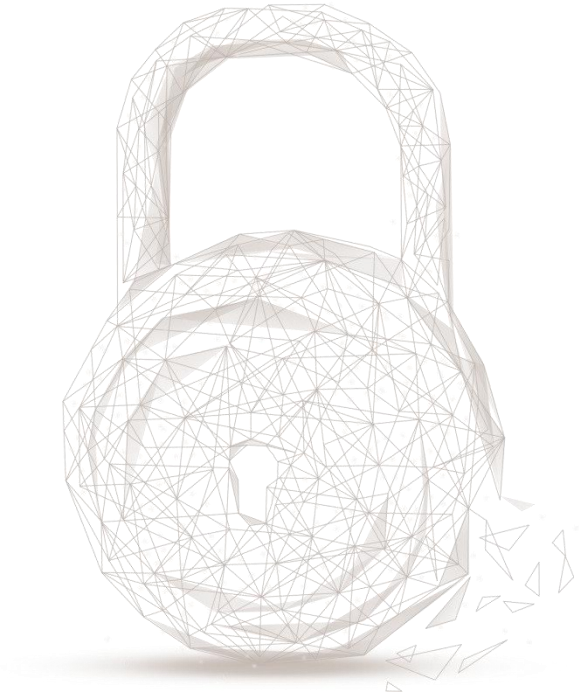
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202106171658

Smart Contract Name：

DVGToken (DVG)

Commit hash：

3652d4c70f766150b1be46ad07725d5363f1950a

Smart Contract Address Link：

https://github.com/daoventures/DVG/commit/3652d4c70f766150b1be46ad07725d5363f1950a

Start Date：2021.06.02

Completion Date：2021.06.17

Overall Result：Pass (Distinction)

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

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	BEP-20 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

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 DVG, including Coding Standards, Security, and Business Logic. **DVG 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	DVGToken
Token symbol	DVG
decimals	18
totalSupply	Initial is issue when deploy, mint without cap
Token type	BEP-20

Table 1 – Basic Token Information

2. Token Vesting Information

N/A

Audited Source Code with Comments

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

pragma solidity 0.7.6;
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "./libraries/ERC20Permit.sol";
/// DVG token
contract DVGToken is ERC20Permit("DVGToken", "DVG"), Ownable {
    using Address for address;
    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Using the SafeMath library for Mathematical operation. Avoid integer overflow.
    /// A checkpoint for marking number of votes from a given block
    struct Checkpoint {
        uint32 fromBlock;
        uint256 votes;
    }
    /// We need mint some DVG in advance
    uint256 public dvgInAdvance; // Beosin (Chengdu LianAn) // Declare the variable 'dvgInAdvance' to store the initial issue amount.
    /// The address of Treasury wallet
    address public treasuryWalletAddr; // Beosin (Chengdu LianAn) // Declare the variable 'treasuryWalletAddr' to store the initial token receiving address.
    /// The EIP-712 typehash for the contract's domain
    bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");
```

```
    /// The EIP-712 typehash for the delegation struct used by the contract
    bytes32 public constant DELEGATION_TYPEHASH = keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)");
    // A record of each accounts delegate
    mapping (address => address) internal _delegates; // Beosin (Chengdu LianAn) // Declare the mapping variable '_delegates' for storing the delegator of corresponding address.
    /// A record of votes checkpoints for each account, by index
    mapping (address => mapping (uint32 => Checkpoint)) public checkpoints; // Beosin (Chengdu LianAn) // Declare the mapping variable 'checkpoints' for storing the specified snapshot information for the corresponding address.
    /// The number of checkpoints for each account
    mapping (address => uint32) public numCheckpoints; // Beosin (Chengdu LianAn) // Declare the mapping variable 'numCheckpoints' for storing the amount of snapshot for the corresponding address.
    /// A record of states for signing / validating signatures
    /// nonce has been declared in ERC20Permit.
    // mapping (address => uint) public override nonces;
    // set of minters, can be this bridge or other bridges
    mapping(address => bool) public isMinter;
    address[] public minters;
    /// An event thats emitted when an account changes its delegate
    event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate); // Beosin (Chengdu LianAn) // Declare the event 'DelegateChanged'.
    /// An event thats emitted when a delegate account's vote balance changes
    event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance); // Beosin (Chengdu LianAn) // Declare the event 'DelegateVotesChanged'.
    /// @notice We need mint some DVGs in advance
    constructor(address _treasuryWalletAddr, uint256 _dvgInAdvance) {
        require(!_treasuryWalletAddr.isContract(), "The Treasury wallet address should not be the smart contract address");
        treasuryWalletAddr = _treasuryWalletAddr;
        dvgInAdvance = _dvgInAdvance;
        mint(treasuryWalletAddr, dvgInAdvance); // Beosin (Chengdu LianAn) // Send the initial token to the specified address.
    }
    modifier onlyMinter() {
        require((owner() == msg.sender) || (isMinter[msg.sender] == true), "Restricted Access!");
        _;
    }
    // Beosin (Chengdu LianAn) // Gives the specified address minter permission, which can only be called by the owner.
    function addMinter(address _minter) external onlyOwner {
        require(isMinter[_minter] == false, "The address already has minter role");
        isMinter[_minter] = true;
        minters.push(_minter);
    }
    // Beosin (Chengdu LianAn) // Remove the specified address minter permission, which can only be called by the owner.
    function removeMinter(address _minter) external onlyOwner {
```

```
        require(isMinter[_minter], "The address does not have minter role");
        isMinter[_minter] = false;
        for (uint i = 0; i < minters.length; i ++){
            if (minters[i] == _minter) {
                minters[i] = minters[minters.length-1];
                minters.pop();
                break;
            }
        }
    }
    // Beosin (Chengdu LianAn) // Returns the addresses of all minter.
    function getAllMinters() external view returns (address[] memory) {
        return minters;
    }
    /// @notice Creates `_amount` token to `_to`. Must only be called by the owner
    function mint(address _to, uint256 _amount) public onlyMinter {
        _mint(_to, _amount); // Beosin (Chengdu LianAn) // Mint a specified number of tokens to a specified address.
        _moveDelegates(address(0), _delegates[_to], _amount); // Beosin (Chengdu LianAn) // Call internal function '_moveDelegates' to update voting information and snapshots.
    }
    /// @notice Destroys `_amount` tokens from `_from`. Must only be called by the owner
    function burn(address _from, uint256 _amount) public onlyMinter {
        _burn(_from, _amount); // Beosin (Chengdu LianAn) // Call the internal function '_burn' burn '_from' for the specified number of tokens.
        if (_from != _msgSender()) {
            _approve(_from, _msgSender(), allowance(_from, _msgSender()).sub(_amount, "ERC20: burn amount exceeds allowance")); // Beosin (Chengdu LianAn) // If '_from' is not the caller, then reduce its allowance to the caller.
        }
        _moveDelegates(_delegates[_from], address(0), _amount); // Beosin (Chengdu LianAn) // Call internal function '_moveDelegates' to update voting information and snapshots.
    }
    /**
     * @notice Get `delegatee` of `delegator`
     * @param delegator The address to get delegatee for
     */
    function delegates(address delegator)
        external
        view
        returns (address)
    {
        return _delegates[delegator];
    }
   /**
    * @notice Delegate votes from `msg.sender` to `delegatee`
    * @param delegatee The address to delegate votes to
    */
```

```
    // Beosin (Chengdu LianAn) // The 'delegate' function, msg.sender vote for the specified address.
    function delegate(address delegatee) external {
        return _delegate(msg.sender, delegatee); // Beosin (Chengdu LianAn) // Call internal function '_delegate' to delegate.
    }
    /**
     * @notice Delegates votes from signatory to `delegatee`
     * @param delegatee The address to delegate votes to
     * @param nonce The contract state required to match the signature
     * @param expiry The time at which to expire the signature
     * @param v The recovery byte of the signature
     * @param r Half of the ECDSA signature pair
     * @param s Half of the ECDSA signature pair
     */
    // Beosin (Chengdu LianAn) // The 'delegateBySig' function, msg.sender vote and sign for the specified address.
    function delegateBySig(
        address delegatee,
        uint nonce,
        uint expiry,
        uint8 v,
        bytes32 r,
        bytes32 s
    )
        external
    {
        bytes32 domainSeparator = keccak256(
            abi.encode(
                DOMAIN_TYPEHASH,
                keccak256(bytes(name())),
                getChainId(),
                address(this)
            )
        );
        bytes32 structHash = keccak256(
            abi.encode(
                DELEGATION_TYPEHASH,
                delegatee,
                nonce,
                expiry
            )
        );
        bytes32 digest = keccak256(
            abi.encodePacked(
                "\x19\x01",
                domainSeparator,
                structHash
            )
```

```
            );
            address signatory = ecrecover(digest, v, r, s);
            require(signatory != address(0), "delegateBySig: invalid signature"); // Beosin (Chengdu LianAn) //
The non-zero address check for 'signatory'.
            require(nonce == nonces[signatory]++, "delegateBySig: invalid nonce"); // Beosin (Chengdu LianAn)
// 'nonce' is required to be equal to the specified value.
            require(block.timestamp <= expiry, "delegateBySig: signature expired"); // Beosin (Chengdu LianAn)
// It is required that the current time is not greater than 'expiry'.
            return _delegate(signatory, delegatee); // Beosin (Chengdu LianAn) // Call internal function
'_delegate' to vote.
        }
        /**
         * @notice Gets the current votes balance for `account`
         * @param account The address to get votes balance
         * @return The number of current votes for `account`
         */
        // Beosin (Chengdu LianAn) // Get the number of votes held at the specified address from the
snapshot.
        function getCurrentVotes(address account)
            external
            view
            returns (uint256)
        {
            uint32 nCheckpoints = numCheckpoints[account];
            return nCheckpoints > 0 ? checkpoints[account][nCheckpoints - 1].votes : 0; // Beosin (Chengdu
LianAn) // Returns 0 if the account address does not have a snapshot.
        }
        /**
         * @notice Determine the prior number of votes for an account as of a block number
         * @dev Block number must be a finalized block or else this function will revert to prevent misinformation.
         * @param account The address of the account to check
         * @param blockNumber The block number to get the vote balance at
         * @return The number of votes the account had as of the given block
         */
        // Beosin (Chengdu LianAn) // Get the number of votes held before a block at the specified address
from the snapshot.
        function getPriorVotes(address account, uint blockNumber)
            external
            view
            returns (uint256)
        {
            require(blockNumber < block.number, "getPriorVotes: not yet determined"); // Beosin (Chengdu
LianAn) // 'blockNumber' is required to be less than the current block time.
            uint32 nCheckpoints = numCheckpoints[account];
            // Beosin (Chengdu LianAn) // Returns 0 if the account address does not have a snapshot.
            if (nCheckpoints == 0) {
                return 0;
            }
```

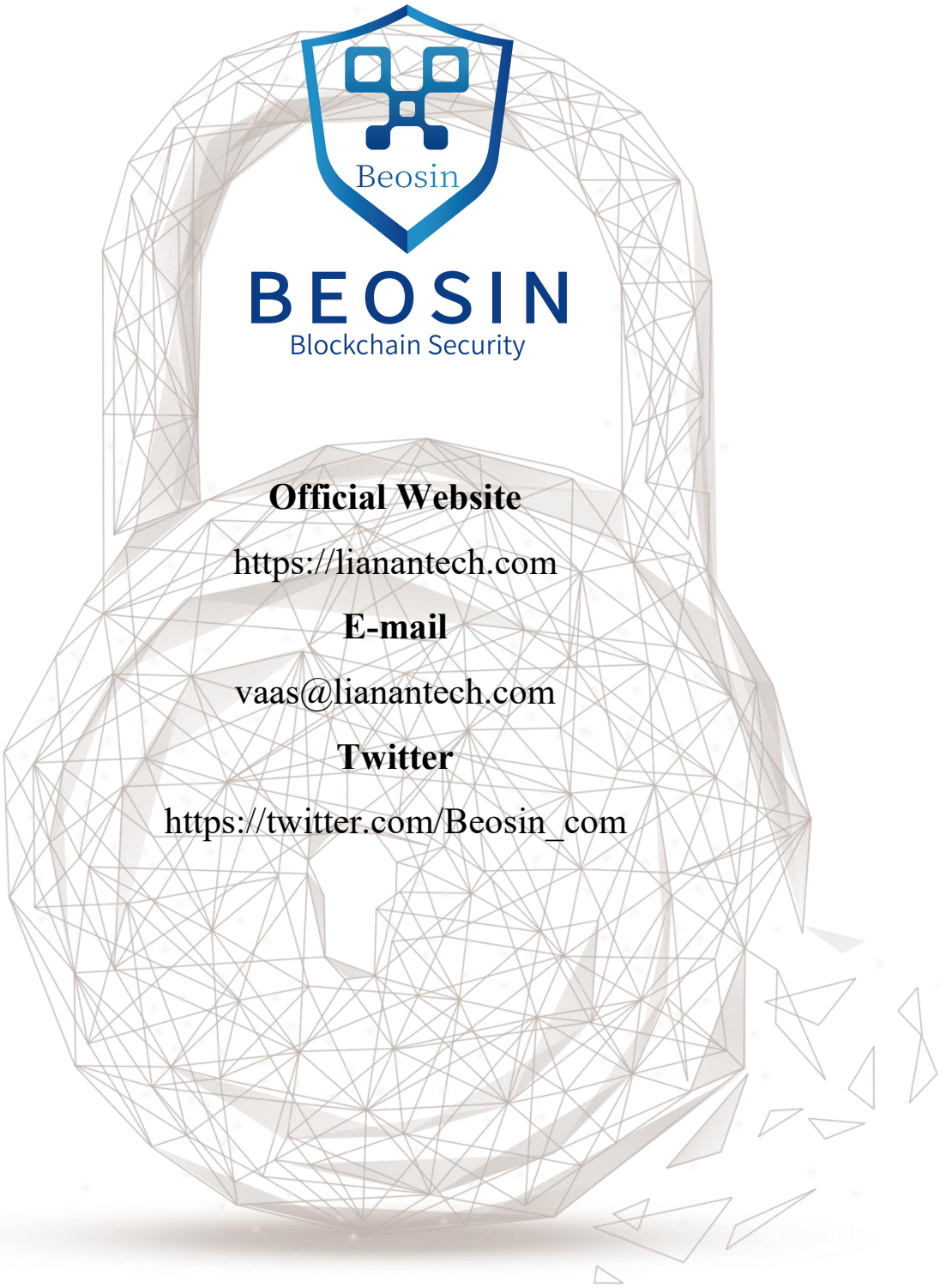
```
        // First check most recent balance // Beosin (Chengdu LianAn) // If the latest snapshot time is not greater than blockNumber, return the number of votes saved for the latest snapshot.
        if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
            return checkpoints[account][nCheckpoints - 1].votes;
        }
        // Next check implicit zero balance // Beosin (Chengdu LianAn) // Return 0 if the first snapshot time is greater than blockNumber.
        if (checkpoints[account][0].fromBlock > blockNumber) {
            return 0;
        }
        // Beosin (Chengdu LianAn) // Use dichotomy to find snapshots that meet the requirements.
        uint32 lower = 0;
        uint32 upper = nCheckpoints - 1;
        while (upper > lower) {
            uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
            Checkpoint memory cp = checkpoints[account][center];
            if (cp.fromBlock == blockNumber) {
                return cp.votes;
            } else if (cp.fromBlock < blockNumber) {
                lower = center;
            } else {
                upper = center - 1;
            }
        }
        return checkpoints[account][lower].votes;
    }
    // Beosin (Chengdu LianAn) // Internal function, implement the 'delegate' operation.
    function _delegate(address delegator, address delegatee)
        internal
    {
        address currentDelegate = _delegates[delegator]; // Beosin (Chengdu LianAn) // Get the current voting address of 'delegator'.
        uint256 delegatorBalance = balanceOf(delegator); // balance of underlying DVGs (not scaled); // Beosin (Chengdu LianAn) // Get the token balance of 'delegator'.
        _delegates[delegator] = delegatee; // Beosin (Chengdu LianAn) // Set the new voting address of 'delegator' to 'delegatee'.
        emit DelegateChanged(delegator, currentDelegate, delegatee); // Beosin (Chengdu LianAn) // Trigger the event 'DelegateChanged'.
        _moveDelegates(currentDelegate, delegatee, delegatorBalance); // Beosin (Chengdu LianAn) // Call internal function '_moveDelegates' to update voting information and snapshots.
    }

  /**
     * @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(ERC20) returns (bool) {
        address sender = _msgSender();
        _transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Call internal function '_transfer' to transfer tokens.
        _moveDelegates(_delegates[sender], _delegates[recipient], amount); // Beosin (Chengdu LianAn) // Call internal function '_moveDelegates' to update voting information and snapshots.
        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(ERC20) returns (bool) {
        address spender = _msgSender();
        uint256 spenderAllowance = allowance(sender, spender);
        if (spenderAllowance != type(uint256).max) {
            _approve(sender, spender, spenderAllowance.sub(amount, "ERC20: transfer amount exceeds allowance")); // Beosin (Chengdu LianAn) // If the sender's allowance for the spender is not the maximum allowance, the allowance is reduced.
        }
        _transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Call internal function '_transfer' to transfer tokens.
        _moveDelegates(_delegates[sender], _delegates[recipient], amount); // Beosin (Chengdu LianAn) // Call internal function '_moveDelegates' to update voting information and snapshots.
        return true;
    }
    function transferWithPermit(address from, address to, uint256 amount, uint256 deadline, uint8 v, bytes32 r, bytes32 s) public virtual override(ERC20Permit) returns (bool) {
        _checkPermit(from, to, amount, deadline, v, r, s); // Beosin (Chengdu LianAn) // Call internal function '_checkPermit' to get permit.
        _transfer(from, to, amount); // Beosin (Chengdu LianAn) // Call internal function '_transfer' to transfer tokens.
        _moveDelegates(_delegates[from], _delegates[to], amount); // Beosin (Chengdu LianAn) // Call internal function '_moveDelegates' to update voting information and snapshots.
        return true;
    }
```

```
    // Beosin (Chengdu LianAn) // Internal function, called when the user delegate and the balance changes to update the snapshot information.
    function _moveDelegates(address srcRep, address dstRep, uint256 amount) internal {
        if (srcRep != dstRep && amount > 0) {
            if (srcRep != address(0)) {
                // decrease old representative
                uint32 srcRepNum = numCheckpoints[srcRep];
                uint256 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum - 1].votes : 0;
                uint256 srcRepNew;
                if (srcRepOld > amount) {
                    srcRepNew = srcRepOld.sub(amount);
                } else {
                    srcRepNew = 0;
                    amount = srcRepOld;
                } // Beosin (Chengdu LianAn) // Calculate the new value of the 'srcRep' address stored in the snapshot.
                _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew); // Beosin (Chengdu LianAn) // Call internal function '_writeCheckpoint' to update snapshot.
            }
            if (dstRep != address(0)) {
                // increase new representative
                uint32 dstRepNum = numCheckpoints[dstRep];
                uint256 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum - 1].votes : 0;
                uint256 dstRepNew = dstRepOld.add(amount); // Beosin (Chengdu LianAn) // Calculate the new value of the 'dstRep' address stored in the snapshot.
                _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew); // Beosin (Chengdu LianAn) // Call internal function '_writeCheckpoint' to update snapshot.
            }
        }
    }
    function _writeCheckpoint(
        address delegatee,
        uint32 nCheckpoints,
        uint256 oldVotes,
        uint256 newVotes
    )
        internal
    {
        uint32 blockNumber = safe32(block.number, "_writeCheckpoint: block number exceeds 32 bits");
        // Beosin (Chengdu LianAn) // If the number of snapshots is greater than 0 and the latest snapshot is the current time, update the latest snapshot information.
        if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
            checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
        } else {
            checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes); // Beosin (Chengdu LianAn) // Create new snapshot.
            numCheckpoints[delegatee] = nCheckpoints + 1; // Beosin (Chengdu LianAn) // Update amount of snapshots for 'delegatee'.
```

```
            }
            emit DelegateVotesChanged(delegatee, oldVotes, newVotes); // Beosin (Chengdu LianAn) // Trigger the event 'DelegateVotesChanged'.
        }
        // Beosin (Chengdu LianAn) // Bytes length check for uint32 and uint96.
        function safe32(uint n, string memory errorMessage) internal pure returns (uint32) {
            require(n < 2**32, errorMessage);
            return uint32(n);
        }
        function getChainId() internal pure returns (uint) {
            uint256 chainId;
            assembly { chainId := chainid() }
            return chainId;
        }
}
// 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