Smart Contract Security Audit Report

[2021]

The SlowMist Security Team received the Travala team's application for smart contract security audit of the AVA on 2021.08.09. The following are the details and results of this smart contract security audit:

Token Name :

AVA

The contract address :

https://github.com/travala/travala-ava-erc20-smartcontract/blob/master/AnyswapV5ERC20.sol

commit: 6bc6f1e3bd5fe54ccea1fc067e0ec184ec71878a

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	Some Risks
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

Audit Result : Medium Risk

Audit Number : 0x002108110002

Audit Date : 2021.08.09 - 2021.08.11

Audit Team : SlowMist Security Team

Summary conclusion : This is a cross-chain token contract that contains the tokenVault section. The total amount of contract tokens can be changed. 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 issues:

1. Only the Auth role can mint tokens at will and there is no upper limit of the minted amount of tokens.
2. The Auth role can burn users' tokens at will through the burn function.
3. Only the Auth role can execute Swapin operation to mint tokens.
4. Only the vault role can execute the withdrawVault operation to withdraw the user's asset.

The source code:

```
// SPDX-License-Identifier: GPL-3.0-or-later
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity 0.8.2;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    function totalSupply() external view returns (uint256);
    function decimals() external view returns (uint8);
    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);
    function permit(address target, address spender, uint256 value, uint256 deadline,
uint8 v, bytes32 r, bytes32 s) external;
    function transferWithPermit(address target, address to, uint256 value, uint256
deadline, uint8 v, bytes32 r, bytes32 s) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

/**
 * @dev Interface of the ERC2612 standard as defined in the EIP.
 *
 * Adds the {permit} method, which can be used to change one's
 * {IERC20-allowance} without having to send a transaction, by signing a
 * message. This allows users to spend tokens without having to hold Ether.
 *
 * See https://eips.ethereum.org/EIPS/eip-2612.
 */
interface IERC2612 {

    /**
     * @dev Returns the current ERC2612 nonce for `owner`. This value must be
     * included whenever a signature is generated for {permit}.
     *
     * Every successful call to {permit} increases ``owner``'s nonce by one. This
     * prevents a signature from being used multiple times.
     */
    function nonces(address owner) external view returns (uint256);
}

/// @dev Wrapped ERC-20 v10 (AnyswapV3ERC20) is an ERC-20 ERC-20 wrapper. You can
`deposit` ERC-20 and obtain an AnyswapV3ERC20 balance which can then be operated as
an ERC-20 token. You can
/// `withdraw` ERC-20 from AnyswapV3ERC20, which will then burn AnyswapV3ERC20 token
in your wallet. The amount of AnyswapV3ERC20 token in any wallet is always identical
to the
/// balance of ERC-20 deposited minus the ERC-20 withdrawn with that specific wallet.
interface IAnyswapV3ERC20 is IERC20, IERC2612 {

    /// @dev Sets `value` as allowance of `spender` account over caller account's
AnyswapV3ERC20 token,
```

```
    /// after which a call is executed to an ERC677-compliant contract with the
`data` parameter.
    /// Emits {Approval} event.
    /// Returns boolean value indicating whether operation succeeded.
    /// For more information on approveAndCall format, see
https://github.com/ethereum/EIPs/issues/677.
    function approveAndCall(address spender, uint256 value, bytes calldata data)
external returns (bool);

    /// @dev Moves `value` AnyswapV3ERC20 token from caller's account to account
(`to`),
    /// after which a call is executed to an ERC677-compliant contract with the
`data` parameter.
    /// A transfer to `address(0)` triggers an ERC-20 withdraw matching the sent
AnyswapV3ERC20 token in favor of caller.
    /// Emits {Transfer} event.
    /// Returns boolean value indicating whether operation succeeded.
    /// Requirements:
    ///   - caller account must have at least `value` AnyswapV3ERC20 token.
    /// For more information on transferAndCall format, see
https://github.com/ethereum/EIPs/issues/677.
    function transferAndCall(address to, uint value, bytes calldata data) external
returns (bool);
}

interface ITransferReceiver {
    function onTokenTransfer(address, uint, bytes calldata) external returns (bool);
}

interface IApprovalReceiver {
    function onTokenApproval(address, uint, bytes calldata) external returns (bool);
}

library Address {
    function isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }
}

library SafeERC20 {
```

```
    using Address for address;

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

    function safeTransferFrom(IERC20 token, address from, address to, uint value)
internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector,
from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint value) internal {
        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 callOptionalReturn(IERC20 token, bytes memory data) private {
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

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

contract AnyswapV5ERC20 is IAnyswapV3ERC20 {
    using SafeERC20 for IERC20;
    string public name;
    string public symbol;
    uint8  public immutable override decimals;

    address public immutable underlying;

    bytes32 public constant PERMIT_TYPEHASH = keccak256("Permit(address owner,address
spender,uint256 value,uint256 nonce,uint256 deadline)");
```

```
    bytes32 public constant TRANSFER_TYPEHASH = keccak256("Transfer(address
owner,address to,uint256 value,uint256 nonce,uint256 deadline)");
    bytes32 public immutable DOMAIN_SEPARATOR;

    /// @dev Records amount of AnyswapV3ERC20 token owned by account.
    mapping (address => uint256) public override balanceOf;
    uint256 private _totalSupply;

    // init flag for setting immediate vault, needed for CREATE2 support
    bool private _init;

    // flag to enable/disable swapout vs vault.burn so multiple events are triggered
    bool private _vaultOnly;

    // configurable delay for timelock functions
    uint public delay = 2*24*3600;


    // set of minters, can be this bridge or other bridges
    mapping(address => bool) public isMinter;
    address[] public minters;

    // primary controller of the token contract
    address public vault;

    address public pendingMinter;
    uint public delayMinter;

    address public pendingVault;
    uint public delayVault;

    uint public pendingDelay;
    uint public delayDelay;


    modifier onlyAuth() {
        require(isMinter[msg.sender], "AnyswapV4ERC20: FORBIDDEN");
        _;
    }

    modifier onlyVault() {
        require(msg.sender == mpc(), "AnyswapV3ERC20: FORBIDDEN");
        _;
    }
```

```
function owner() public view returns (address) {
    return mpc();
}

function mpc() public view returns (address) {
    if (block.timestamp >= delayVault) {
        return pendingVault;
    }
    return vault;
}

function setVaultOnly(bool enabled) external onlyVault {
    _vaultOnly = enabled;
}

function initVault(address _vault) external onlyVault {
    require(_init);
    vault = _vault;
    pendingVault = _vault;
    isMinter[_vault] = true;
    minters.push(_vault);
    delayVault = block.timestamp;
    _init = false;
}
//SlowMist// Only the Vault role can set minter
function setMinter(address _auth) external onlyVault {
    pendingMinter = _auth;
    delayMinter = block.timestamp + delay;
}
//SlowMist// Only the Vault role can set a new vault
function setVault(address _vault) external onlyVault {
    pendingVault = _vault;
    delayVault = block.timestamp + delay;
}

function applyVault() external onlyVault {
    require(block.timestamp >= delayVault);
    vault = pendingVault;
}

function applyMinter() external onlyVault {
    require(block.timestamp >= delayMinter);
    isMinter[pendingMinter] = true;
    minters.push(pendingMinter);
}
```

```
    // No time delay revoke minter emergency function
    //SlowMist// Only the Vault role can revoke a minter role
    function revokeMinter(address _auth) external onlyVault {
        isMinter[_auth] = false;
    }

    function getAllMinters() external view returns (address[] memory) {
        return minters;
    }

    //SlowMist// Only the Vault role can change an old vault to a new one
    function changeVault(address newVault) external onlyVault returns (bool) {
        require(newVault != address(0), "AnyswapV3ERC20: address(0x0)");
        pendingVault = newVault;
        delayVault = block.timestamp + delay;
        emit LogChangeVault(vault, pendingVault, delayVault);
        return true;
    }
    //SlowMist// Only the Vault role can change the MPCOwner
    function changeMPCOwner(address newVault) public onlyVault returns (bool) {
        require(newVault != address(0), "AnyswapV3ERC20: address(0x0)");
        pendingVault = newVault;
        delayVault = block.timestamp + delay;
        emit LogChangeMPCOwner(vault, pendingVault, delayVault);
        return true;
    }
    //SlowMist// Only the Auth role can mint tokens at will and there is no upper
limit of the minted amount of tokens
    function mint(address to, uint256 amount) external onlyAuth returns (bool) {
        _mint(to, amount);
        return true;
    }
    //SlowMist// Only the Auth role can burn tokens through the burn function
    function burn(address from, uint256 amount) external onlyAuth returns (bool) {
        require(from != address(0), "AnyswapV3ERC20: address(0x0)");
        _burn(from, amount);
        return true;
    }
    //SlowMist// Only the Auth role can execute swapin operation
    function Swapin(bytes32 txhash, address account, uint256 amount) public onlyAuth
returns (bool) {
        _mint(account, amount);
        emit LogSwapin(txhash, account, amount);
        return true;
```

```
    }

    function Swapout(uint256 amount, address bindaddr) public returns (bool) {
        require(!_vaultOnly, "AnyswapV4ERC20: onlyAuth");
        require(bindaddr != address(0), "AnyswapV3ERC20: address(0x0)");
        _burn(msg.sender, amount);
        emit LogSwapout(msg.sender, bindaddr, amount);
        return true;
    }

    /// @dev Records current ERC2612 nonce for account. This value must be included
whenever signature is generated for {permit}.
    /// Every successful call to {permit} increases account's nonce by one. This
prevents signature from being used multiple times.
    mapping (address => uint256) public override nonces;

    /// @dev Records number of AnyswapV3ERC20 token that account (second) will be
allowed to spend on behalf of another account (first) through {transferFrom}.
    mapping (address => mapping (address => uint256)) public override allowance;

    event LogChangeVault(address indexed oldVault, address indexed newVault, uint
indexed effectiveTime);
    event LogChangeMPCOwner(address indexed oldOwner, address indexed newOwner, uint
indexed effectiveHeight);
    event LogSwapin(bytes32 indexed txhash, address indexed account, uint amount);
    event LogSwapout(address indexed account, address indexed bindaddr, uint amount);
    event LogAddAuth(address indexed auth, uint timestamp);

    constructor(string memory _name, string memory _symbol, uint8 _decimals, address
_underlying, address _vault) {
        name = _name;
        symbol = _symbol;
        decimals = _decimals;
        underlying = _underlying;
        if (_underlying != address(0x0)) {
            require(_decimals == IERC20(_underlying).decimals());
        }

        // Use init to allow for CREATE2 accross all chains
        _init = true;

        // Disable/Enable swapout for v1 tokens vs mint/burn for v3 tokens
        _vaultOnly = false;

        vault = _vault;
```

```
        pendingVault = _vault;
        delayVault = block.timestamp;

        uint256 chainId;
        assembly {chainId := chainid()}
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256("EIP712Domain(string name,string version,uint256
chainId,address verifyingContract)"),
                keccak256(bytes(name)),
                keccak256(bytes("1")),
                chainId,
                address(this)));
    }

    /// @dev Returns the total supply of AnyswapV3ERC20 token as the ETH held in this
contract.
    function totalSupply() external view override returns (uint256) {
        return _totalSupply;
    }

    function depositWithPermit(address target, uint256 value, uint256 deadline, uint8
v, bytes32 r, bytes32 s, address to) external returns (uint) {
        IERC20(underlying).permit(target, address(this), value, deadline, v, r, s);
        IERC20(underlying).safeTransferFrom(target, address(this), value);
        return _deposit(value, to);
    }

    function depositWithTransferPermit(address target, uint256 value, uint256
deadline, uint8 v, bytes32 r, bytes32 s, address to) external returns (uint) {
        IERC20(underlying).transferWithPermit(target, address(this), value, deadline,
v, r, s);
        return _deposit(value, to);
    }

    function deposit() external returns (uint) {
        uint _amount = IERC20(underlying).balanceOf(msg.sender);
        IERC20(underlying).safeTransferFrom(msg.sender, address(this), _amount);
        return _deposit(_amount, msg.sender);
    }

    function deposit(uint amount) external returns (uint) {
        IERC20(underlying).safeTransferFrom(msg.sender, address(this), amount);
        return _deposit(amount, msg.sender);
    }
```

```
    function deposit(uint amount, address to) external returns (uint) {
        IERC20(underlying).safeTransferFrom(msg.sender, address(this), amount);
        return _deposit(amount, to);
    }

    function depositVault(uint amount, address to) external onlyVault returns (uint)
{
        return _deposit(amount, to);
    }

    function _deposit(uint amount, address to) internal returns (uint) {
        require(underlying != address(0x0) && underlying != address(this));
        _mint(to, amount);
        return amount;
    }

    function withdraw() external returns (uint) {
        return _withdraw(msg.sender, balanceOf[msg.sender], msg.sender);
    }

    function withdraw(uint amount) external returns (uint) {
        return _withdraw(msg.sender, amount, msg.sender);
    }

    function withdraw(uint amount, address to) external returns (uint) {
        return _withdraw(msg.sender, amount, to);
    }
    //SlowMist// Only the vault role can execute the withdrawVault operation
    function withdrawVault(address from, uint amount, address to) external onlyVault
returns (uint) {
        return _withdraw(from, amount, to);
    }

    function _withdraw(address from, uint amount, address to) internal returns (uint)
{
        _burn(from, amount);
        IERC20(underlying).safeTransfer(to, amount);
        return 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 {
        require(account != address(0), "ERC20: mint to the zero address");

        _totalSupply += amount;
        balanceOf[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 {
        require(account != address(0), "ERC20: burn from the zero address");

        balanceOf[account] -= amount;
        _totalSupply -= amount;
        emit Transfer(account, address(0), amount);
    }

    /// @dev Sets `value` as allowance of `spender` account over caller account's
AnyswapV3ERC20 token.
    /// Emits {Approval} event.
    /// Returns boolean value indicating whether operation succeeded.
    function approve(address spender, uint256 value) external override returns (bool)
{
        // _approve(msg.sender, spender, value);
        allowance[msg.sender][spender] = value;
        emit Approval(msg.sender, spender, value);

        return true;
    }
```

```
    /// @dev Sets `value` as allowance of `spender` account over caller account's
AnyswapV3ERC20 token,
    /// after which a call is executed to an ERC677-compliant contract with the
`data` parameter.
    /// Emits {Approval} event.
    /// Returns boolean value indicating whether operation succeeded.
    /// For more information on approveAndCall format, see
https://github.com/ethereum/EIPs/issues/677.
    function approveAndCall(address spender, uint256 value, bytes calldata data)
external override returns (bool) {
        // _approve(msg.sender, spender, value);
        allowance[msg.sender][spender] = value;
        emit Approval(msg.sender, spender, value);

        return IApprovalReceiver(spender).onTokenApproval(msg.sender, value, data);
    }

    /// @dev Sets `value` as allowance of `spender` account over `owner` account's
AnyswapV3ERC20 token, given `owner` account's signed approval.
    /// Emits {Approval} event.
    /// Requirements:
    ///   - `deadline` must be timestamp in future.
    ///   - `v`, `r` and `s` must be valid `secp256k1` signature from `owner` account
over EIP712-formatted function arguments.
    ///   - the signature must use `owner` account's current nonce (see {nonces}).
    ///   - the signer cannot be zero address and must be `owner` account.
    /// For more information on signature format, see
https://eips.ethereum.org/EIPS/eip-2612#specification[relevant EIP section].
    /// AnyswapV3ERC20 token implementation adapted from
https://github.com/albertocuestacanada/ERC20Permit/blob/master/contracts/ERC20Permit.
sol.
    function permit(address target, address spender, uint256 value, uint256 deadline,
uint8 v, bytes32 r, bytes32 s) external override {
        require(block.timestamp <= deadline, "AnyswapV3ERC20: Expired permit");

        bytes32 hashStruct = keccak256(
            abi.encode(
                PERMIT_TYPEHASH,
                target,
                spender,
                value,
                nonces[target]++,
                deadline));

        require(verifyEIP712(target, hashStruct, v, r, s) ||
```

```
verifyPersonalSign(target, hashStruct, v, r, s));

        // _approve(owner, spender, value);
        allowance[target][spender] = value;
        emit Approval(target, spender, value);
    }

    function transferWithPermit(address target, address to, uint256 value, uint256
deadline, uint8 v, bytes32 r, bytes32 s) external override returns (bool) {
        require(block.timestamp <= deadline, "AnyswapV3ERC20: Expired permit");

        bytes32 hashStruct = keccak256(
            abi.encode(
                TRANSFER_TYPEHASH,
                target,
                to,
                value,
                nonces[target]++,
                deadline));

        require(verifyEIP712(target, hashStruct, v, r, s) ||
verifyPersonalSign(target, hashStruct, v, r, s));

        require(to != address(0) || to != address(this));

        uint256 balance = balanceOf[target];
        require(balance >= value, "AnyswapV3ERC20: transfer amount exceeds balance");

        balanceOf[target] = balance - value;
        balanceOf[to] += value;
        emit Transfer(target, to, value);

        return true;
    }

    function verifyEIP712(address target, bytes32 hashStruct, uint8 v, bytes32 r,
bytes32 s) internal view returns (bool) {
        bytes32 hash = keccak256(
            abi.encodePacked(
                "\x19\x01",
                DOMAIN_SEPARATOR,
                hashStruct));
        address signer = ecrecover(hash, v, r, s);
        return (signer != address(0) && signer == target);
    }
```

```
    function verifyPersonalSign(address target, bytes32 hashStruct, uint8 v, bytes32
r, bytes32 s) internal view returns (bool) {
        bytes32 hash = prefixed(hashStruct);
        address signer = ecrecover(hash, v, r, s);
        return (signer != address(0) && signer == target);
    }

    // Builds a prefixed hash to mimic the behavior of eth_sign.
    function prefixed(bytes32 hash) internal view returns (bytes32) {
        return keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32",
DOMAIN_SEPARATOR, hash));
    }

    /// @dev Moves `value` AnyswapV3ERC20 token from caller's account to account
(`to`).
    /// A transfer to `address(0)` triggers an ETH withdraw matching the sent
AnyswapV3ERC20 token in favor of caller.
    /// Emits {Transfer} event.
    /// Returns boolean value indicating whether operation succeeded.
    /// Requirements:
    ///   - caller account must have at least `value` AnyswapV3ERC20 token.
    function transfer(address to, uint256 value) external override returns (bool) {
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to the loss of token during transfer
        require(to != address(0) || to != address(this));
        uint256 balance = balanceOf[msg.sender];
        require(balance >= value, "AnyswapV3ERC20: transfer amount exceeds balance");

        balanceOf[msg.sender] = balance - value;
        balanceOf[to] += value;
        emit Transfer(msg.sender, to, value);
        //SlowMist// The return value conforms to the EIP20 specification
        return true;
    }

    /// @dev Moves `value` AnyswapV3ERC20 token from account (`from`) to account
(`to`) using allowance mechanism.
    /// `value` is then deducted from caller account's allowance, unless set to
`type(uint256).max`.
    /// A transfer to `address(0)` triggers an ETH withdraw matching the sent
AnyswapV3ERC20 token in favor of caller.
    /// Emits {Approval} event to reflect reduced allowance `value` for caller
account to spend from account (`from`),
    /// unless allowance is set to `type(uint256).max`
```

```
    /// Emits {Transfer} event.
    /// Returns boolean value indicating whether operation succeeded.
    /// Requirements:
    ///   - `from` account must have at least `value` balance of AnyswapV3ERC20
token.
    ///   - `from` account must have approved caller to spend at least `value` of
AnyswapV3ERC20 token, unless `from` and caller are the same account.
    function transferFrom(address from, address to, uint256 value) external override
returns (bool) {
        require(to != address(0) || to != address(this));
        if (from != msg.sender) {
            // _decreaseAllowance(from, msg.sender, value);
            uint256 allowed = allowance[from][msg.sender];
            if (allowed != type(uint256).max) {
                require(allowed >= value, "AnyswapV3ERC20: request exceeds
allowance");
                uint256 reduced = allowed - value;
                allowance[from][msg.sender] = reduced;
                emit Approval(from, msg.sender, reduced);
            }
        }

        uint256 balance = balanceOf[from];
        require(balance >= value, "AnyswapV3ERC20: transfer amount exceeds balance");

        balanceOf[from] = balance - value;
        balanceOf[to] += value;
        emit Transfer(from, to, value);
        //SlowMist// The return value conforms to the EIP20 specification
        return true;
    }

    /// @dev Moves `value` AnyswapV3ERC20 token from caller's account to account
(`to`),
    /// after which a call is executed to an ERC677-compliant contract with the
`data` parameter.
    /// A transfer to `address(0)` triggers an ETH withdraw matching the sent
AnyswapV3ERC20 token in favor of caller.
    /// Emits {Transfer} event.
    /// Returns boolean value indicating whether operation succeeded.
    /// Requirements:
    ///   - caller account must have at least `value` AnyswapV3ERC20 token.
    /// For more information on transferAndCall format, see
https://github.com/ethereum/EIPs/issues/677.
    function transferAndCall(address to, uint value, bytes calldata data) external
```

```
override returns (bool) {
        require(to != address(0) || to != address(this));

        uint256 balance = balanceOf[msg.sender];
        require(balance >= value, "AnyswapV3ERC20: transfer amount exceeds balance");

        balanceOf[msg.sender] = balance - value;
        balanceOf[to] += value;
        emit Transfer(msg.sender, to, value);

        return ITransferReceiver(to).onTokenTransfer(msg.sender, value, data);
    }
}
```

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