



JohnWick Sec

Guard the internet of value



# ETH SMART CONTRACT AUDIT REPORT

JOHNWICK SECURITY LAB

[WWW.JOHNWICK.IO](http://WWW.JOHNWICK.IO)

John Wick Security Lab received the Lendefi (company/team) LDFI project smart contract code audit requirements on 2021/02/02.

Project Name: Lendefi Token (LDFI)

Smart Contract Address:

<https://etherscan.io/address/0x5479d565e549f3ecdbde4ab836d02d86e0d6a8c7#code>

Audit Number: 20210205

Audit Date: 20210202

Audit Category and Result:

Category	Sub-category	Result(Pass/Not Pass)
Contract vulnerability	Integer overflow	Pass
	Race condition	Pass
	Denial of service	Pass
	Logical vulnerability	Pass
	Hardcoded address	Pass
	Function input parameter check	Pass
	Function access control bypass	Pass
	Random number generation	Pass
	Random number use	Pass
Contract specification	Solidity compiler version	Pass
	Event use	Pass
	fallback function use	Pass
	Constructor use	Pass
	Function visibility declaration	Pass
	Variable storage declaration	Pass
	Deprecated keyword use	Pass
	ERC20/223 standard	Pass
	ERC721 standard	Pass
Business risk	Able to arbitrarily create token	Pass
	Able to arbitrarily destroy token	Pass
	Able to arbitrarily suspend tx.	Pass
	"Short address" attack	Pass
	"Fake recharge" attack	Pass
GAS optimization	assert()/require()	Pass
	Loop(for/while) optimization	Pass
	Storage optimization	Pass
Automated fuzzing		Pass

(Other unknown security vulnerabilities and Ethereum design flaws are not included in this audit responsibility)

**Audit Result: PASS****Auditor: John Wick Security Lab**

(Disclaimer: The John Wick Security Lab issues this report based on the facts that have occurred or existed before the issuance of this report and assumes corresponding responsibility in this regard. For the facts that occur or exist after the issuance of this report, the John Wick Security Lab cannot judge the security status of its smart contracts and does not assume any responsibility for it. The safety audit analysis and other contents of this report are based on the relevant materials and documents provided by the information provider to the John Wick Security Lab when the report is issued (referred to as the information provided). The John Wick Security Lab assumes that there is no missing, falsified, deleted, or concealed information provided. If the information provided is missing, falsified, deleted, concealed, or the information provider's response is inconsistent with the actual situation, the John Wick Security Lab shall not bear any responsibility for the resulting loss and adverse effects.)

**Audit Details:**

//JohnWick:

```
453: contract ERC20 is Context, IERC20 {
454:     using SafeMath for uint256;
```

This contract uses the `SafeMath` function library to avoid potential integer overflow problems, which is in line with the recommended practice.

//JohnWick:

```
745: function rescueTokens(address token, address to, uint256 amount)
external onlyOwner {
746:     require(token != to, "Invalid address");
747:     require(ERC20(token).transfer(to, amount), "Retrieve failed");
748: }
```

The contract owner can use the `rescueTokens(address token, address to, uint256 amount)` function to return the amount of token that was incorrectly transferred to this contract from address `to`, which is in line with the recommended practice.

//JohnWick:

```
735: ERC20 ('Lendefi Token', 'LDFI') {
736:     uint _decimals = 18;
```

The number of decimal points in this contract is 18, which is consistent with the number of decimal points of Ethereum's base currency `ETH`, which is in line with the recommended practice.

**Note:** The line number of the code involved in the audit details is based on the verified contract source code uploaded by the project party at etherscan.io, which is also displayed as a backup in the **Smart Contract Source Code** section of this report.

#### Smart Contract Source Code:

```
/**
 *Submitted for verification at Etherscan.io on 2021-01-20
 */

pragma solidity 0.6.2;

// SPDX-License-Identifier: MIT
/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending
 and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    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;
    }
}

// SPDX-License-Identifier: MIT
/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
```

```
* @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.
 */
event Transfer(address indexed from, address indexed to, uint256 value);

/**
 * @dev Emitted when the allowance of a `spender` for an `owner` is set by
 * a call to {approve}. `value` is the new allowance.
 */
event Approval(address indexed owner, address indexed spender, uint256
value);
}

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

```
* @dev Returns the addition of two unsigned integers, reverting on
* overflow.
*
* Counterpart to Solidity's `+` operator.
*
* Requirements:
*
* - Addition cannot overflow.
*/
function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");

    return c;
}

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

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

```
        return c;
    }

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

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

        return c;
    }

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



```
/**
 * @dev Returns the integer division of two unsigned integers. Reverts with
custom message on
 * division by zero. The result is rounded towards zero.
 *
 * Counterpart to Solidity's `/` operator. Note: this function uses a
 * `revert` opcode (which leaves remaining gas untouched) while Solidity
 * uses an invalid opcode to revert (consuming all remaining gas).
 *
 * Requirements:
 *
 * - The divisor cannot be zero.
 */
function div(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
    require(b > 0, errorMessage);
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't
hold

    return c;
}

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

/**
 * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
 * Reverts with custom message when dividing by zero.
```

```
*
* Counterpart to Solidity's `%` operator. This function uses a `revert`
* opcode (which leaves remaining gas untouched) while Solidity uses an
* invalid opcode to revert (consuming all remaining gas).
*
* Requirements:
*
* - The divisor cannot be zero.
*/
function mod(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
    require(b != 0, errorMessage);
    return a % b;
}

contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed
newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(_owner == _msgSender(), "Ownable: caller is not the owner");
        _;
    }
}
```

```
/**
 * @dev Leaves the contract without owner. It will not be possible to call
 * `onlyOwner` functions anymore. Can only be called by the current owner.
 *
 * NOTE: Renouncing ownership will leave the contract without an owner,
 * thereby removing any functionality that is only available to the owner.
 */
function renounceOwnership() public virtual onlyOwner {
    emit OwnershipTransferred(_owner, address(0));
    _owner = address(0);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Can only be called by the current owner.
 */
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero
address");
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
}
}

// SPDX-License-Identifier: MIT
/**
 * @dev Collection of functions related to the address type
 */
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) {
    // According to EIP-1052, 0x0 is the value returned for not-yet created
accounts
    // and
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
returned
    // for accounts without code, i.e. `keccak256('')`
    bytes32 codehash;
    bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
    // solhint-disable-next-line no-inline-assembly
    assembly { codehash := extcodehash(account) }
    return (codehash != accountHash && codehash != 0x0);
}

/**
 * @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-}[`functionCallWi
thValue`], 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);
    }
}
}
}

contract ERC20 is Context, IERC20 {
    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;
    }

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

```
/**
 * @dev Returns the symbol of the token, usually a shorter version of the
 * name.
 */
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}.
 */
function decimals() public view returns (uint8) {
    return _decimals;
}

/**
 * @dev See {IERC20-totalSupply}.
 */
function totalSupply() public view override returns (uint256) {
    return _totalSupply;
}

/**
 * @dev See {IERC20-balanceOf}.
 */
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);
    return true;
}

/**
 * @dev See {IERC20-allowance}.
 */
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.
 */
function approve(address spender, uint256 amount) public virtual override
returns (bool) {
    _approve(_msgSender(), spender, amount);
    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);
    _approve(sender, _msgSender(),
_allowances[sender][_msgSender()].sub(amount, "ERC20: 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 {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));
    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"));
        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 virtual {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(sender, recipient, amount);

        _balances[sender] = _balances[sender].sub(amount, "ERC20: 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 virtual {
        require(account != address(0), "ERC20: mint to the zero address");
```

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

        _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 virtual {
        require(account != address(0), "ERC20: burn from the zero address");

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

        _balances[account] = _balances[account].sub(amount, "ERC20: burn
amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner`'s tokens.
     *
     * This is internal function is equivalent to `approve`, and can be used
to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(address owner, address spender, uint256 amount) internal
```

```
virtual {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

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

/**
 * @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 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 { }
}

/**
 * @title LendefiToken
 * @dev Lendefi ERC20 Token
 */
contract LendefiToken is ERC20, Ownable {
    constructor ()
    public
```

```
ERC20 ('Lendefi Token', 'LDFI') {
    uint _decimals = 18;
    uint totalSupply = 10000000 * 10**uint256(_decimals);
    _mint(msg.sender, totalSupply);
}

function burn(uint256 amount) external {
    _burn(msg.sender, amount);
}

function rescueTokens(address token, address to, uint256 amount) external
onlyOwner {
    require(token != to, "Invalid address");
    require(ERC20(token).transfer(to, amount), "Retrieve failed");
}
}
```