Smart Contract
Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the Neopin Token on 2022.03.11. The following are the details and results of this smart contract security audit:

Token Name :

Neopin Token

The contract address :

https://github.com/Neopin/Contracts/tree/master

commit: 666931973823ad5caa76e0f8b1f9fa0f8f9532f7

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

NO.	Audit Items	Result
1	Replay Vulnerability	Passed
2	Denial of Service Vulnerability	Passed
3	Race Conditions Vulnerability	Passed
4	Authority Control Vulnerability	Passed
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Passed
8	Uninitialized Storage Pointers Vulnerability	Passed
9	Arithmetic Accuracy Deviation Vulnerability	Passed
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed

NO.	Audit Items	Result
12	Scoping and Declarations Audit	Passed
13	Safety Design Audit	Passed
14	Non-privacy/Non-dark Coin Audit	Passed

Audit Result : Passed

Audit Number : 0X002203160001

Audit Date : 2022.03.11 - 2022.03.16

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that contains the tokenVault section. The total amount of contract tokens can be changed, the owner can burn his own tokens through the burn function. SafeMath security module is used, which is a recommended approach. The contract does not have the Overflow and the Race Conditions issue. During the audit, we found the following information:

1. The owner can add new members to gain the locked amount of tokens and the mstart time is not checked. After communication with the project team, they express that the labor contract was the standard when performing Operation Lockup, and each developer's labor contract was signed before the token contract issuance.
2. The currentAllowance and the require check make the approve function a redundant part.

The source code:

Lockup.sol

```
// SPDX-License-Identifier: MIT
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity 0.8.10;

import "./zepplin/ERC20/SafeMath.sol";
```

```
import "./zepplin/ERC20/Ownable.sol";
import "./zepplin/ERC20/SafeERC20.sol";
import "./NeopinToken.sol";

contract Lockup is Ownable {
    using SafeERC20 for IERC20;
    using SafeMath for uint256;

    address private _npt;
    address private _owner;
    // uint256 private minfoSize = 0;
    mapping (address => MInfo) private _mapInfo;
    mapping(address => uint256) balances;
    mapping(address => mapping(address => uint256)) allowed;

    event DelMember(address member);
    event AddMember(address member, uint256 paid);
    event UpdateMInfo(address member, uint256 paid);
    event TrsMember(address member, uint256 paid);

          constructor(address cnt) {
                _npt = cnt;
                _owner = msg.sender;
        }

        struct MInfo {
        uint256 start;
        bool status;
    }

    modifier onlyMember() {
        require(isMember(msg.sender), "Role : msg sender is not momber");
        _;
    }

    function _getLimitBalance(address member) internal view returns(uint256 res){
        require(isMember(member), "not found member");
        uint256 curTime = block.timestamp;

        MInfo memory _info = _mapInfo[member];
        uint256 init = _info.start;
        uint256 total = allowed[_owner][member];

        if (init + ( 1260 * 1 days ) < curTime)
```

```
            res = total * 20/20;
        else if(init + ( 990 * 1 days ) < curTime)
            res = total * 14/20;
        else if(init + ( 720 * 1 days ) < curTime)
            res = total * 9/20;
        else if(init + ( 450 * 1 days ) < curTime)
            res = total * 4/20;
        else if(init + ( 270 * 1 days ) < curTime)
            res = total * 2/20;
        else if (init + (90 * 1 days ) < curTime)
            res = total * 1/20;
        else
            res = 0;
    }
    //SlowMist// The owner can add a new members to gain the locked amount of tokens and the mstart time is not checked
    function setMember(address member, uint256 mstart, uint256 amount) external onlyOwner {
        require(!isMember(member), "already join member");
        require(amount > 0, "fail amount limit");

        allowed[msg.sender][member] = amount * 1e18;
        balances[member] = 0;

        _mapInfo[member] = MInfo({
            start : mstart,
            status : true
        });

        // minfoSize ++;
        emit AddMember(member, amount);
    }

    function _setPaidBalance(address account, uint256 amount) internal
        returns (bool)
        {
        require(isMember(account), "not found member");
        balances[account] = SafeMath.add(balances[account], amount);

        emit UpdateMInfo(account, amount);
        return true;
    }

    function payableBalance(address account) public view
```

```
        returns (uint256 total)
        {
        require(isMember(account), "not found member");
        total = SafeMath.sub(_getLimitBalance(account),  balances[account]);
    }

    function isMember(address account) public view returns(bool) {
      return _mapInfo[account].status;
    }

    function removeMember(address account) external onlyOwner{
        require(isMember(account));
        balances[account] = 0 ;
        allowed[_owner][account] = 0;
        delete _mapInfo[account];
        emit DelMember(account);
    }

    function withdraw(uint256 amount) public {
        // require(IERC20(address(this)).allowance(_owner, msg.sender) >= amount,
"check the balances") ;
        require(payableBalance(msg.sender) >= amount, "check the balances");
        require(_setPaidBalance(msg.sender, amount), "check the total balances");
        require(IERC20(address(_npt)).transferFrom(_owner, msg.sender, amount));
        emit TrsMember(msg.sender, amount);
    }
}
```

NeopinToken.sol

```
// SPDX-License-Identifier: GPL-3.0
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity 0.8.10;

import "./zepplin/ERC20/ERC20.sol";
import "./zepplin/ERC20/SafeERC20.sol";

contract NeopinToken is ERC20 {
    using SafeERC20 for IERC20;
    address public _owner;
    // mapping(address => uint256) balances;
    // mapping(address => mapping(address => uint256)) allowed;
```

```
    event Burned(address request, uint256 amount);

    constructor() ERC20("NEOPIN Token", "NPT") {
        _owner = msg.sender;
        _mint(msg.sender, 1000000000e18);
    }

    function burn(uint256 _amount) external returns (bool) {
        require(msg.sender == _owner);
        _burn(msg.sender, _amount);
        emit Burned(msg.sender, _amount);
        return true;
    }
}
```

Address.sol

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

pragma solidity 0.8.10;

/**
 * @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) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        assembly {
            size := extcodesize(account)
        }
        return size > 0;
    }

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

        (bool success, ) = recipient.call{value: amount}("");
        require(
            success,
            "Address: unable to send value, recipient may have reverted"
        );
    }
```

```
    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain `call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?
highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data)
        internal
        returns (bytes memory)
    {
        return functionCall(target, data, "Address: low-level call failed");
    }

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

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
```

```
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(
        address target,
        bytes memory data,
        uint256 value
    ) internal returns (bytes memory) {
        return
            functionCallWithValue(
                target,
                data,
                value,
                "Address: low-level call with value failed"
            );
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}
[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(
        address target,
        bytes memory data,
        uint256 value,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(
            address(this).balance >= value,
            "Address: insufficient balance for call"
        );
        require(isContract(target), "Address: call to non-contract");

        (bool success, bytes memory returndata) = target.call{value: value}(
            data
        );
```

```
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data)
        internal
        view
        returns (bytes memory)
    {
        return
            functionStaticCall(
                target,
                data,
                "Address: low-level static call failed"
            );
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}
[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal view returns (bytes memory) {
        require(isContract(target), "Address: static call to non-contract");

        (bool success, bytes memory returndata) = target.staticcall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
```

```
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data)
        internal
        returns (bytes memory)
    {
        return
            functionDelegateCall(
                target,
                data,
                "Address: low-level delegate call failed"
            );
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}
[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract");

        (bool success, bytes memory returndata) = target.delegatecall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function _verifyCallResult(
        bool success,
        bytes memory returndata,
        string memory errorMessage
    ) private pure returns (bytes memory) {
        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
```

```
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}
```

Context.sol

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

pragma solidity 0.8.10;

/*
 * @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 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) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes calldata) {
        this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}
```

ERC20.sol

```
// SPDX-License-Identifier: MIT
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity 0.8.10;

import "./IERC20.sol";
import "./IERC20Metadata.sol";
import "./Context.sol";

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-
mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20, IERC20Metadata {
    mapping(address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
```

```
/**
 * @dev Sets the values for {name} and {symbol}.
 *
 * The defaut value of {decimals} is 18. To select a different value for
 * {decimals} you should overload it.
 *
 * All two of these values are immutable: they can only be set once during
 * construction.
 */
constructor(string memory name_, string memory symbol_) {
    _name = name_;
    _symbol = symbol_;
}

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

/**
 * @dev Returns the symbol of the token, usually a shorter version of the
 * name.
 */
function symbol() external view virtual override 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 this function is
 * overridden;
 *
 * 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() external view virtual override returns (uint8) {
    return 18;
```

```
}

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

/**
 * @dev See {IERC20-balanceOf}.
 */
function balanceOf(address account)
    external
    view
    virtual
    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);
    emit Transfer(_msgSender(), recipient, amount);
    //SlowMist// The return value conforms to the EIP20 specification
    return true;
}

/**
 * @dev See {IERC20-allowance}.
```

```
     */
    function allowance(address owner, address spender)
        external
        view
        virtual
        override
        returns (uint256)
    {
        return _allowances[owner][spender];
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    //SlowMist// The currentAllowance and the require check make the approve function
as a redundant part and it is designed like this
    function approve(address spender, uint256 amount)
        external
        virtual
        override
        returns (bool)
    {
        uint256 currentAllowance = _allowances[spender][_msgSender()];
        require(
            currentAllowance >= amount,
            "ERC20: transfer amount exceeds allowance"
        );
        _approve(_msgSender(), spender, currentAllowance, amount);
        //SlowMist// The return value conforms to the EIP20 specification
        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
    ) external virtual override returns (bool) {
        _transfer(sender, recipient, amount);
        emit Transfer(_msgSender(), sender, recipient, amount);

        uint256 currentAllowance = _allowances[sender][_msgSender()];
        require(
            currentAllowance >= amount,
            "ERC20: transfer amount exceeds allowance"
        );
        _approve(
            sender,
            _msgSender(),
            currentAllowance,
            currentAllowance - amount
        );
        //SlowMist// The return value conforms to the EIP20 specification
        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)
        external
        virtual
        returns (bool)
```

```
{
    uint256 currentAllowance = _allowances[_msgSender()][spender];
    _approve(
        _msgSender(),
        spender,
        currentAllowance,
        currentAllowance + 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)
    external
    virtual
    returns (bool)
{
    uint256 currentAllowance = _allowances[_msgSender()][spender];
    require(
        currentAllowance >= subtractedValue,
        "ERC20: decreased allowance below zero"
    );
    _approve(
        _msgSender(),
        spender,
        currentAllowance,
        currentAllowance - subtractedValue
    );

    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");
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to the loss of token during transfer
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(sender, recipient, amount);

        uint256 senderBalance = _balances[sender];
        require(
            senderBalance >= amount,
            "ERC20: transfer amount exceeds balance"
        );
        _balances[sender] = senderBalance - amount;
        _balances[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 += amount;
        _balances[account] += 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);

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
        _balances[account] = accountBalance - amount;
        _totalSupply -= amount;

        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
     *
     * This 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.
     */
    //SlowMist// The currentAmount is not used, it is recommended to delete the redundant code
    function _approve(
        address owner,
        address spender,
        uint256 currentAmount,
        uint256 amount
    ) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        //SlowMist// This kind of check is very good, avoiding user mistake leading to approve errors
        require(spender != address(0), "ERC20: approve to the zero address");
        require(
            currentAmount == _allowances[owner][spender],
            "ERC20: invalid currentAmount"
        );
        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, currentAmount, amount);
    }

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

```
    ) internal virtual {}
}
```

IERC20.sol

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

pragma solidity 0.8.10;

/**
 * @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);

event Transfer(
```

```
        address indexed spender,
        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 oldValue,
        uint256 value
    );
}
```

IERC20Metadata.sol

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

pragma solidity ^0.8.0;

import "./IERC20.sol";

/**
 * @dev Interface for the optional metadata functions from the ERC20 standard.
 *
 * _Available since v4.1._
 */
interface IERC20Metadata is IERC20 {
    /**
     * @dev Returns the name of the token.
     */
    function name() external view returns (string memory);

    /**
     * @dev Returns the symbol of the token.
     */
    function symbol() external view returns (string memory);
```

```
    /**
     * @dev Returns the decimals places of the token.
     */
    function decimals() external view returns (uint8);
}
```

Ownable.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./Context.sol";

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract 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() {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    /**
```

```
     * @dev Returns the address of the current owner.
     */
    function owner() public view virtual 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 Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return _msgSender() == _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 {
        //SlowMist// This check is quite good in avoiding losing control of the
contract caused by user mistakes
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
```

```
    }
}
```

SafeERC20.sol

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.10;

import "./IERC20.sol";
import "./Address.sol";

/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your
contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using Address for address;

    function safeTransfer(
        IERC20 token,
        address to,
        uint256 value
    ) internal {
        _callOptionalReturn(
            token,
            abi.encodeWithSelector(token.transfer.selector, to, value)
        );
    }

    function safeTransferFrom(
        IERC20 token,
        address from,
        address to,
        uint256 value
    ) internal {
        _callOptionalReturn(
```

```
        token,
        abi.encodeWithSelector(token.transferFrom.selector, from, to, value)
    );
}

/**
 * @dev Deprecated. This function has issues similar to the ones found in
 * {IERC20-approve}, and its usage is discouraged.
 *
 * Whenever possible, use {safeIncreaseAllowance} and
 * {safeDecreaseAllowance} instead.
 */
function safeApprove(
    IERC20 token,
    address spender,
    uint256 value
) internal {
    // safeApprove should only be called when setting an initial allowance,
    // or when resetting it to zero. To increase and decrease it, use
    // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
    require(
        (value == 0) || (token.allowance(address(this), spender) == 0),
        "SafeERC20: approve from non-zero to non-zero allowance"
    );
    _callOptionalReturn(
        token,
        abi.encodeWithSelector(token.approve.selector, spender, value)
    );
}

function safeIncreaseAllowance(
    IERC20 token,
    address spender,
    uint256 value
) internal {
    uint256 newAllowance = token.allowance(address(this), spender) + value;
    _callOptionalReturn(
        token,
        abi.encodeWithSelector(
            token.approve.selector,
            spender,
            newAllowance
        )
    );
```

```
    }

    function safeDecreaseAllowance(
        IERC20 token,
        address spender,
        uint256 value
    ) internal {
        unchecked {
            uint256 oldAllowance = token.allowance(address(this), spender);
            require(
                oldAllowance >= value,
                "SafeERC20: decreased allowance below zero"
            );
            uint256 newAllowance = oldAllowance - value;
            _callOptionalReturn(
                token,
                abi.encodeWithSelector(
                    token.approve.selector,
                    spender,
                    newAllowance
                )
            );
        }
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a
contract), relaxing the requirement
     * on the return value: the return value is optional (but if data is returned, it
must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data
size checking mechanism, since
        // we're implementing it ourselves. We use {Address.functionCall} to perform
this call, which verifies that
        // the target address contains contract code and also asserts for success in
the low-level call.

        bytes memory returndata = address(token).functionCall(
            data,
            "SafeERC20: low-level call failed"
```

```
        );
        if (returndata.length > 0) {
            // Return data is optional
            require(
                abi.decode(returndata, (bool)),
                "SafeERC20: ERC20 operation did not succeed"
            );
        }
    }
}
```

SafeMath.sol

```
// SPDX-License-Identifier: MIT
//SlowMist// SafeMath security module is used, which is a recommended approach, but Solidity 0.8 has an integrated SafeMath
pragma solidity 0.8.10;

// CAUTION
// This version of SafeMath should only be used with Solidity 0.8 or later,
// because it relies on the compiler's built in overflow checks.

/**
 * @dev Wrappers over Solidity's arithmetic operations.
 *
 * NOTE: `SafeMath` is no longer needed starting with Solidity 0.8. The compiler
 * now has built in overflow checking.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b)
        internal
        pure
        returns (bool, uint256)
    {
        unchecked {
            uint256 c = a + b;
            if (c < a) return (false, 0);
```

```
            return (true, c);
        }
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b)
        internal
        pure
        returns (bool, uint256)
    {
        unchecked {
            if (b > a) return (false, 0);
            return (true, a - b);
        }
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b)
        internal
        pure
        returns (bool, uint256)
    {
        unchecked {
            // 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 (true, 0);
            uint256 c = a * b;
            if (c / a != b) return (false, 0);
            return (true, c);
        }
    }

    /**
```

```
     * @dev Returns the division of two unsigned integers, with a division by zero
flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b)
        internal
        pure
        returns (bool, uint256)
    {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a / b);
        }
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division
by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b)
        internal
        pure
        returns (bool, uint256)
    {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a % b);
        }
    }

    /**
     * @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) {
```

```
        return a + b;
    }

    /**
     * @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 a - b;
    }

    /**
     * @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) {
        return a * b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator.
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return a / b;
    }
```

```
    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
     * reverting 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 a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom
message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for
the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        unchecked {
            require(b <= a, errorMessage);
            return a - b;
        }
    }
```

```
    /**
     * @dev Returns the integer division of two unsigned integers, reverting with
custom message on
     * division by zero. The result is rounded towards 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).
     *
     * 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) {
        unchecked {
            require(b > 0, errorMessage);
            return a / b;
        }
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for
the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * 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) {
        unchecked {
            require(b > 0, errorMessage);
            return a % b;
        }
    }
}
```

Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist