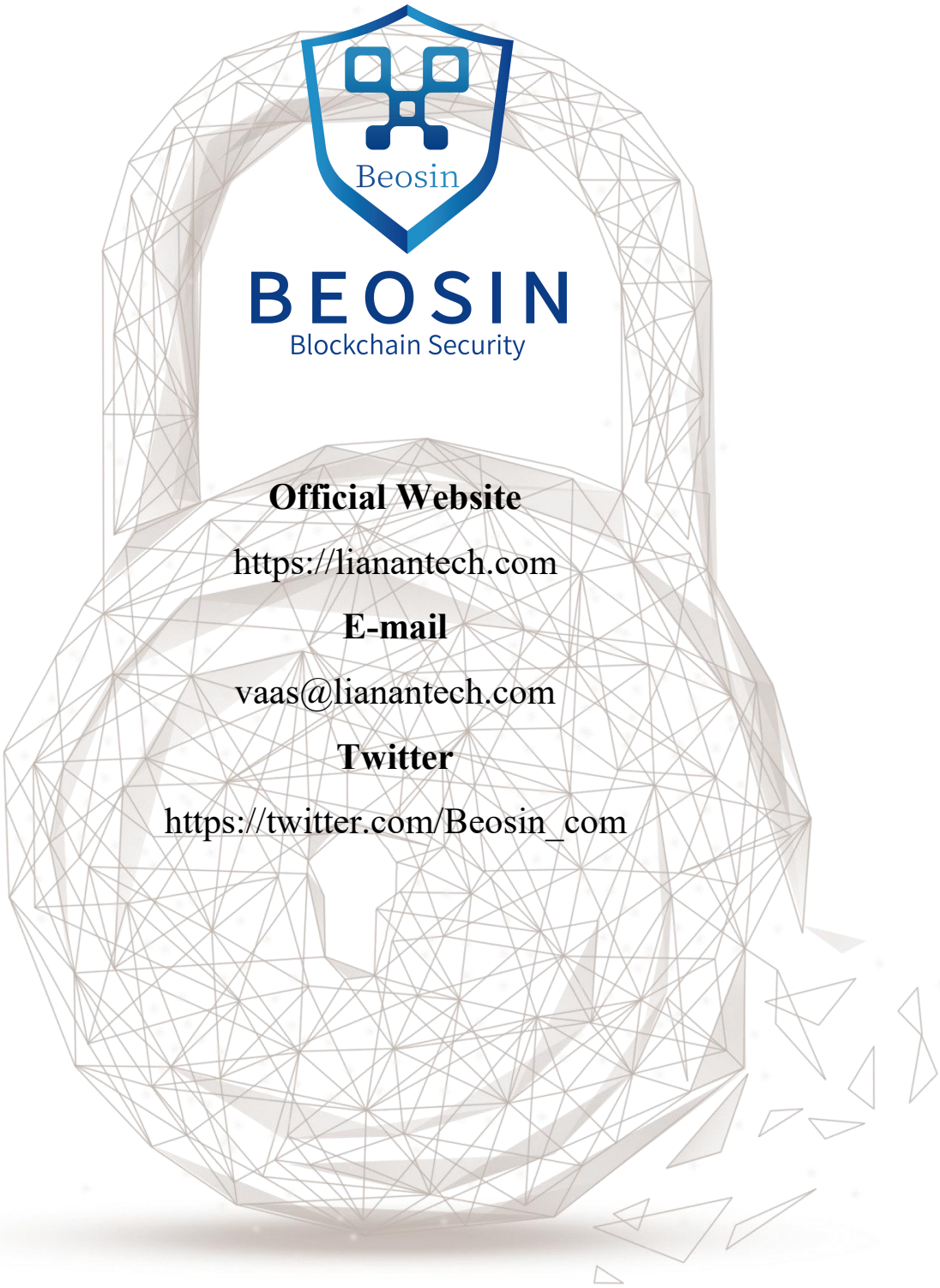
```
     * - ‘spender’ must have allowance for the caller of at least
     * ‘subtractedValue’.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue,
"ERC20: decreased allowance below zero")); // Beosin (Chengdu LianAn) // Call the internal function
'_approve' to decrease the allowance between two addresses.
    }

    /**
     * @dev Moves tokens ‘amount’ from ‘sender’ to ‘recipient’.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - ‘sender’ cannot be the zero address.
     * - ‘recipient’ cannot be the zero address.
     * - ‘sender’ must have a balance of at least ‘amount’.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal virtual {
        require(sender != address(0), "ERC20: transfer from the zero address"); // Beosin (Chengdu
LianAn) // The non-zero address check for 'sender'.
        require(recipient != address(0), "ERC20: transfer to the zero address"); // Beosin (Chengdu
LianAn) // The non-zero address check for 'recipient'. Avoid losing transferred tokens.

        _beforeTokenTransfer(sender, recipient, amount);

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance"); //
Beosin (Chengdu LianAn) // Alter the token balance of 'sender'.
        _balances[recipient] = _balances[recipient].add(amount); // Beosin (Chengdu LianAn) // Alter the
token balance of 'recipient'.
        emit Transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Trigger the event
'Transfer'.
    }

    /** @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) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.

        _beforeTokenTransfer(address(0), account, amount);

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

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance"); // Beosin (Chengdu LianAn) // Alter the token balance of 'account'.
        _totalSupply = _totalSupply.sub(amount); // Beosin (Chengdu LianAn) // Update the total token supply.
        emit Transfer(account, address(0), amount); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
    }

    function _approve(address owner, address spender, uint256 amount) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'owner'.
        require(spender != address(0), "ERC20: approve to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'spender'.

        _allowances[owner][spender] = amount; // Beosin (Chengdu LianAn) // The allowance which 'owner' allowed to 'spender' is set to 'amount'.
        emit Approval(owner, spender, amount); // Beosin (Chengdu LianAn) // Trigger the event
```

'Approval'.

```
    }

    /**
     * @dev Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */
    function  setupDecimals(uint8 decimals  ) internal {
        _decimals = decimals_;
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when ‘from’ and ‘to’ are both non-zero, ‘amount’ of ‘‘from’’'s tokens
     * will be to transferred to ‘to’.
     * - when ‘from’ is zero, ‘amount’ tokens will be minted for ‘to’.
     * - when ‘to’ is zero, ‘amount’ of ‘‘from’’'s tokens will be burned.
     * - ‘from’ and ‘to’ are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
    uint256[44] private __gap;
}


abstract contract ERC20BurnableUpgradeable is Initializable, ContextUpgradeable, ERC20Upgradeable {
    function __ERC20Burnable_init() internal initializer {
        __Context_init_unchained();
        __ERC20Burnable_init_unchained();
    }

    function __ERC20Burnable_init_unchained() internal initializer {
    }
    using SafeMathUpgradeable for uint256; // Beosin (Chengdu LianAn) // Use the
SafeMathUpgradeable library for mathematical operation. Avoid integer overflow/underflow.

    /**
     * @dev Destroys ‘amount’ tokens from the caller.
     *
     * See {ERC20-_burn}.
```

```
     */
    function burn(uint256 amount) public virtual {
        _burn(_msgSender(), amount); // Beosin (Chengdu LianAn) // Call the internal function '_burn' to burn tokens.
    }

    /**
     * @dev Destroys ‘amount’ tokens from ‘account’, deducting from the caller's
     * allowance.
     *
     * See {ERC20- burn} and {ERC20-allowance}.
     *
     * Requirements:
     *
     * - the caller must have allowance for ‘‘accounts’’s tokens of at least
     * ‘amount’.
     */
    function burnFrom(address account, uint256 amount) public virtual {
        uint256 decreasedAllowance = allowance(account, msgSender()).sub(amount, "ERC20: burn amount exceeds allowance");

        _approve(account, _msgSender(), decreasedAllowance); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to decrease the allowance between two addresses.
        _burn(account, amount); // Beosin (Chengdu LianAn) // Call the internal function '_burn' to burn tokens.
    }
    uint256[50] private __gap;
}



contract ZCX is Initializable, ERC20BurnableUpgradeable {

    function initialize(string memory name, string memory symbol) public virtual initializer {
        __ERC20_init(name, symbol); // Beosin (Chengdu LianAn) // Initialize the token name and symbol.
        _mint(address(0xB406dAaD0B8c447E4566666F3C3986A399f75eae), 1000000000 * (10 ** uint256(decimals()))); // Beosin (Chengdu LianAn) // Call the internal function '_mint' to mint tokens to the specified address.
    }

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



Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com