# Smart Contract
# Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the GHUB on 2022.04.04. The following are the details and results of this smart contract security audit:

**Token Name :**

GHUB

**The contract address :**

https://scope.klaytn.com/account/0x4836cc1f355bb2a61c210eaa0cd3f729160cd95e?tabId=contractCode

**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 :** 0X002204060002

**Audit Date :** 2022.04.04 - 2022.04.06

**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 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 minter role can add a new minter role and the minter can mint tokens arbitrarily through the mint function and there is no upper limit on the amount of tokens that can be minted.

   After communication with the project team, they removed the minter role so that the mint function can not be used anymore and also no more tokens can be minted. The following are the transactions' links: https://scope.klaytn.com/tx/0x8ae3ad26bba0edbcb597682282e35caada58781fc2b37f60f18267a4be57fde7?tabId=internalTx

## The source code:

```
// SPDX-License-Identifier: MIT
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity >=0.8.0 <0.9.0;

/**
 * @title SafeMath
 * @dev Math operations with safety checks that revert on error
```

```
 */
//SlowMist// SafeMath security module is used, which is a recommended approach
library SafeMath {

  /**
  * @dev Multiplies two numbers, reverts on 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-solidity/pull/522
    if (a == 0) {
      return 0;
    }

    uint256 c = a * b;
    require(c / a == b);

    return c;
  }

  /**
  * @dev Integer division of two numbers truncating the quotient, reverts on division by zero.
  */
  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b > 0); // Solidity only automatically asserts when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold

    return c;
  }

  /**
  * @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater than minuend).
  */
  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b <= a);
    uint256 c = a - b;

    return c;
  }
```

```
  /**
  * @dev Adds two numbers, reverts on overflow.
  */
  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a);

    return c;
  }

  /**
  * @dev Divides two numbers and returns the remainder (unsigned integer modulo),
  * reverts when dividing by zero.
  */
  function mod(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b != 0);
    return a % b;
  }
}

library Address {
    function isContract(address account) internal view returns (bool) {
        uint256 size;

        assembly { size := extcodesize(account) }

        return (size > 0);
    }
}

library Roles {
    struct Role {
        mapping (address => bool) bearer;
    }

    function add(Role storage role, address account) internal {
        require(!has(role, account), "Roles: account already has role");

        role.bearer[account] = true;
    }

    function remove(Role storage role, address account) internal {
        require(has(role, account), "Roles: account does not have role");
```

```
        role.bearer[account] = false;
    }

    function has(Role storage role, address account) internal view returns (bool) {
        require(account != address(0), "Roles: account is the zero address");

        return role.bearer[account];
    }
}

contract MinterRole {
    using Roles for Roles.Role;

    Roles.Role private _minters;

    event MinterAdded(address indexed account);
    event MinterRemoved(address indexed account);

    modifier onlyMinter() {
        require(isMinter(msg.sender), "MinterRole: caller does not have the Minter
role");
        _;
    }

    constructor(address owner) {
        _addMinter(owner);
    }

    function isMinter(address account) public view returns (bool) {
        return _isMinter(account);
    }

    function addMinter(address account) public onlyMinter {
        _addMinter(account);
    }

    function renounceMinter() public {
        _removeMinter(msg.sender);
    }

    /* internal functions */

    function _isMinter(address account) internal view returns (bool) {
        return _minters.has(account);
```

```
    }

    function _addMinter(address account) internal {
        _minters.add(account);

        emit MinterAdded(account);
    }

    function _removeMinter(address account) internal {
        _minters.remove(account);

        emit MinterRemoved(account);
    }
}

contract PauserRole {
    using Roles for Roles.Role;

    Roles.Role private _pausers;

    event PauserAdded(address indexed account);
    event PauserRemoved(address indexed account);

    modifier onlyPauser() {
        require(isPauser(msg.sender), "PauserRole: caller does not have the Pauser
role");
        _;
    }

    constructor(address owner) {
        _addPauser(owner);
    }

    function isPauser(address account) public view returns (bool) {
        return _isPauser(account);
    }

    function addPauser(address account) public onlyPauser {
        _addPauser(account);
    }

    function renouncePauser() public {
        _removePauser(msg.sender);
    }
```

```
    /* internal functions */

    function _isPauser(address account) internal view returns (bool) {
        return _pausers.has(account);
    }

    function _addPauser(address account) internal {
        _pausers.add(account);

        emit PauserAdded(account);
    }

    function _removePauser(address account) internal {
        _pausers.remove(account);

        emit PauserRemoved(account);
    }
}

abstract contract Pausable is PauserRole {
    bool private _paused;

    event Paused(address account);
    event Unpaused(address account);

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

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

    constructor() {
        _paused = false;
    }

    function paused() public view returns (bool) {
        return _paused;
    }
    //SlowMist// Suspending all transactions upon major abnormalities is a
```

**recommended approach**

```
    function pause() public onlyPauser whenNotPaused {
        _paused = true;

        emit Paused(msg.sender);
    }

    function unpause() public onlyPauser whenPaused {
        _paused = false;

        emit Unpaused(msg.sender);
    }
}

contract OwnerRole is MinterRole, PauserRole {
    address payable private _owner;

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

    modifier onlyOwner() {
        require(msg.sender == _owner, "OwnerRole: caller is not owner");
        _;
    }

    constructor(address payable owner) MinterRole(owner)
                                       PauserRole(owner) {
        _owner = owner;
    }

    function isOwner(address account) public view returns (bool) {
        return (account == _owner);
    }

    function renounceOwnership() public onlyOwner {
        _transferOwnership(payable(address(0)));
    }

    function transferOwnership(address payable newOwner) public onlyOwner {
        //SlowMist// This check is quite good in avoiding losing control of the
contract caused by user mistakes
        require(newOwner != address(0), "OwnerRole: new owner is the zero address");
        require(newOwner != _owner, "OwnerRole: already is the owner");
```

```
        _transferOwnership(newOwner);
    }

    /* internal functions */

    function _transferOwnership(address payable newOwner) public onlyOwner {
        if (!_isMinter(newOwner)) {
            _addMinter(newOwner);
        }

        if (_isMinter(_owner)) {
            _removeMinter(_owner);
        }

        if (!_isPauser(newOwner)) {
            _addPauser(newOwner);
        }

        if (_isPauser(_owner)) {
            _removePauser(_owner);
        }

        emit OwnershipTransferred(_owner, newOwner);

        _owner = newOwner;
    }
}

abstract contract IKIP13 {
    function supportsInterface(bytes4 interfaceId) public virtual view returns
(bool);
}

abstract contract IKIP7 is IKIP13 {
    function totalSupply() public virtual view returns (uint256);
    function balanceOf(address account) public virtual view returns (uint256);
    function transfer(address recipient, uint256 amount) public virtual returns
(bool);
    function allowance(address owner, address spender) public virtual view returns
(uint256);
    function approve(address spender, uint256 amount) public virtual returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) public
virtual returns (bool);
    function safeTransfer(address recipient, uint256 amount) virtual public;
```

```
    function safeTransfer(address recipient, uint256 amount, bytes memory data)
virtual public;
    function safeTransferFrom(address sender, address recipient, uint256 amount,
bytes memory data) virtual public;
    function safeTransferFrom(address sender, address recipient, uint256 amount)
virtual public;

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

abstract contract IKIP7Metadata {
    function name() public virtual view returns (string memory);
    function symbol() public virtual view returns (string memory);
    function decimals() public virtual view returns (uint8);
}

abstract contract IKIP7Mintable {
    function mint(address account, uint256 amount) public virtual returns (bool);
}

abstract contract IKIP7Burnable {
    function burn(uint256 amount) public virtual;
    function burnFrom(address account, uint256 amount) public virtual;
}

contract KIP13 is IKIP13 {
    bytes4 private constant _INTERFACE_ID_KIP13 = 0x01ffc9a7;

    mapping(bytes4 => bool) private _supportedInterfaces;

    constructor() {
        _registerInterface(_INTERFACE_ID_KIP13);
    }

    function supportsInterface(bytes4 interfaceId) override public view returns
(bool) {
        return _supportedInterfaces[interfaceId];
    }

    /* internal functions */

    function _registerInterface(bytes4 interfaceId) internal {
        require(interfaceId != 0xffffffff, "KIP13: invalid interface id");
```

```
        _supportedInterfaces[interfaceId] = true;
    }
}

contract KIP7 is IKIP7, KIP13 {
    using SafeMath for uint256;
    using Address for address;

    bytes4 private constant _INTERFACE_ID_KIP7 = 0x65787371; /* IKIP7 */
    bytes4 private constant _KIP7_RECEIVED     = 0x9d188c22; /*
onKIP7Received(address,address,uint256,bytes) */

    uint256 private _totalSupply;

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

    constructor() {
        _registerInterface(_INTERFACE_ID_KIP7);
    }

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

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

    function transfer(address recipient, uint256 amount) virtual override public
returns (bool) {
        _transfer(msg.sender, recipient, amount);
        //SlowMist// The return value conforms to the KIP7 specification
        return true;
    }

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

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

```
        _approve(msg.sender, spender, value);
        //SlowMist// The return value conforms to the KIP7 specification
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) virtual
override public returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount));
        //SlowMist// The return value conforms to the KIP7 specification
        return true;
    }

    function safeTransfer(address recipient, uint256 amount) override public {
        safeTransfer(recipient, amount, "");
    }

    function safeTransfer(address recipient, uint256 amount, bytes memory data)
override public {
        transfer(recipient, amount);
        require(_checkOnKIP7Received(msg.sender, recipient, amount, data), "KIP7:
transfer to non KIP7Receiver implementer");
    }

    function safeTransferFrom(address sender, address recipient, uint256 amount)
override public {
        safeTransferFrom(sender, recipient, amount, "");
    }

    function safeTransferFrom(address sender, address recipient, uint256 amount,
bytes memory data) override public {
        transferFrom(sender, recipient, amount);
        require(_checkOnKIP7Received(sender, recipient, amount, data), "KIP7:
transfer to non KIP7Receiver implementer");
    }

    /* internal functions */

    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "KIP7: 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), "KIP7: transfer to the zero address");
```

```
        _balances[sender] = _balances[sender].sub(amount);
        _balances[recipient] = _balances[recipient].add(amount);

        emit Transfer(sender, recipient, amount);
    }

    function _approve(address owner, address spender, uint256 value) internal {
        require(owner != address(0), "KIP7: approve from the zero address");
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to approve errors
        require(spender != address(0), "KIP7: approve to the zero address");

        _allowances[owner][spender] = value;

        emit Approval(owner, spender, value);
    }

    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "KIP7: mint to the zero address");

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);

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

    function _burn(address account, uint256 value) internal {
        require(account != address(0), "KIP7: burn from the zero address");

        _totalSupply = _totalSupply.sub(value);
        _balances[account] = _balances[account].sub(value);

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

    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount));
    }

    function _checkOnKIP7Received(address from, address to, uint256 amount, bytes
memory data) internal returns (bool) {
        bool success;
        bytes memory retval;
```

```
        if (!to.isContract()) {
            return true;
        }

        (success, retval) = to.call(
            abi.encodeWithSelector(_KIP7_RECEIVED, msg.sender, from, amount, data)
        );
        if (retval.length != 0 && abi.decode(retval, (bytes4)) == _KIP7_RECEIVED) {
            return true;
        }

        return false;
    }
}

contract KIP7Metadata is IKIP7Metadata, KIP13 {
    bytes4 private constant _INTERFACE_ID_KIP7_METADATA = 0xa219a025;

    string private _name;
    string private _symbol;
    uint8  private _decimals;

    constructor(string memory name_, string memory symbol_, uint8 decimals_) {
        _name     = name_;
        _symbol   = symbol_;
        _decimals = decimals_;

        _registerInterface(_INTERFACE_ID_KIP7_METADATA);
    }

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

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

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

```
abstract contract KIP7Mintable is IKIP7Mintable, KIP13, KIP7, MinterRole {
    bytes4 private constant _INTERFACE_ID_KIP7_MINTABLE = 0xeab83e20;

    constructor() {
        _registerInterface(_INTERFACE_ID_KIP7_MINTABLE);
    }
    //SlowMist// The minter role can add a new minter role and the minter can mint tokens arbitrarily through the mint function and there is no upper limit on the amount of tokens that can be minted
    function mint(address account, uint256 amount) public override onlyMinter returns (bool) {
        _mint(account, amount);

        return true;
    }
}

contract KIP7Burnable is IKIP7Burnable, KIP13, KIP7 {
    bytes4 private constant _INTERFACE_ID_KIP7_BURNABLE = 0x3b5a0bf8;

    constructor() {
        _registerInterface(_INTERFACE_ID_KIP7_BURNABLE);
    }

    function burn(uint256 amount) public override {
        _burn(msg.sender, 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 override {
        _burnFrom(account, amount);
    }
}

abstract contract KIP7Pausable is KIP13, KIP7, Pausable {
    bytes4 private constant _INTERFACE_ID_KIP7_PAUSABLE = 0x4d5507ff;

    constructor() {
        _registerInterface(_INTERFACE_ID_KIP7_PAUSABLE);
    }

    function transfer(address to, uint256 value) virtual override public whenNotPaused returns (bool) {
        return super.transfer(to, value);
```

```
    }

    function transferFrom(address from, address to, uint256 value) virtual override
public whenNotPaused returns (bool) {
        return super.transferFrom(from, to, value);
    }

    function approve(address spender, uint256 value) virtual override public
whenNotPaused returns (bool) {
        return super.approve(spender, value);
    }
}

contract BasicToken is KIP7, KIP7Mintable, KIP7Burnable, KIP7Pausable, KIP7Metadata,
OwnerRole {
    constructor(address payable owner, string memory name, string memory symbol,
uint8 decimals, uint256 initialSupply) KIP7Metadata(name, symbol, decimals)

OwnerRole(owner) {
        if (initialSupply > 0) {
            _mint(owner, initialSupply);
        }
    }

    /* override functions */

    function transfer(address to, uint256 value) override(KIP7, KIP7Pausable) public
virtual returns (bool) {
        return KIP7Pausable.transfer(to, value);
    }

    function transferFrom(address from, address to, uint256 value) override(KIP7,
KIP7Pausable) public virtual returns (bool) {
        return KIP7Pausable.transferFrom(from, to, value);
    }

    function approve(address spender, uint256 value) override(KIP7, KIP7Pausable)
public returns (bool) {
        return KIP7Pausable.approve(spender, value);
    }
}
```

# 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