Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.10.18, the SlowMist security team received the HurricaneSwap team's security audit application for HurricaneSwap-v2-contract, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

Level	Description
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Short Address Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Uninitialized Storage Pointers Vulnerability
- Arithmetic Accuracy Deviation Vulnerability
- tx.origin Authentication Vulnerability

- "False top-up" Vulnerability
- Variable Coverage Vulnerability
- Gas Optimization Audit
- Malicious Event Log Audit
- Redundant Fallback Function Audit
- Unsafe External Call Audit
- Explicit Visibility of Functions State Variables Aduit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Audit Version:

Project address:

https://github.com/Caijiawen/HurricaneSwap-v2-contract

commit:

424c826ea144ad8ff83ba8580e98cabddb04d57e

Fixed Version:

Project address:

https://github.com/HurricaneSwap/v2-contract

commit:

4eb25ad188669297eaf0821ad92854d8712fdabe

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Possible replay risk	Replay Vulnerability	Suggestion	Confirmed
N2	Risk of excessive authority	Authority Control Vulnerability	Medium	Confirmed
N3	Missing event record	Others	Suggestion	Ignored
N4	GasToken attack	Others	Suggestion	Ignored
N5	Gas Optimization	Gas Optimization Audit	Suggestion	Fixed
N6	Arbitrage advice	Others	Suggestion	Confirmed
N7	Event capture recommendations	Others	Suggestion	Confirmed
N8	DoS issues	Others	Suggestion	Ignored

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

HcSwapAvaxERC20			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_mint	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-
_approve	Internal	Can Modify State	-
_transfer	Internal	Can Modify State	-
approve	External	Can Modify State	-
transfer	External	Can Modify State	-
transferFrom	External	Can Modify State	-
permit	External	Can Modify State	-

HcSwapAvaxFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
allPairsLength	External	-	-
createPair	External	Can Modify State	-
setFeeTo	External	Can Modify State	-
setFeeToSetter	External	Can Modify State	-
setOwner	External	Can Modify State	onlyOwner

HcSwapAvaxPair

HcSwapAvaxPair			
Function Name	Visibility	Mutability	Modifiers
getReserves	Public	-	-
_safeTransfer	Private	Can Modify State	-
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	-
setCrossPair	External	Can Modify State	onlyOwner
_update	Private	Can Modify State	-
_mintFee	Private	Can Modify State	-
mint	External	Can Modify State	onlyOwner lock
burnQuery	External	-	-
burn	External	Can Modify State	onlyOwner lock
swap	External	Can Modify State	onlyOwner lock
skim	External	Can Modify State	onlyOwner lock
sync	External	Can Modify State	onlyOwner lock

HcSwapAvaxRouter			
Function Name	Visibility	Mutability	Modifiers
isOperator	Public	-	-
setOwner	Public	Can Modify State	onlyOwner
setFactoryOwner	Public	Can Modify State	onlyOwner

HcSwapAvaxRouter			
setPause	Public	Can Modify State	onlyOwner
setCrossToken	Public	Can Modify State	onlyOwner
setBscToAvax	Public	Can Modify State	onlyOwner
setOperator	Public	Can Modify State	onlyOwner
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
initBSCToken	Internal	Can Modify State	-
CreateBSCCrossLiquidity	Public	Can Modify State	onlyOperator
onCrossTask	Public	Can Modify State	onlyOperator
onAddLPCrossTask	Internal	Can Modify State	-
onRemoveLPCrossTask	Internal	Can Modify State	-
onEmitCrossAction	Internal	Can Modify State	-
_mappingBSCTokenToAvax	Internal	-	-
_addLiquidity	Internal	Can Modify State	-
_addLiquidityNoRevert	Internal	-	-
addLiquidity	Public	Can Modify State	ensure whenNotPaused
addLiquidityETH	External	Payable	ensure whenNotPaused
removeLiquidity	Public	Can Modify State	ensure whenNotPaused
removeLiquidityETH	Public	Can Modify State	ensure whenNotPaused
_swap	Internal	Can Modify State	-

HcSwapAvaxRouter			
swapExactTokensForTokens	External	Can Modify State	ensure whenNotPaused
swapTokensForExactTokens	External	Can Modify State	ensure whenNotPaused
swapExactETHForTokens	External	Payable	ensure whenNotPaused
swapTokensForExactETH	External	Can Modify State	ensure whenNotPaused
swapExactTokensForETH	External	Can Modify State	ensure whenNotPaused
swapETHForExactTokens	External	Payable	ensure whenNotPaused
quote	Public	-	-

Pausable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Internal	Can Modify State	-
paused	Public	-	-
_pause	Internal	Can Modify State	whenNotPaused
_unpause	Internal	Can Modify State	whenPaused

HcSwapHelper			
Function Name	Visibility	Mutability	Modifiers
sqrt	Internal	-	-
calcMintFee	Public	-	-
calcReserve	Public	-	-
getReservesWithCross	Public	-	-

HcSwapHelper			
getReserves	Public	-	-
getAmountOutNoCross	Public	-	-
getAmountInNoCross	Public	-	-
getAmountOut	Public	-	-
getAmountIn	Public	-	-
getAmountsOut	Public	-	-
getAmountsIn	Public	-	-

HcToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	HcSwapAvaxERC20
transferOwnership	Public	Can Modify State	onlyOwner
setBlackList	Public	Can Modify State	onlyOwner
setMinter	Public	Can Modify State	onlyOwner
superMint	Public	Can Modify State	onlyMinter
superBurn	Public	Can Modify State	onlyMinter
burn	Public	Can Modify State	-
_transfer	Internal	Can Modify State	-

HcTokenFactory			
Function Name	Visibility	Mutability	Modifiers

HcTokenFactory			
<Constructor>	Public	Can Modify State	-
transferOwnership	Public	Can Modify State	onlyOwner
createAToken	External	Can Modify State	onlyOwner

HcSwapBSCPair			
Function Name	Visibility	Mutability	Modifiers
getReserves	Public	-	-
_safeTransfer	Private	Can Modify State	-
<Constructor>	Public	Can Modify State	-
owner	Public	-	-
initialize	External	Can Modify State	-
_update	Private	Can Modify State	-
_mintFee	Private	Can Modify State	-
mint	External	Can Modify State	onlyOwner lock
directlyMint	External	Can Modify State	onlyOwner lock
directlyBurn	External	Can Modify State	onlyOwner lock
directlySync	External	Can Modify State	onlyOwner lock
burn	External	Can Modify State	onlyOwner lock
swap	External	Can Modify State	onlyOwner lock
skim	External	Can Modify State	onlyOwner lock

HcSwapBSCPair			
sync	External	Can Modify State	onlyOwner lock
_sync	Internal	Can Modify State	-

HcSwapERC20			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_mint	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-
_approve	Private	Can Modify State	-
_transfer	Private	Can Modify State	-
approve	External	Can Modify State	-
transfer	External	Can Modify State	-
transferFrom	External	Can Modify State	-
permit	External	Can Modify State	-

HcSwapFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
allPairsLength	External	-	-
createPair	External	Can Modify State	onlyOwner
setFeeTo	External	Can Modify State	-

HcSwapFactory			
setFeeToSetter	External	Can Modify State	-
setOwner	External	Can Modify State	onlyOwner

HcSwapV2Router02			
Function Name	Visibility	Mutability	Modifiers
getTask	Public	-	-
isOperator	Public	-	-
setOwner	Public	Can Modify State	onlyOwner
setFactoryOwner	Public	Can Modify State	onlyOwner
setPause	Public	Can Modify State	onlyOwner
setOperator	Public	Can Modify State	onlyOwner
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
requireMinAmount	Public	-	-
setMinAmount	Public	Can Modify State	onlyOwner
onCrossSync	External	Can Modify State	onlyOperator
_addLiquidity	Internal	Can Modify State	-
addLiquidityFromUser	External	Can Modify State	ensure lock whenNotPaused
addLiquidity	External	Can Modify State	ensure onlyOperator
addLiquidityETH	External	Payable	ensure onlyOperator

HcSwapV2Router02			
removeLiquidity	Public	Can Modify State	ensure onlyOperator
removeLiquidityFromUser	External	Can Modify State	ensure lock whenNotPaused
removeLiquidityETH	Public	Can Modify State	ensure onlyOperator
swapExactTokensForTokens	External	Can Modify State	ensure onlyOperator
swapTokensForExactTokens	External	Can Modify State	ensure onlyOperator
swapExactETHForTokens	External	Payable	ensure onlyOperator
swapTokensForExactETH	External	Can Modify State	ensure onlyOperator
swapExactTokensForETH	External	Can Modify State	ensure onlyOperator
swapETHForExactTokens	External	Payable	ensure onlyOperator
_swap	Internal	Can Modify State	-
quote	Public	-	-
getAmountOut	Public	-	-
getAmountIn	Public	-	-
getAmountsOut	Public	-	-
getAmountsIn	Public	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Possible replay risk

Category: Replay Vulnerability

Content

DOMAIN_SEPARATOR is defined when the contract is initialized, but it will not be reimplemented when DOMAIN_SEPARATOR is used in the permit function. DOMAIN_SEPARATOR contains chainId and is defined when the contract is deployed instead of rebuilding for each signature. In the case of future chain forks, there may be a risk of replay attacks between chains.

code location:contracts/avax/HcSwapAvaxERC20.sol #L24-L38

```
    constructor() public {
        uint 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)
            )
        );
    }
```

code location:contracts/avax/HcSwapAvaxERC20.sol #L81-L93

```
    function permit(address owner, address spender, uint value, uint deadline, uint8
v, bytes32 r, bytes32 s) external {
        require(deadline >= block.timestamp, 'HcSwap: EXPIRED');
        bytes32 digest = keccak256(
            abi.encodePacked(
                '\x19\x01',
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value,
nonces[owner]++, deadline))
            )
        );
        address recoveredAddress = ecrecover(digest, v, r, s);
        require(recoveredAddress != address(0) && recoveredAddress == owner, 'HcSwap:
INVALID_SIGNATURE');
```

```
        _approve(owner, spender, value);
    }
```

code location:contracts/bsc/HcSwapERC20.sol #L24-L38

```
    constructor() public {
        uint 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)
            )
        );
    }
```

code location:contracts/bsc/HcSwapERC20.sol #L81-L93

```
    function permit(address owner, address spender, uint value, uint deadline, uint8
v, bytes32 r, bytes32 s) external {
        require(deadline >= block.timestamp, 'HcSwap: EXPIRED');
        bytes32 digest = keccak256(
            abi.encodePacked(
                '\x19\x01',
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value,
nonces[owner]++, deadline))
            )
        );
        address recoveredAddress = ecrecover(digest, v, r, s);
        require(recoveredAddress != address(0) && recoveredAddress == owner, 'HcSwap:
INVALID_SIGNATURE');
        _approve(owner, spender, value);
    }
```

Solution

It is recommended to dynamically obtain the chainId parameter in the constructor whenever the permit function is executed.

Reference: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-2612.md

Status

Confirmed

[N2] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

Minter characters can mint arbitrarily through the supermint function, and there is no upper limit on the number of minted coins. The Minter role can also use superburn to destroy the user's tokens. And onlyMinter allows the owner role to directly call these two supermint and superburn functions to mint and burn coins for any user.

code location:contracts/avax/HcToken.sol #L21-L24

```
    modifier onlyMinter() {
        require(msg.sender == owner || minter[msg.sender] == true,
string(abi.encodePacked("HcToken ", name, ":ONLY_MINTER")));
        _;
    }
```

code location:contracts/avax/HcToken.sol #L58-L64

```
    function superMint(address to_, uint256 amount_) onlyMinter public {
        _mint(to_, amount_);
    }

    function superBurn(address account_, uint256 amount_) onlyMinter public {
        _burn(account_, amount_);
    }
```

The owner can set a blacklist, and the address set as the blacklist cannot transfer. However, if ordinary users become blacklisted after deposit, they will not be able to withdraw their assets, so there is a risk of Admin having too much authority.

code location:contracts/avax/HcToken.sol #L42-L48

```
    function setBlackList(address[] memory addresses_, bool[] memory status_)
onlyOwner public {
        require(addresses_.length == status_.length, "HcToken::setBlackList
WRONG_DATA");
        for (uint i = 0; i < addresses_.length; i++) {
            blackList[addresses_[i]] = status_[i];
            emit SetBlackList(addresses_[i], status_[i]);
        }
    }
```

Solution

It is recommended to transfer the authority to the community or governance contract.

Status

Confirmed; The permissions will be transferred to Timelock for management by the project team. Owner permissions will have been removed in onlyMinter.

[N3] [Suggestion] Missing event record

Category: Others

Content

There is no event record for modifying contract parameters, which is not conducive to the review of community users.

code location:constracts/avax/HcSwapAvaxFactory.sol #L47-L60

```
    function setFeeTo(address _feeTo) external {
        require(msg.sender == feeToSetter, 'HcSwap: FORBIDDEN');
        feeTo = _feeTo;
    }
```

```
function setFeeToSetter(address _feeToSetter) external {
    require(msg.sender == feeToSetter, 'HcSwap: FORBIDDEN');
    feeToSetter = _feeToSetter;
}

function setOwner(address _owner) external onlyOwner{
    owner = _owner;
}
```

code location:constracts/avax/HcSwapAvaxPair.sol #L76-L78

```
function setCrossPair(bool status_) external onlyOwner {
    crossPair = status_;
}
```

code location:constracts/avax/HcSwapAvaxRouter.sol #L76-L84

```
function setOwner(address _owner) onlyOwner public {
    owner = _owner;
}

function setFactoryOwner(address _owner) onlyOwner public {
    if (IHcSwapAvaxFactory(factory).owner() == address(this)) {
        IHcSwapAvaxFactory(factory).setOwner(_owner);
    }
}
```

code location:constracts/bsc/HcSwapFactory.sol #L47-L59

```
function setFeeTo(address _feeTo) external {
    require(msg.sender == feeToSetter, 'HcSwap: FORBIDDEN');
    feeTo = _feeTo;
}

function setFeeToSetter(address _feeToSetter) external {
    require(msg.sender == feeToSetter, 'HcSwap: FORBIDDEN');
    feeToSetter = _feeToSetter;
}

function setOwner(address _owner) external onlyOwner{
```

```
        owner = _owner;
    }
```

code location:constracts/bsc/HcSwapV2Router02.sol #L76-L84

```
    function setOwner(address _owner) onlyOwner public {
        owner = _owner;
    }

    function setFactoryOwner(address _owner) onlyOwner public {
        if(IHcSwapBSCFactory(factory).owner() == address(this)){
            IHcSwapBSCFactory(factory).setOwner(_owner);
        }
    }
```

Solution

It is recommended to record events when modifying sensitive parameters. _feeto address recommends using a multi-signature contract to avoid the disclosure of private keys and the theft of team income.

The setOwner function can change the owner's address. If the _owner address is incorrectly passed in, the permissions will be lost. It is recommended to add the dependingOwner operation for a second confirmation operation, and then set it after confirmation. This can prevent the loss of permissions due to misinformation of addresses.

Status

Ignored

[N4] [Suggestion] GasToken attack

Category: Others

Content

The safeTransferETH function does not limit the gaslimit of the call. If the to address is a third-party address entered by the user, there may be a gas token attack.

code location:contracts/avax/TransferHelper.sol #L47-#50

```
    function safeTransferETH(address to, uint256 value) internal {
        (bool success, ) = to.call{value: value}(new bytes(0));
        require(success, 'TransferHelper::safeTransferETH: ETH transfer failed');
    }
```

Solution

It is recommended to limit the call gaslimit.

Reference: https://floriantramer.com/docs/slides/CESC18gastoken.pdf

Status

Ignored

[N5] [Suggestion] Gas Optimization

Category: Gas Optimization Audit

Content

It is recommended to change assert to require to optimize gas, so as to avoid using up the remaining gas in the transaction after assert. The specific location is in //SlowMist// comments.

code location:contracts/avax/HcSwapAvaxRouter.sol #L110-L128

```
    function setOperator(address[] memory _ops, bool[] memory _status) onlyOwner
public {
        require(_ops.length == _status.length,
"HcSwapV2Router:SET_OPERATOR_WRONG_DATA");
        for (uint i = 0; i < _ops.length; i++) {
            operator[_ops[i]] = _status[i];
        }
    }

    constructor(address _factory, address _WETH, address _tokenFactory) public {
        factory = _factory;
        WETH = _WETH;
        tokenFactory = _tokenFactory;
        owner = msg.sender;
        tasksIndex = 1;
    }
```

```
    receive() external payable {
//SlowMist// Here assert is used for assertion judgment
        assert(msg.sender == WETH);
        // only accept ETH via fallback from the WETH contract
    }
```

code location:contracts/avax/HcSwapAvaxRouter.sol #L263-L296

```
    function _addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin
    ) internal virtual returns (uint amountA, uint amountB) {
        address pair = IUniswapV2Factory(factory).getPair(tokenA, tokenB);
        // create the pair if it doesn't exist yet
        if (pair == address(0)) {
            require(!(crossToken[tokenA] && crossToken[tokenB]),
"HcSwapV2Router::_addLiquidity CROSS_TOKEN_NOT_ALLOW_CREATE");
            pair = IUniswapV2Factory(factory).createPair(tokenA, tokenB);
        }
        if (IHcSwapAvaxPair(pair).crossPair()) {
            require(isOperator(), "HcSwapV2Router::_addLiquidity
ONLY_OP_CAN_ADD_CROSS_LP");
        }

        (uint reserveA, uint reserveB) = UniswapV2Library.getReserves(factory,
tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint amountBOptimal = UniswapV2Library.quote(amountADesired, reserveA,
reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(amountBOptimal >= amountBMin, 'HcSwapV2Router:
INSUFFICIENT_B_AMOUNT');
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint amountAOptimal = UniswapV2Library.quote(amountBDesired,
reserveB, reserveA);
//SlowMist// Here assert is used for assertion judgment
                assert(amountAOptimal <= amountADesired);
```

```
                require(amountAOptimal >= amountAMin, 'HcSwapV2Router:
INSUFFICIENT_A_AMOUNT');
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }
```

code location:contracts/avax/HcSwapAvaxRouter.sol #L350-L373

```
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external virtual override payable ensure(deadline) whenNotPaused returns (uint
amountToken, uint amountETH, uint liquidity) {
        (amountToken, amountETH) = _addLiquidity(
            token,
            WETH,
            amountTokenDesired,
            msg.value,
            amountTokenMin,
            amountETHMin
        );
        address pair = UniswapV2Library.pairFor(factory, token, WETH);
        TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
        IWETH(WETH).deposit{value : amountETH}();
//SlowMist// Here assert is used for assertion judgment
        assert(IWETH(WETH).transfer(pair, amountETH));
        liquidity = IUniswapV2Pair(pair).mint(to);
        // refund dust eth, if any
        if (msg.value > amountETH) TransferHelper.safeTransferETH(msg.sender,
msg.value - amountETH);
    }
```

code location:contracts/avax/HcSwapAvaxRouter.sol #L462-L477

```
    function swapExactETHForTokens(uint amountOutMin, address[] calldata path,
address to, uint deadline)
    external
```

```
    virtual
    override
    payable
    ensure(deadline)
    whenNotPaused
    returns (uint[] memory amounts)
    {
        require(path[0] == WETH, 'HcSwapV2Router: INVALID_PATH');
        amounts = UniswapV2Library.getAmountsOut(factory, msg.value, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'HcSwapV2Router:
INSUFFICIENT_OUTPUT_AMOUNT');
        IWETH(WETH).deposit{value : amounts[0]}();
//SlowMist// Here assert is used for assertion judgment
        assert(IWETH(WETH).transfer(UniswapV2Library.pairFor(factory, path[0],
path[1]), amounts[0]));
        _swap(amounts, path, to);
    }
```

code location:contracts/avax/HcSwapAvaxRouter.sol #L517-L534

```
    function swapETHForExactTokens(uint amountOut, address[] calldata path, address
to, uint deadline)
    external
    virtual
    override
    payable
    ensure(deadline)
    whenNotPaused
    returns (uint[] memory amounts)
    {
        require(path[0] == WETH, 'HcSwapV2Router: INVALID_PATH');
        amounts = UniswapV2Library.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= msg.value, 'HcSwapV2Router: EXCESSIVE_INPUT_AMOUNT');
        IWETH(WETH).deposit{value : amounts[0]}();
//SlowMist// Here assert is used for assertion judgment
        assert(IWETH(WETH).transfer(UniswapV2Library.pairFor(factory, path[0],
path[1]), amounts[0]));
        _swap(amounts, path, to);
        // refund dust eth, if any
        if (msg.value > amounts[0]) TransferHelper.safeTransferETH(msg.sender,
msg.value - amounts[0]);
    }
```

code location:contracts/bsc/HcSwapV2Router.sol #L96-L113

```
    function setOperator(address[] memory _ops, bool[] memory _status) onlyOwner
public {
        require(_ops.length == _status.length,
"HcSwapV2Router:SET_OPERATOR_WRONG_DATA");
        for (uint i = 0; i < _ops.length; i++) {
            operator[_ops[i]] = _status[i];
        }
    }

    constructor(address _factory, address _WETH) public {
        factory = _factory;
        WETH = _WETH;
        owner = msg.sender;
        tasks.initStorage();
    }

    receive() external payable {
//SlowMist// Here assert is used for assertion judgment
        assert(msg.sender == WETH);
        // only accept ETH via fallback from the WETH contract
    }
```

code location:contracts/bsc/HcSwapV2Router.sol #L185-L215

```
    function _addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin
    ) internal virtual returns (uint amountA, uint amountB) {
        // create the pair if it doesn't exist yet
        if (IHcSwapBSCFactory(factory).getPair(tokenA, tokenB) == address(0)) {
            require(isOperator(msg.sender), "HcSwapBSC:NOT_OPERATOR");
            IHcSwapBSCPair pair =
IHcSwapBSCPair(IHcSwapBSCFactory(factory).createPair(tokenA, tokenB));
            (uint amount0,uint amount1) = pair.token0() == tokenA ? (amountADesired,
amountBDesired) : (amountBDesired, amountADesired);
            emit CreateCrossLP(address(pair), pair.token0(), pair.token1(), amount0,
amount1);
```

```
        }
        (uint reserveA, uint reserveB) = UniswapV2Library.getReserves(factory,
tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint amountBOptimal = UniswapV2Library.quote(amountADesired, reserveA,
reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(amountBOptimal >= amountBMin, 'HcSwapV2Router:
INSUFFICIENT_B_AMOUNT');
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint amountAOptimal = UniswapV2Library.quote(amountBDesired,
reserveB, reserveA);
//SlowMist// Here assert is used for assertion judgment
                assert(amountAOptimal <= amountADesired);
                require(amountAOptimal >= amountAMin, 'HcSwapV2Router:
INSUFFICIENT_A_AMOUNT');
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }
```

code location:contracts/bsc/HcSwapV2Router.sol #L257-L280

```
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external virtual override payable ensure(deadline) onlyOperator returns (uint
amountToken, uint amountETH, uint liquidity) {
        (amountToken, amountETH) = _addLiquidity(
            token,
            WETH,
            amountTokenDesired,
            msg.value,
            amountTokenMin,
            amountETHMin
        );
        address pair = UniswapV2Library.pairFor(factory, token, WETH);
```

```
        TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
        IWETH(WETH).deposit{value : amountETH}();
//SlowMist// Here assert is used for assertion judgment
        assert(IWETH(WETH).transfer(pair, amountETH));
        liquidity = IUniswapV2Pair(pair).mint(to);
        // refund dust eth, if any
        if (msg.value > amountETH) TransferHelper.safeTransferETH(msg.sender,
msg.value - amountETH);
    }
```

code location:contracts/bsc/HcSwapV2Router.sol #L375-L390

```
    function swapExactETHForTokens(uint amountOutMin, address[] calldata path,
address to, uint deadline)
    external
    virtual
    override
    payable
    ensure(deadline)
    onlyOperator
    returns (uint[] memory amounts)
    {
        require(path[0] == WETH, 'HcSwapV2Router: INVALID_PATH');
        amounts = UniswapV2Library.getAmountsOut(factory, msg.value, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'HcSwapV2Router:
INSUFFICIENT_OUTPUT_AMOUNT');
        IWETH(WETH).deposit{value : amounts[0]}();
//SlowMist// Here assert is used for assertion judgment
        assert(IWETH(WETH).transfer(UniswapV2Library.pairFor(factory, path[0],
path[1]), amounts[0]));
        _swap(amounts, path, to);
    }
```

code location:contracts/bsc/HcSwapV2Router.sol #L430-L447

```
    function swapETHForExactTokens(uint amountOut, address[] calldata path, address
to, uint deadline)
    external
    virtual
    override
    payable
    ensure(deadline)
```

```
    onlyOperator
    returns (uint[] memory amounts)
    {
        require(path[0] == WETH, 'HcSwapV2Router: INVALID_PATH');
        amounts = UniswapV2Library.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= msg.value, 'HcSwapV2Router: EXCESSIVE_INPUT_AMOUNT');
        IWETH(WETH).deposit{value : amounts[0]}();
//SlowMist// Here assert is used for assertion judgment
        assert(IWETH(WETH).transfer(UniswapV2Library.pairFor(factory, path[0],
path[1]), amounts[0]));
        _swap(amounts, path, to);
        // refund dust eth, if any
        if (msg.value > amounts[0]) TransferHelper.safeTransferETH(msg.sender,
msg.value - amounts[0]);
    }
```

Solution

It is recommended to change assert to require to optimize gas.

Status

Fixed; The project team has changed all assert to require.

[N6] [Suggestion] Arbitrage advice

Category: Others

Content

After communication, it was found that the project team had an arbitrage operation. If you want to open it up to allow users to arbitrage, you are worried that attackers can use the feature of the longest chain to maliciously arbitrage. Because the data of the two chains are not confirmed without confirming the longest chain, they are worried that they will be used for evil.

Solution

Through communication, the project team claimed that the arbitrage operation currently is operated by the project team. However, this will not completely eliminate the risk, because there is a confirmation number, so there is still the risk of being preemptively arbitrage by others.

Status

Confirmed

[N7] [Suggestion] Event capture recommendations

Category: Others

Content

After communication, it is found that the project team uses Relayer A to capture events to call the contract, so it should be noted that the events captured by Relayer A must be bound to its own contract to prevent events from other contracts from being captured.

Solution

Because I did not audit the code of Relayer A, the project team needs to confirm whether the code of Relayer A can achieve normal functions.

Status

Confirmed

[N8] [Suggestion] DoS issues

Category: Others

Content

The for loop does not have a maximum length limit, so when the number of loops is too large, it will cause outgas and then revert, causing the gas consumed by the previous operation to be wasted.

code location:contracts/bsc/HcSwapV2Router02.sol #L127-L182

```
    function onCrossSync(CrossAction[] calldata actions) external onlyOperator {
        for (uint256 i = 0; i < actions.length; i++) {
            CrossAction memory action = actions[i];
            IHcSwapBSCPair pair = IHcSwapBSCPair(UniswapV2Library.pairFor(factory,
action.tokenA, action.tokenB));

            (address token0,address token1) = (pair.token0(), pair.token1());
            (uint256 amount0,uint256 amount1) = action.tokenA == pair.token0() ?
```

```
(action.amountA, action.amountB) : (action.amountB, action.amountA);
            if (action.actionType == 0) {//sync
                if (IERC20(token0).balanceOf(address(pair)) < amount0) {
                    TransferHelper.safeTransferFrom(token0, msg.sender,
address(pair), amount0.sub(IERC20(token0).balanceOf(address(pair))));
                }
                if (IERC20(token1).balanceOf(address(pair)) < amount1) {
                    TransferHelper.safeTransferFrom(token1, msg.sender,
address(pair), amount1.sub(IERC20(token1).balanceOf(address(pair))));
                }
                pair.directlySync(amount0, amount1);
            } else {// mint lp
                LPQueue.LPAction storage task = tasks.readFirst();
                emit CrossTaskDone(tasks.currentIndex(), action.actionType == 1,
action.success);
                if (action.actionType == 1) {
                    require(action.checksum == task.checksum,
"HcSwap:CHECKSUM_ERROR");
                    (,,uint amountADesired,uint amountBDesired,,,,) =
LPQueue.decodeAddLP(task.payload);
                    if (action.success) {
                        require(action.liquidity > 0, "HcSwap:ZERO_LIQUIDITY");
                        TransferHelper.safeTransfer(action.tokenA, address(pair),
action.amountA);
                        TransferHelper.safeTransfer(action.tokenB, address(pair),
action.amountB);

                        if (amountADesired > action.amountA) {
                            TransferHelper.safeTransfer(action.tokenA, task.to,
amountADesired.sub(action.amountA));
                        }
                        if (amountBDesired > action.amountB) {
                            TransferHelper.safeTransfer(action.tokenB, task.to,
amountBDesired.sub(action.amountB));
                        }

                        pair.directlyMint(action.liquidity, task.to);
                    } else {
                        TransferHelper.safeTransfer(action.tokenA, task.to,
amountADesired);
                        TransferHelper.safeTransfer(action.tokenB, task.to,
amountBDesired);
                    }
                    tasks.dequeue();
                } else if (action.actionType == 2) {
```

```
                    require(action.checksum == task.checksum, "HcSwap:
INVALID_CHECKSUM");
                    //tokenA,tokenB,liquidity,amountAMin,amountBMin,to,deadline
                    (,,uint liquidity,,,,) = LPQueue.decodeRemoveLP(task.payload);
                    require(action.liquidity == liquidity, "HcSwap:
INVALID_LIQUIDITY");
                    if (action.success) {
                        pair.directlyBurn(action.liquidity, address(this), task.to,
amount0, amount1);
                    } else {
                        TransferHelper.safeTransfer(address(pair), task.to,
action.liquidity);
                    }
                    tasks.dequeue();
                } else {
                    revert('HcSwap:UNKNOWN_TYPE');
                }
            }
        }
    }
```

Solution

It is recommended to limit the maximum number of times, or there must be a limit on the chain to avoid wasting gas after DoS.

Status

Ignored; After communication, the project team stated that offline predicted gas is used here to determine whether the transaction is available, and the maximum offline limit is 40.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002110270001	SlowMist Security Team	2021.10.18 - 2021.10.27	Medium Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 medium risk, 7 suggestion vulnerabilities. And 1 medium risk, 3 suggestion vulnerabilities were confirmed and being fixed; 3 suggestion vulnerabilities were ignored; All other findings were fixed. The code was not deployed to the mainnet.

6 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