BlockSAFU

ADVANCE MANUAL
SMART CONTRACT AUDIT

Project: LinkDao

Website: https://linkdao.network/

PASSED

BlockSAFU Score:

82

Contract Address:

0xaF027427DC6d31A3e7e162A710a5Fe27e63E275F

DISCLAMER

BlockSAFU has completed this report to provide a summary of the Smart Contract functions, and any security, dependency, or cybersecurity vulnerabilities. This is often a constrained report on our discoveries based on our investigation and understanding of the current programming versions as of this report's date. To understand the full scope of our analysis, it is vital for you to at the date of this report. To understand the full scope of our analysis, you need to review the complete report. Although we have done our best in conducting our investigation and creating this report, it is vital to note that you should not depend on this report and cannot make any claim against BlockSAFU or its Subsidiaries and Team members on the premise of what has or has not been included in the report. Please remember to conduct your independent examinations before making any investment choices. We do not provide investment advice or in any way claim to determine if the project will be successful or not.

By perusing this report or any portion of it, you concur to the terms of this disclaimer. In the unlikely situation where you do not concur to the terms, you should immediately terminate reading this report, and erase and discard any duplicates of this report downloaded and/or printed by you. This report is given for data purposes as it were and on a non-reliance premise and does not constitute speculation counsel. No one should have any right to depend on the report or its substance, and BlockSAFU and its members (including holding companies, shareholders, backups, representatives, chiefs, officers, and other agents) BlockSAFU and its subsidiaries owe no obligation of care towards you or any other person, nor does BlockSAFU make any guarantee or representation to any individual on the precision or completeness of the report.

ABOUT THE AUDITOR:

BlockSAFU (BSAFU) is an Anti-Scam Token Utility that reviews Smart Contracts and Token information to Identify Rug Pull and Honey Pot scamming activity. BlockSAFUs Development Team consists of several Smart Contract creators, Auditors Developers, and Blockchain experts. BlockSAFU provides solutions, prevents, and hunts down scammers. BSAFU is a utility token with features Audit, KYC, Token Generators, and Bounty Scammers. It will enrich the crypto ecosystem.

OVERVIEW

BlockSAFU was commissioned by LinkDao to complete a Smart Contract audit. The objective of the Audit is to achieve the following:

- Review the Project and experience and Development team
- Ensure that the Smart Contract functions are necessary and operate as intended.
- Identify any vulnerabilities in the Smart Contract code.

DISCLAIMER: This Audit is intended to inform about token Contract Risks, the result does not imply an endorsement or provide financial advice in any way, all investments are made at your own risk. (https://blocksafu.com/)

SMART CONTRACT REVIEW

Token Name	**LinkDao**
Token Symbol	**LKD**
Token Decimal	18
Total Supply	10,000,000 **LKD**
Contract Address	0xaF027427DC6d31A3e7e162A710a5Fe27e63E275F
Deployer Address	0x7ab8B73FCd3AbB89F99d96694fa7701D2a83Ce34
Owner Address	0x7ab8B73FCd3AbB89F99d96694fa7701D2a83Ce34
Tax Fees Buy	0%
Tax Fees Sell	0%
Gas Used for Buy	*will be updated after the DEX listing*
Gas Used for Sell	*will be updated after the DEX listing*
Contract Created	Apr-05-2022 08:33:11 PM +UTC
Initial Liquidity	*will be updated after the DEX listing*
Liquidity Status	Locked
Unlocked Date	*will be updated after the DEX listing*
Verified CA	Yes
Compiler	v0.6.12+commit.27d51765
Optimization	Yes with 200 runs
Sol License	MIT License
Top 5 Holders	*will be updated after the DEX listing*
Other	default evmVersion

TAX

BUY	0%	**SELL**	0%

OVERVIEW

Mint Function

- Mint functions found (This token minting function is useless, user can’t send to zero address, and there is no burn function, because default supply is 10,000,000 Token, and the condition limit is same).

Fees

- Buy 0%.
- Sell 0%.

Tx Amount

- Owner cannot set max tx amount.

Transfer Pausable

- Owner cannot pause.

Blacklist

- Owner cannot blacklist.

Ownership

- Owner cannot take back ownership.

Proxy

- This contract has no proxy.

Anti Whale

- Owner cannot limit the number of wallet holdings.

Trading Cooldown

- Owner cannot set the selling time interval.

Token Metrics

Rank	Address	Quantity	Percentage	Analytics
1	0x7ab8b73fcd3abb89f99d96694fa7701d2a83ce34	4,017,088.2	40.1709%	
2	0x7a43577cf71f877b4c94f3a5f949d0845ff8b6d6	2,000,000	20.0000%	
3	0xe306001f1d264bbd1028e1530a1eb9c6825351d9	1,500,000	15.0000%	
4	0x0b0c654b9839e2a060f7d89005418baaba52d4bd	1,000,000	10.0000%	
5	0xd0fbdbb74d59c79a6d8e8352473d3829d97dd4ba	666,660.6	6.6666%	
6	0x4314ba8b840d4953c1a939b677a9aef1f8079b07	399,184.8	3.9918%	
7	0xb76191e68906c78bcd93f50d11001caec6f31192	335,150.4	3.3515%	
8	0x77c7674faf29c41e797bae8f2923e0c8036fb2de	6,726.6	0.0673%	
9	0x769ec031490cca96681f13c0f0f1e496def8c169	6,011.4	0.0601%	
10	0x354bfe3317304285ea17b73d16b3bd88784a7775	6,008.4	0.0601%	

Team Review

The LinkDao team has a nice website, their website is professionally built and the Smart contract is well developed, their social media is growing with over 37,933 people in their telegram group (count in audit date).

Official Website And Social Media

Website: https://linkdao.network/

Telegram Group: https://t.me/linkdao_network

Twitter: https://twitter.com/LinkdaoN

MANUAL CODE REVIEW

🟢 Minor-risk

1 minor-risk code issue found

Could be fixed, and will not bring problems.

1. The return value of an external transfer/transferFrom return value is checked. Recommendation: use SafeERC20, or ensure that the transfer/transferFrom return value is checked

```
function transferFrom(
    address sender,
    address recipient,
    uint256 amount
  ) external returns (bool);
```

🟡 Medium-risk

0 medium-risk code issues found

Should be fixed, could bring problems.

🔴 High-Risk

1 high-risk code issues found

Must be fixed, and will bring problem.

```
function mint(uint256 amount) public onlyOwner returns (bool) {
        _mint(_msgSender(), amount);
        return true;
    }
```

🔴 Critical-Risk

0 critical-risk code issues found

Must be fixed, and will bring problem.

EXTRA NOTES SMART CONTRACT

1. IBEP20

```
interface IBEP20 {
  /**
   * @dev Returns the number of tokens in existence.
   */
  function totalSupply() external view returns (uint256);
…
  function balanceOf(address account) external view returns (uint256);
…
  function transfer(address recipient, uint256 amount) external returns (bool);
…
  function allowance(address owner, address spender) external view returns (uint256);
…
  function approve(address spender, uint256 amount) external returns (bool);
…
  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.
   */
  event Transfer(address indexed from, address indexed to, uint256 value);
…
}
```

IERC20 Normal Base Template

2. SafeMath Contract

```
library SafeMath {
...
    function add(uint256 a, uint256 b) internal pure returns
(uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }
...
    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 mod(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        unchecked {
            require(b > 0, errorMessage);
            return a % b;
        }
    }
}
```

Standard Safemath contract

3. BEP20 Contract

```
contract BEP20 is Context, IBEP20, Ownable {
    uint256 private constant _preMineSupply = 10000000 * 1e18;
    uint256 private constant _maxSupply = 10000000 * 1e18;
    using SafeMath for uint256;
    using Address for address;

    mapping(address => uint256) private _balances;

    mapping(address => mapping(address => uint256)) private
_allowances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
    uint8 private _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.
     */
    constructor(string memory name, string memory symbol) public {
        _name = name;
        _symbol = symbol;
        _decimals = 18;

        _mint(msg.sender, _preMineSupply);
    }

    /**
     * @dev Returns the bep token owner.
     */
    function getOwner() external override view returns (address) {
```

```
        return owner();
    }

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

    /**
     * @dev Returns the token decimals.
     */
    function decimals() public override view returns (uint8) {
        return _decimals;
    }

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

    /**
     * @dev See {BEP20-totalSupply}.
     */
    function totalSupply() public override view returns (uint256)
{
        return _totalSupply;
    }

        function preMineSupply() public override view returns
(uint256) {
        return _preMineSupply;
    }

    function maxSupply() public override view returns (uint256) {
        return _maxSupply;
    }
```

```
    /**
     * @dev See {BEP20-balanceOf}.
     */
    function balanceOf(address account) public override view
returns (uint256) {
        return _balances[account];
    }

    /**
     * @dev See {BEP20-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
override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    /**
     * @dev See {BEP20-allowance}.
     */
    function allowance(address owner, address spender) public
override view returns (uint256) {
        return _allowances[owner][spender];
    }

    /**
     * @dev See {BEP20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public
override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }
```

```
    /**
     * @dev See {BEP20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance.
This is not
     * required by the EIP. See the note at the beginning of
{BEP20};
     *
     * 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 override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(
            sender,
            _msgSender(),
            _allowances[sender][_msgSender()].sub(amount, 'BEP20:
transfer amount exceeds allowance')
        );
        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 {BEP20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
```

```
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256
addedValue) public returns (bool) {
        _approve(_msgSender(), spender,
_allowances[_msgSender()][spender].add(addedValue));
        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 {BEP20-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 returns (bool) {
        _approve(
            _msgSender(),
            spender,

_allowances[_msgSender()][spender].sub(subtractedValue, 'BEP20:
decreased allowance below zero')
        );
        return true;
    }

    /**
     * @dev Creates `amount` tokens and assigns them to
`msg.sender`, increasing
     * the total supply.
```

```
     *
     * Requirements
     *
     * - `msg.sender` must be the token owner
     */
    function mint(uint256 amount) public onlyOwner returns (bool)
{
        _mint(_msgSender(), amount);
        return true;
    }

    /**
     * @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 {
        require(sender != address(0), 'BEP20: transfer from the
zero address');
        require(recipient != address(0), 'BEP20: transfer to the
zero address');

        _balances[sender] = _balances[sender].sub(amount, 'BEP20:
transfer amount exceeds balance');
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }
```

```
    /** @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
returns(bool) {
        require(account != address(0), 'BEP20: mint to the zero
address');
        if (amount.add(_totalSupply) > _maxSupply) {
            return false;
        }

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

    /**
     * @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 {
        require(account != address(0), 'BEP20: burn from the zero
address');

        _balances[account] = _balances[account].sub(amount,
'BEP20: burn amount exceeds balance');
```

```
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }
    function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal {
        require(owner != address(0), 'BEP20: approve from the zero
address');
        require(spender != address(0), 'BEP20: approve to the zero
address');

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`.`amount` is
then deducted
     * from the caller's allowance.
     *
     * See {_burn} and {_approve}.
     */
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(
            account,
            _msgSender(),
            _allowances[account][_msgSender()].sub(amount, 'BEP20:
burn amount exceeds allowance')
        );
    }
}
```

4. LinkDao Contract

```
contract LINKDAO_Token is BEP20('LinkDao', 'LKD') {
    /// @notice Creates `_amount` token to `_to`. Must only be
called by the owner (MasterChef).
    function mint(address _to, uint256 _amount) public onlyOwner {
        _mint(_to, _amount);
        _moveDelegates(address(0), _delegates[_to], _amount);
    }

    // Copied and modified from YAM code:
    //
https://github.com/yam-finance/yam-protocol/blob/master/contracts/
token/YAMGovernanceStorage.sol
    //
https://github.com/yam-finance/yam-protocol/blob/master/contracts/
token/YAMGovernance.sol
    // Which is copied and modified from COMPOUND:
    //
https://github.com/compound-finance/compound-protocol/blob/master/
contracts/Governance/Comp.sol


    mapping (address => address) internal _delegates;

    /// @notice A checkpoint for marking number of votes from a
given block
    struct Checkpoint {
        uint32 fromBlock;
        uint256 votes;
    }

    /// @notice A record of votes checkpoints for each account, by
index
    mapping (address => mapping (uint32 => Checkpoint)) public
checkpoints;

    /// @notice The number of checkpoints for each account
    mapping (address => uint32) public numCheckpoints;

    /// @notice The EIP-712 typehash for the contract's domain
    bytes32 public constant DOMAIN_TYPEHASH =
keccak256("EIP712Domain(string name,uint256 chainId,address
```

```
verifyingContract)");

    /// @notice 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)");

    /// @notice A record of states for signing / validating
signatures
    mapping (address => uint) public nonces;

      /// @notice An event thats emitted when an account changes
its delegate
    event DelegateChanged(address indexed delegator, address
indexed fromDelegate, address indexed toDelegate);

    /// @notice An event thats emitted when a delegate account's
vote balance changes
    event DelegateVotesChanged(address indexed delegate, uint
previousBalance, uint newBalance);

    /**
     * @notice Delegate votes from `msg.sender` to `delegatee`
     * @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
    */
    function delegate(address delegatee) external {
        return _delegate(msg.sender, delegatee);
    }
```

```
    /**
     * @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
     */
    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), "LKD::delegateBySig:
invalid signature");
        require(nonce == nonces[signatory]++, "LKD::delegateBySig:
invalid nonce");
        require(now <= expiry, "LKD::delegateBySig: signature
expired");
        return _delegate(signatory, delegatee);
    }

    /**
     * @notice Gets the current votes balance for `account`
     * @param account The address to get votes balance
     * @return The number of current votes for `account`
     */
    function getCurrentVotes(address account)
        external
        view
        returns (uint256)
    {
        uint32 nCheckpoints = numCheckpoints[account];
        return nCheckpoints > 0 ?
checkpoints[account][nCheckpoints - 1].votes : 0;
    }

    /**
     * @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
     */
    function getPriorVotes(address account, uint blockNumber)
        external
```

```
        view
        returns (uint256)
    {
        require(blockNumber < block.number, "LKD::getPriorVotes:
not yet determined");

        uint32 nCheckpoints = numCheckpoints[account];
        if (nCheckpoints == 0) {
            return 0;
        }

        // First check most recent balance
        if (checkpoints[account][nCheckpoints - 1].fromBlock <=
blockNumber) {
            return checkpoints[account][nCheckpoints - 1].votes;
        }

        // Next check implicit zero balance
        if (checkpoints[account][0].fromBlock > blockNumber) {
            return 0;
        }

        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;
    }

    function _delegate(address delegator, address delegatee)
        internal
    {
```

```
        address currentDelegate = _delegates[delegator];
        uint256 delegatorBalance = balanceOf(delegator); //
balance of underlying LKDs (not scaled);
        _delegates[delegator] = delegatee;

        emit DelegateChanged(delegator, currentDelegate,
delegatee);

        _moveDelegates(currentDelegate, delegatee,
delegatorBalance);
    }

    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 = srcRepOld.sub(amount);
                _writeCheckpoint(srcRep, srcRepNum, srcRepOld,
srcRepNew);
            }

            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);
                _writeCheckpoint(dstRep, dstRepNum, dstRepOld,
dstRepNew);
            }
        }
    }

    function _writeCheckpoint(
        address delegatee,
        uint32 nCheckpoints,
        uint256 oldVotes,
        uint256 newVotes
```

```
    )
        internal
    {
        uint32 blockNumber = safe32(block.number,
"LKD::_writeCheckpoint: block number exceeds 32 bits");

        if (nCheckpoints > 0 &&
checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber)
{
            checkpoints[delegatee][nCheckpoints - 1].votes =
newVotes;
        } else {
            checkpoints[delegatee][nCheckpoints] =
Checkpoint(blockNumber, newVotes);
            numCheckpoints[delegatee] = nCheckpoints + 1;
        }

        emit DelegateVotesChanged(delegatee, oldVotes, newVotes);
    }

    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;
    }
}
```

5. **MInt Function** - (This token minting function is useless, user can't send to zero address, and there is no burn function, because default supply is 10,000,000 Token, and the condition limit is same)

```
function mint(uint256 amount) public onlyOwner returns (bool) {
        _mint(_msgSender(), amount);
        return true;
    }

function _mint(address account, uint256 amount) internal
returns(bool) {
        require(account != address(0), 'BEP20: mint to the zero
address');
        if (amount.add(_totalSupply) > _maxSupply) {
            return false;
        }

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
```

READ CONTRACT (ONLY NEED TO KNOW)

1. decimals
18 uint8
(Shows Contract Decimals)

2. getOwner
0x7ab8b73fcd3abb89f99d96694fa7701d2a83ce34 address

3. maxSupply
100000000000000000000000000 uint256

4. name
LinkDao string

5. symbol
LKD string

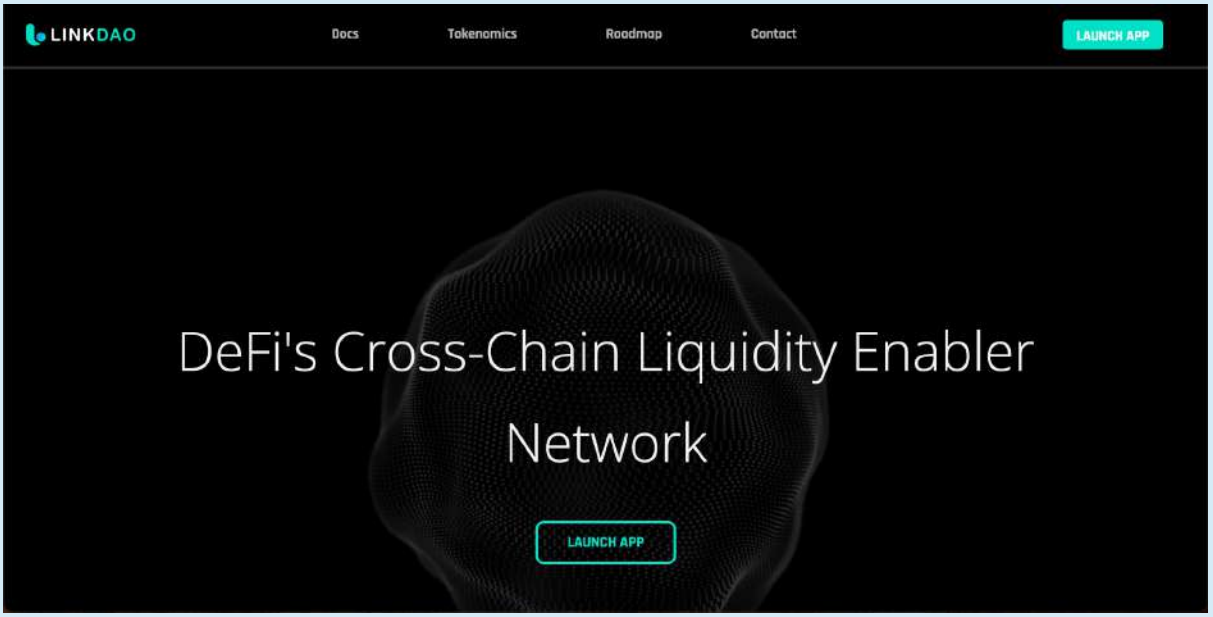
WRITE CONTRACT

1. mint
amount (uint256)
(The form is filled with the amount for minting)

2. renounceOwnership
(Renouncing ownership will leave the contract without an owner, thereby removing any functionality that is only available to the owner)

3. transferOwnership
newOwner (address)
(Its function is to change the owner)

WEBSITE REVIEW



- **Mobile Friendly**
- **Contains no code error**
- **SSL Secured (By R3 SSL)**

Web-Tech stack: jQuery, Bootstrap, Wordpress

Domain .com (hostinger) - Tracked by whois

First Contentful Paint:	492ms
Fully Loaded Time	1.6s
Performance	100%
Accessibility	94%
Best Practices	92%
SEO	91%

RUG-PULL REVIEW

Based on the available information analyzed by us, we come to the following conclusions:

- Locked liquidity (Locked by pinksale)

  (*Will be updated after DEX listing*)

- TOP 5 Holder.

  (*Will be updated after DEX listing*)

- The team hasn't done KYC yet.

- Owner can mint - (This token minting function is useless, user can’t send to zero address, and there is no burn function, because default supply is 10,000,000 Token, and the condition limit is same)

HONEYPOT REVIEW

- Ability to sell.
- The owner is not able to pause the contract.
- The owner can’t set fees

Note: Please check the disclaimer above and note, that the audit makes no statements or warranties on the business model, investment attractiveness, or code sustainability. The report is provided for the only contract mentioned in the report and does not include any other potential contracts deployed by the project owner.