Smart Contract
Security Audit Report

The SlowMist Security Team received the Covenant team's application for smart contract security audit of the COVN on 2022.06.17. The following are the details and results of this smart contract security audit:

Token Name :

COVN

The contract address :

https://etherscan.io/address/0x19ac2659599fd01c853de846919544276ad26f50

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
12	Scoping and Declarations Audit	Passed

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

Audit Result : Passed

Audit Number : 0X002206210002

Audit Date : 2022.06.17 - 2022.06.21

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that does not contain the tokenVault section. The total amount of contract tokens can be changed, users can burn their own tokens through the burn and burnFrom 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 role can freeze or unfreeze any user's account through the freezeAccount and unfreezeAccount function.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2021-12-06
*/
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
// SPDX-License-Identifier: MIT
pragma solidity 0.8.5;

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;
  }
}

abstract contract Ownable is Context {
  address internal _owner;

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

  function owner() public view virtual returns (address) {
    return _owner;
  }

  modifier onlyOwner() {
    require(owner() == _msgSender(), "Ownable: caller is not the owner");
    _;
  }

  function renounceOwnership() public virtual onlyOwner {
    emit OwnershipTransferred(_owner, address(0));
    _owner = address(0);
  }

  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;
  }
}

abstract contract Freezable is Context {
  event Freeze(address indexed holder);
  event Unfreeze(address indexed holder);

  mapping(address => bool) private _frozenAccount;

  modifier whenNotFrozen(address holder) {
    require(!_frozenAccount[holder]);
```

```
    _;
  }

  function isFrozen(address holder) public view virtual returns (bool frozen) {
    return _frozenAccount[holder];
  }

  function _freezeAccount(address holder) internal virtual returns (bool success) {
    require(!isFrozen(holder));
    _frozenAccount[holder] = true;
    emit Freeze(holder);
    success = true;
  }

  function _unfreezeAccount(address holder) internal virtual returns (bool success) {
    require(isFrozen(holder));
    _frozenAccount[holder] = false;
    emit Unfreeze(holder);
    success = true;
  }
}

abstract contract Pausable is Context {
  event Paused(address account);
  event Unpaused(address account);

  bool private _paused;

  constructor() {
    _paused = false;
  }

  function paused() public view virtual returns (bool) {
    return _paused;
  }

  modifier whenNotPaused() {
    require(!paused(), "Pausable: paused");
    _;
  }

  modifier whenPaused() {
    require(paused(), "Pausable: not paused");
    _;
```

```
  }

  function _pause() internal virtual whenNotPaused {
    _paused = true;
    emit Paused(_msgSender());
  }

  function _unpause() internal virtual whenPaused {
    _paused = false;
    emit Unpaused(_msgSender());
  }
}

interface IERC20 {
  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);

  event Transfer(address indexed from, address indexed to, uint256 value);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}

interface IERC20Metadata is IERC20 {
  function name() external view returns (string memory);

  function symbol() external view returns (string memory);

  function decimals() external view returns (uint8);
}

contract ERC20 is Context, IERC20, IERC20Metadata {
  mapping(address => uint256) private _balances;
```

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

  uint256 private _totalSupply;

  string internal _name;
  string internal _symbol;
  uint8 internal _decimals;

  function name() public view virtual override returns (string memory) {
    return _name;
  }

  function symbol() public view virtual override returns (string memory) {
    return _symbol;
  }

  function decimals() public view virtual override returns (uint8) {
    return _decimals;
  }

  function totalSupply() public view virtual override returns (uint256) {
    return _totalSupply;
  }

  function balanceOf(address account) public view virtual override returns (uint256)
{
    return _balances[account];
  }

  function transfer(address recipient, uint256 amount) public virtual override
returns (bool) {
    _transfer(_msgSender(), recipient, amount);
    //SlowMist// The return value conforms to the EIP20 specification
    return true;
  }

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

  function approve(address spender, uint256 amount) public virtual override returns
(bool) {
```

```
    _approve(_msgSender(), spender, amount);
    //SlowMist// The return value conforms to the EIP20 specification
    return true;
  }

  function transferFrom(
    address sender,
    address recipient,
    uint256 amount
  ) public virtual override returns (bool) {
    _transfer(sender, recipient, amount);

    uint256 currentAllowance = _allowances[sender][_msgSender()];
    require(currentAllowance >= amount, "ERC20: transfer amount exceeds allowance");
    unchecked {
      _approve(sender, _msgSender(), currentAllowance - amount);
    }
    //SlowMist// The return value conforms to the EIP20 specification
    return true;
  }

  function increaseAllowance(address spender, uint256 addedValue) public virtual
returns (bool) {
    _approve(_msgSender(), spender, _allowances[_msgSender()][spender] + addedValue);
    return true;
  }

  function decreaseAllowance(address spender, uint256 subtractedValue) public virtual
returns (bool) {
    uint256 currentAllowance = _allowances[_msgSender()][spender];
    require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below
zero");
    unchecked {
      _approve(_msgSender(), spender, currentAllowance - subtractedValue);
    }

    return true;
  }

  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");
    unchecked {
      _balances[sender] = senderBalance - amount;
    }
    _balances[recipient] += amount;

    emit Transfer(sender, recipient, amount);
  }

  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);
  }

  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");
    unchecked {
      _balances[account] = accountBalance - amount;
    }
    _totalSupply -= amount;

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

  function _approve(
    address owner,
```

```
    address spender,
    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");

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

  function _beforeTokenTransfer(
    address from,
    address to,
    uint256 amount
  ) internal virtual {}
}

abstract contract ERC20Burnable is Context, ERC20, Ownable {
  function burn(uint256 amount) public virtual {
    _burn(_msgSender(), amount);
  }
  //SlowMist// Because burnFrom() and transferFrom() share the allowed amount of approve(), if the agent be evil, there is the possibility of malicious burn
  function burnFrom(address account, uint256 amount) public virtual {
    uint256 currentAllowance = allowance(account, _msgSender());
    require(currentAllowance >= amount, "ERC20: burn amount exceeds allowance");
    unchecked {
      _approve(account, _msgSender(), currentAllowance - amount);
    }
    _burn(account, amount);
  }
}

contract Token is ERC20, Pausable, Freezable, ERC20Burnable {
  bool private initialized = false;

  function initialize(
    address initalizeOwner,
    string memory initalizeName,
    string memory initalizeSymbol,
    uint8 initalizeDecimals,
    uint256 initalizeSupply
```

```
  ) external {
    require(!initialized, "Token: already initialized");

    _owner = initalizeOwner;
    _name = initalizeName;
    _symbol = initalizeSymbol;
    _decimals = initalizeDecimals;
    _mint(initalizeOwner, initalizeSupply * (10**initalizeDecimals));

    initialized = true;
  }
  //SlowMist// Suspending all transactions upon major abnormalities is a recommended approach
  function pause() public onlyOwner {
    _pause();
  }

  function unpause() public onlyOwner {
    _unpause();
  }

  function freezeAccount(address holder) public onlyOwner {
    _freezeAccount(holder);
  }

  function unfreezeAccount(address holder) public onlyOwner {
    _unfreezeAccount(holder);
  }

  function _beforeTokenTransfer(
    address from,
    address to,
    uint256 amount
  ) internal override whenNotPaused whenNotFrozen(from) {
    super._beforeTokenTransfer(from, to, amount);
  }
}
```

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