Smart contract security audit report

Audit Number：202005281143

Smart Contract Name：

ClinTex (CLI)

Smart Contract Address Link：

https://github.com/ClinTex/CLXTokenSale/tree/develop

Commit Hash:

3677da08f4e330d9ac0b76b032d69fe15696f57d

Audit Contracts Name:

No.	Path
1	contracts/CLIToken.sol
2	contracts/LockupContract.sol
3	contracts/managment/Management.sol
4	contracts/managment/Managed.sol
5	contracts/managment/Constants.sol

Start Date：2020.05.08

Completion Date：2020.05.28

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	-	Pass
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 CLI, including Coding Standards, Security, and Business Logic. **CLI 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	ClinTex
Token symbol	CLI
decimals	18
totalSupply	200 million (mintable with Cap is 200 million, burnable)
Token type	ERC20

Table 1 – Basic Token Information

2、Token Vesting Information

This contract has token vesting function, the address with lock permission can call the functions *allocationLog* and *setManuallyLockedForAddress* to set token vesting information.

Audited Source Code with Comments:

```
// Beosin (Chengdu LianAn) // File: contracts/CLIToken.sol
pragma solidity 0.5.17;
// Beosin (Chengdu LianAn) // Import the contract modules which required by this contract.
import "@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "./managment/Managed.sol";
import "./LockupContract.sol";
import "./CLIAllocator.sol";


contract CLIToken is ERC20, ERC20Detailed, Managed {
    // Beosin (Chengdu LianAn) // Modifier, require that the caller of the modified function must have sufficient token balance that can do transfer related operations.
    modifier requireUnlockedBalance(
        address _address,
        uint256 _value,
        uint256 _time,
        uint256 _holderBalance
    ) {

        require(
            LockupContract(
                management.contractRegistry(CONTRACT_LOCKUP)
            ).isTransferAllowed( // Beosin (Chengdu LianAn) // Check the token transfer status of the specified
address.
                _address,
                _value,
                _time,
                _holderBalance
            ),
            ERROR_NOT_AVAILABLE
        );
        _;
    }
    // Beosin (Chengdu LianAn) // Constructor, initialize the token information, and set Managed contract
instance.
    constructor(
        address _management
    )
        public
        ERC20Detailed("ClinTex", "CLI", 18) // Beosin (Chengdu LianAn) // Execute the constructor of contract
'ERC20Detailed' to initialize the basic token information.
        Managed(_management)
    {
    }
    // Beosin (Chengdu LianAn) // The 'mint' function is used to mint tokens to the specified address.
    function mint(
        address _account,
        uint256 _amount
    )
        public
        requirePermission(CAN_MINT_TOKENS) // Beosin (Chengdu LianAn) // Require that the caller must
have permission of minting tokens.
        canCallOnlyRegisteredContract(CONTRACT_ALLOCATOR) // Beosin (Chengdu LianAn) // Require that
the caller must be the contract 'CLIAllocator'.
        returns (bool)
    {
        require(
            _amount <= CLIAllocator(
                management.contractRegistry(CONTRACT_ALLOCATOR)
            ).tokensAvailable(totalSupply()), // Beosin (Chengdu LianAn) // Call the function 'tokensAvailable' of
contract 'CLIAllocator' to get the available token supply, and restrict that the minting amount cannot
exceed the available token supply.
```

```
            ERROR_WRONG_AMOUNT
        );
        _mint(_account, _amount); // Beosin (Chengdu LianAn) // Call the internal function '_mint' to mint
tokens.
        return true;
    }
    // Beosin (Chengdu LianAn) // The 'transfer' function, function caller sends the specified amount of tokens
to a specified address.
    function transfer(
        address _to,
        uint256 _tokens
    )
        public
        requireUnlockedBalance( // Beosin (Chengdu LianAn) // Require that the caller must have sufficient
token balance that can transfer.
            msg.sender,
            _tokens,
            block.timestamp,
            balanceOf(msg.sender)
        )
        returns (bool)
    {
        super.transfer(_to, _tokens); // Beosin (Chengdu LianAn) // Call function 'transfer' of the parent contract
to transfer tokens.

        return true;
    }
    // Beosin (Chengdu LianAn) // The 'transferFrom' function, 'msg.sender' as a delegate of '_holder'
transfers the specified amount of tokens to a specified address.
    function transferFrom(
        address _holder,
        address _to,
        uint256 _tokens
    )
        public
        requireUnlockedBalance( // Beosin (Chengdu LianAn) // Require that the '_holder' must have sufficient
token balance that can transfer.
            _holder,
            _tokens,
            block.timestamp,
            balanceOf(_holder)
        )
        returns (bool)
    {
        super.transferFrom(_holder, _to, _tokens); // Beosin (Chengdu LianAn) // Call function 'transferFrom' of
the parent contract to transfer tokens.

        return true;
    }
    // Beosin (Chengdu LianAn) // The 'burn' function, 'msg.sender' burns the specific amount of tokens of itself.
    function burn(uint256 value)
        public
        requirePermission(CAN_BURN_TOKENS) // Beosin (Chengdu LianAn) // Require that the caller must have permission of destroying tokens.
        requireUnlockedBalance( // Beosin (Chengdu LianAn) // Require that the caller must have sufficient token balance that can be burn.
            msg.sender,
            value,
            block.timestamp,
            balanceOf(msg.sender)
        )
    {
        require(balanceOf(msg.sender) >= value, ERROR_WRONG_AMOUNT); // Beosin (Chengdu LianAn) // Balance check, require that the destroying value cannot exceed the token balance of 'msg.sender'.
        super._burn(msg.sender, value); // Beosin (Chengdu LianAn) // Call the internal function '_burn' of the parent contract to destroy tokens.
    }
}




// Beosin (Chengdu LianAn) // File: contracts/LockupContract.sol
pragma solidity 0.5.17;
// Beosin (Chengdu LianAn) // Import the contract modules which required by this contract.
import "@openzeppelin/contracts/math/SafeMath.sol";
import "./managment/Managed.sol";


contract LockupContract is Managed {
    using SafeMath for uint256;

    uint256 public constant PERCENT_ABS_MAX = 100;
    bool public isPostponedStart; // Beosin (Chengdu LianAn) // Declare the variable 'isPostponedStart' for storing whether the specified lock start timestamp is set.
    uint256 public postponedStartDate; // Beosin (Chengdu LianAn) // Declare the variable 'postponedStartDate' for storing the specified lock start timestamp.

    mapping(address => uint256[]) public lockedAllocationData; // Beosin (Chengdu LianAn) // Declare the mapping variable 'lockedAllocationData' for storing the information about the lock of the specified address.

    mapping(address => uint256) public manuallyLockedBalances; // Beosin (Chengdu LianAn) // Declare the mapping variable 'manuallyLockedBalances' for storing the manually locked token balance of the soecified address.

    event Lock(address holderAddress, uint256 amount); // Beosin (Chengdu LianAn) // Declare the event 'Lock'.
    // Beosin (Chengdu LianAn) // Constructor, initialize the 'isPostponedStart' status.
    constructor(address _management) public Managed(_management) {
        isPostponedStart = true;
    }
    // Beosin (Chengdu LianAn) // The 'isTransferAllowed' function, check whether the specified address is
allowed to transfer.
    function isTransferAllowed(
        address _address,
        uint256 _value,
        uint256 _time,
        uint256 _holderBalance
    )
    external
    view
    returns (bool)
    {
        uint256 unlockedBalance = getUnlockedBalance(
            _address,
            _time,
            _holderBalance
        ); // Beosin (Chengdu LianAn) // Get the unlocked token balance of the specified address.
        if (unlockedBalance >= _value) { // Beosin (Chengdu LianAn) // If the unlocked token balance is not less
than the amount of transfer value.
            return true;
        }
        return false;
    }
    // Beosin (Chengdu LianAn) // The 'allocationLog' function, set the token lock allocation.
    function allocationLog(
        address _address,
        uint256 _amount,
        uint256 _startingAt,
        uint256 _lockPeriodInSeconds,
        uint256 _initialUnlockInPercent,
        uint256 _releasePeriodInSeconds
    )
        public
        requirePermission(CAN_LOCK_TOKENS) // Beosin (Chengdu LianAn) // Require that the caller must
have permission of setting lock.
    {
        // Beosin (Chengdu LianAn) // Store lock information.
        lockedAllocationData[_address].push(_startingAt);
        if (_initialUnlockInPercent > 0) { // Beosin (Chengdu LianAn) // If the initial release token percentage is
greater than 0.
            _amount = _amount.mul(uint256(PERCENT_ABS_MAX)
                .sub(_initialUnlockInPercent)).div(PERCENT_ABS_MAX);
        }
        lockedAllocationData[_address].push(_amount);
        lockedAllocationData[_address].push(_lockPeriodInSeconds);
        lockedAllocationData[_address].push(_releasePeriodInSeconds);
        emit Lock(_address, _amount); // Beosin (Chengdu LianAn) // Trigger the event 'Lock'.
    }
    // Beosin (Chengdu LianAn) // The 'getUnlockedBalance' function, get the unlocked token balance of the
specified address.
    function getUnlockedBalance(
        address _address,
        uint256 _time,
        uint256 _holderBalance
    )
        public
        view
        returns (uint256)
    {
        uint256 blockedAmount = manuallyLockedBalances[_address]; // Beosin (Chengdu LianAn) // Declare the
local variable 'blockedAmount' for storing the amount of tokens locked at the specified address. It is
'manuallyLockedBalances' as default.
        // Beosin (Chengdu LianAn) // If the specified address '_address' have no locked allocation data, return
the difference between its token balance and the locked tokens.
        if (lockedAllocationData[_address].length == 0) {
            return _holderBalance.sub(blockedAmount);
        }
        uint256[] memory  addressLockupData = lockedAllocationData[_address]; // Beosin (Chengdu LianAn) //
Declare the array 'addressLockupData' for storing lock information.
        // Beosin (Chengdu LianAn) // Traverse lock information.
        for (uint256 i = 0; i < addressLockupData.length / 4; i++) {
            uint256 lockedAt = addressLockupData[i.mul(4)]; // Beosin (Chengdu LianAn) // Get the start time of
the lock of number i.
            uint256 lockedBalance = addressLockupData[i.mul(4).add(1)]; // Beosin (Chengdu LianAn) // Get the
locked number of tokens in the lock of number i.
            uint256 lockPeriodInSeconds = addressLockupData[i.mul(4).add(2)]; // Beosin (Chengdu LianAn) // Get
the total time of the lock of number i.
            uint256 _releasePeriodInSeconds = addressLockupData[
                i.mul(4).add(3)
            ]; // Beosin (Chengdu LianAn) // Get the each release period of the lock of number i.
            // Beosin (Chengdu LianAn) // If the lock start time is 0 and the specified lock start timestamp is set.
            if (lockedAt == 0 && true == isPostponedStart) {
                if (postponedStartDate == 0) {
                    blockedAmount = blockedAmount.add(lockedBalance); // Beosin (Chengdu LianAn) // If the
specified lock start timestamp is 0, the locked amount is updated to the all tokens in the lock of number i.
Note: The operation logic is the original intention of the project party, ignoring the impact.
                    continue;
                }
                lockedAt = postponedStartDate; // Beosin (Chengdu LianAn) // Set the start time of the lock of
number i is 'postponedStartDate'.
            }
            if (lockedAt > _time) {
                blockedAmount = blockedAmount.add(lockedBalance); // Beosin (Chengdu LianAn) // If the '_time' doesn't reach the lock start time, the locked amount is updated to the all tokens in the lock of number i. Note: The operation logic is the original intention of the project party, ignoring the impact.
                continue;
            }
            // Beosin (Chengdu LianAn) // If the '_time' doesn't reach the lock end time.
            if (lockedAt.add(lockPeriodInSeconds) > _time) {
                if (lockedBalance == 0) { // Beosin (Chengdu LianAn) // If the number of locked tokens is 0.
                    blockedAmount = _holderBalance; // Beosin (Chengdu LianAn) // The number of tokens locked at the specified address is '_holderBalance'.
                    break;
                } else { // Beosin (Chengdu LianAn) // If the number of locked tokens is not 0.
                    uint256 tokensUnlocked;
                    if (_releasePeriodInSeconds > 0) { // Beosin (Chengdu LianAn) // If the release period is greater than 0, calculate the number of tokens that can be released of this address at specified time '_time'.
                        uint256 duration = (_time.sub(lockedAt))
                            .div(_releasePeriodInSeconds); // Beosin (Chengdu LianAn) // Declare the local variable 'duration' for getting the passed duration number from 'lockedAt' to the specified time '_time'.
                        tokensUnlocked = lockedBalance.mul(duration)
                            .mul(_releasePeriodInSeconds)
                            .div(lockPeriodInSeconds);
                    }
                    blockedAmount = blockedAmount
                        .add(lockedBalance)
                        .sub(tokensUnlocked); // Beosin (Chengdu LianAn) // Update the locked amount.
                }
            }
        }

        return _holderBalance.sub(blockedAmount); // Beosin (Chengdu LianAn) // Get the total amount of unlocked tokens.
    }
    // Beosin (Chengdu LianAn) // The 'setManuallyLockedForAddress' function, set the number of manually locked tokens at the specified address.
    function setManuallyLockedForAddress (
        address _holder,
        uint256 _balance
    )
        public
        requirePermission(CAN_LOCK_TOKENS) // Beosin (Chengdu LianAn) // Require that the caller must have permission of setting lock.
    {
        manuallyLockedBalances[_holder] = _balance; // Beosin (Chengdu LianAn) // Set the manual lock token number of specified address '_holder' to '_balance'.
    }
    // Beosin (Chengdu LianAn) // The 'setPostponedStartDate' function, set the specified lock start timestamp.
    function setPostponedStartDate(uint256 _postponedStartDate)
```

```
        public
        requirePermission(CAN_LOCK_TOKENS) // Beosin (Chengdu LianAn) // Require that the caller must have permission of setting lock.
    {
        postponedStartDate = _postponedStartDate; // Beosin (Chengdu LianAn) // Set the specified lock start timestamp to '_postponedStartDate'.

    }
}




// Beosin (Chengdu LianAn) // File: contracts/managment/Management.sol
pragma solidity 0.5.17;
// Beosin (Chengdu LianAn) // Import the contract modules which required by this contract.
import "@openzeppelin/contracts/ownership/Ownable.sol";
import "./Constants.sol";


contract Management is Ownable, Constants {

    // Contract Registry
    mapping (uint256 => address payable) public contractRegistry; // Beosin (Chengdu LianAn) // Declare the mapping variable 'contractRegistry' for storing the register contract address corresponding to the key.

    // Permissions
    mapping (address => mapping(uint256 => bool)) public permissions; // Beosin (Chengdu LianAn) // Declare the mapping variable 'permissions' for storing the permissions corresponding to the specified address.

    event PermissionsSet(
        address subject,
        uint256 permission,
        bool value
    ); // Beosin (Chengdu LianAn) // Declare the event 'PermissionsSet'.

    event ContractRegistered(
        uint256 key,
        address source,
        address target
    ); // Beosin (Chengdu LianAn) // Declare the event 'ContractRegistered'.
    // Beosin (Chengdu LianAn) // The 'setPermission' function, the contract owner sets the permission of the specified address.
    function setPermission(
        address _address,
        uint256 _permission,
        bool _value
    )
        public
        onlyOwner
    {
        permissions[_address][_permission] = _value;
        emit PermissionsSet(_address, _permission, _value); // Beosin (Chengdu LianAn) // Trigger the event
'PermissionsSet'.
    }
    // Beosin (Chengdu LianAn) // The 'registerContract' function, contract owner registers contract.
    function registerContract(
        uint256 _key,
        address payable _target
    )
        public
        onlyOwner
    {
        contractRegistry[_key] = _target; // Beosin (Chengdu LianAn) // Set the contract address corresponding to
the '_key'.
        emit ContractRegistered(_key, address(0), _target); // Beosin (Chengdu LianAn) // Trigger the event
'ContractRegistered'.
    }
    // Beosin (Chengdu LianAn) // The 'setWhitelisted' function, the caller add '_address' to the whitelist.
    function setWhitelisted(
        address _address,
        bool _value
    )
        public
    {
        require(
            permissions[msg.sender][CAN_SET_WHITELISTED] == true,
            ERROR_ACCESS_DENIED
        ); // Beosin (Chengdu LianAn) // Require that the caller must have permission of setting whitelist.

        permissions[_address][WHITELISTED] = _value; // Beosin (Chengdu LianAn) // Set the whitelist status of
the specified address '_address' to '_value'.

        emit PermissionsSet(_address, WHITELISTED, _value); // Beosin (Chengdu LianAn) // Trigger the event
'PermissionsSet'.
    }

}




// Beosin (Chengdu LianAn) // File: contracts/managment/Managed.sol
pragma solidity 0.5.17;
// Beosin (Chengdu LianAn) // Import the contract modules which required by this contract.
import "@openzeppelin/contracts/ownership/Ownable.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "./Constants.sol";
import "./Management.sol";


contract Managed is Ownable, Constants {

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

    Management public management;
    // Beosin (Chengdu LianAn) // Modifier, require that the caller must have corresponding permission.
    modifier requirePermission(uint256 _permissionBit) {
        require(
            hasPermission(msg.sender, _permissionBit),
            ERROR_ACCESS_DENIED
        );
        _;
    }
    // Beosin (Chengdu LianAn) // Modifier, require that The caller must be registered contract.
    modifier canCallOnlyRegisteredContract(uint256 _key) {
        require(
            msg.sender == management.contractRegistry(_key),
            ERROR_ACCESS_DENIED
        );
        _;
    }
    // Beosin (Chengdu LianAn) // Modifier, require that the registered contract address corresponding to
'_key' is not zero address.
    modifier requireContractExistsInRegistry(uint256 _key) {
        require(
            management.contractRegistry(_key) != address(0),
            ERROR_NO_CONTRACT
        );
        _;
    }
    // Beosin (Chengdu LianAn) // Constructor, initialize the external Management contract instance.
    constructor(address _managementAddress) public {
        management = Management(_managementAddress);
    }
    // Beosin (Chengdu LianAn) // The 'setManagementContract' function, the contract owner call this
function to set the management contract.
    function setManagementContract(address _management) public onlyOwner {
        require(address(0) != _management, ERROR_NO_CONTRACT); // Beosin (Chengdu LianAn) // The non-
zero address check for '_management'.

        management = Management(_management); // Beosin (Chengdu LianAn) // Set Management contract
instance.
    }
    // Beosin (Chengdu LianAn) // The 'hasPermission' function, query whether the specified address
```

```
'_subject' has the corresponding permission.
    function hasPermission(address _subject, uint256 _permissionBit)
        internal
        view
        returns (bool)
    {
        return management.permissions(_subject, _permissionBit);
    }

}




// Beosin (Chengdu LianAn) // File: contracts/managment/Constants.sol
pragma solidity 0.5.17;

// Beosin (Chengdu LianAn) // Declare the following constants required by the main contract.
contract Constants {
    // Permissions bit constants
    uint256 public constant CAN_MINT_TOKENS = 0;
    uint256 public constant CAN_BURN_TOKENS = 1;
    uint256 public constant CAN_UPDATE_STATE = 2;
    uint256 public constant CAN_LOCK_TOKENS = 3;
    uint256 public constant CAN_UPDATE_PRICE = 4;
    uint256 public constant CAN_INTERACT_WITH_ALLOCATOR = 5;
    uint256 public constant CAN_SET_ALLOCATOR_MAX_SUPPLY = 6;
    uint256 public constant CAN_PAUSE_TOKENS = 7;
    uint256 public constant ECLIUDED_ADDRESSES = 8;
    uint256 public constant WHITELISTED = 9;
    uint256 public constant SIGNERS = 10;
    uint256 public constant EXTERNAL_CONTRIBUTORS = 11;
    uint256 public constant CAN_SEE_BALANCE = 12;
    uint256 public constant CAN_CANCEL_TRANSACTION = 13;
    uint256 public constant CAN_ALLOCATE_REFERRAL_TOKENS = 14;
    uint256 public constant CAN_SET_REFERRAL_MAX_SUPPLY = 15;
    uint256 public constant MANUAL_TOKENS_ALLOCATION = 16;
    uint256 public constant CAN_SET_WHITELISTED = 17;

    // Contract Registry keys
    uint256 public constant CONTRACT_TOKEN = 1;
    uint256 public constant CONTRACT_PRICING = 2;
    uint256 public constant CONTRACT_CROWDSALE = 3;
    uint256 public constant CONTRACT_ALLOCATOR = 4;
    uint256 public constant CONTRACT_AGENT = 5;
    uint256 public constant CONTRACT_FORWARDER = 6;
    uint256 public constant CONTRACT_REFERRAL = 7;
    uint256 public constant CONTRACT_STATS = 8;
    uint256 public constant CONTRACT_LOCKUP = 9;
    uint256 public constant YEAR_IN_SECONDS = 31556952;
    uint256 public constant SIX_MONTHS =  15778476;
    uint256 public constant MONTH_IN_SECONDS = 2629746;

    string public constant ERROR_ACCESS_DENIED = "ERROR_ACCESS_DENIED";
    string public constant ERROR_WRONG_AMOUNT = "ERROR_WRONG_AMOUNT";
    string public constant ERROR_NO_CONTRACT = "ERROR_NO_CONTRACT";
    string public constant ERROR_NOT_AVAILABLE = "ERROR_NOT_AVAILABLE";
}
```

Beosin

成都链安

BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/LianAnTech