Btswap

Smart Contract Security Audit

2020-08-21

1. Executive Summary....3
2 Project Background(Context)....5
2.1 Project Introduction....5
2.2 Project Structure....6
2.3 Contract Structure....8
3 Audit Result....9
3.1 Critical Vulnerabilities....9
3.1.1 Process bypassing....9
3.2 High Risk Vulnerabilities....11
3.3 Medium Risk Vulnerabilities....11
3.3.1 Function flaw....11
3.4 Low Risk Vulnerabilities....14
3.4.1 Missing check....14
3.4.2 Process bypassing....16
3.5 Enhancement Suggestion....17
3.5.1 Accidentally obtain tokens....17
3.5.2 Risk control of fake tokens....17
4 Audit conclusion....18
4.1 Critical Vulnerabilities....18
4.2 Medium Risk vulnerabilities....18
4.3 Low Risk vulnerabilities....18

4.4 Enhancement Suggestion 18

4.5 Conclusion 18

5 Statement 19

1. Executive Summary

On Aug. 12, 2020, the SlowMist security team received the btswap team's security audit application for Btswap system, 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.

SlowMist Smart Contract DeFi project test method:

Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code module 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 wether there are vulnerabilities in programs suck as nodes, SDK, etc.

SlowMist Smart Contract DeFi project risk level:

Critical vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.

High-risk vulnerabilities	High-risk vulnerabilities will affect the normal operation of DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium-risk vulnerablities	Medium vulnerability will affect the operation of DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low-risk vulnerabilities	Low-risk vulnerabilities may affect the operation of DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weaknesses	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Enhancement Suggestions	There are better practices for coding or architecture.

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 public and in-house 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 attack and other Race Conditions
- Replay attack
- Reordering attack
- Short address attack
- Denial of service attack

- Transaction Ordering Dependence attack
- Conditional Completion attack
- Authority Control attack
- Integer Overflow and Underflow attack
- TimeStamp Dependence attack
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Explicit visibility of functions state variables
- Logic Flaws
- Uninitialized Storage Pointers
- Floating Points and Numerical Precision
- tx.origin Authentication
- "False top-up" Vulnerability
- Scoping and Declarations

2 Project Background(Context)

2.1 Project Introduction

BtSwap is an automatic market-making decentralized exchange based on the concept of fund pools. It is similar in function to some DEX on the market, but on this basis, it adds the element of liquidity mining. In other words, as long as a transaction is performed, a certain amount of Token will be generated and sent to the transaction party.

We audited Btswap's smart contract code, the following is the relevant file information:

Audit version file information

File name：btswap 审计资料.zip

SHA256: 1b65ce06b180aa12fdaa05fca4e31fe46faaccd82c3b569ff7898c3cb8ccc07f

Fixed version file information

File name：btswap-master.zip

SHA256: 065a1ad01acac3fb7dd2d16b7afbf8642e901a5c120c3a856ca217fb3af51fe9

File name: solidity-common-master.zip

SHA256: 22dbf53450c2931d5b643dccab6e9e520be21036c1c66f1cf7b42a912c893dc2

The project includes the following smart contract files:

2.2 Project Structure

Btswap-master

```
contracts
├── Migrations.sol
├── biz
│   ├── BtswapERC20.sol
│   ├── BtswapETH.sol
│   ├── BtswapFactory.sol
│   ├── BtswapPairToken.sol
│   ├── BtswapRouter.sol
│   └── BtswapToken.sol
├── interface
│   ├── IBtswapCallee.sol
│   ├── IBtswapERC20.sol
│   ├── IBtswapETH.sol
│   ├── IBtswapFactory.sol
│   ├── IBtswapPairToken.sol
│   ├── IBtswapRouter02.sol
│   ├── IBtswapToken.sol
│   └── IBtswapWhitelistedRole.sol
├── library
│   ├── TransferHelper.sol
│   └── UQ112x112.sol
└── vendor
```

```
    ├── DAI.sol
    ├── SAN.sol
    ├── SNX.sol
    └── USDT.sol
```

solidity-common

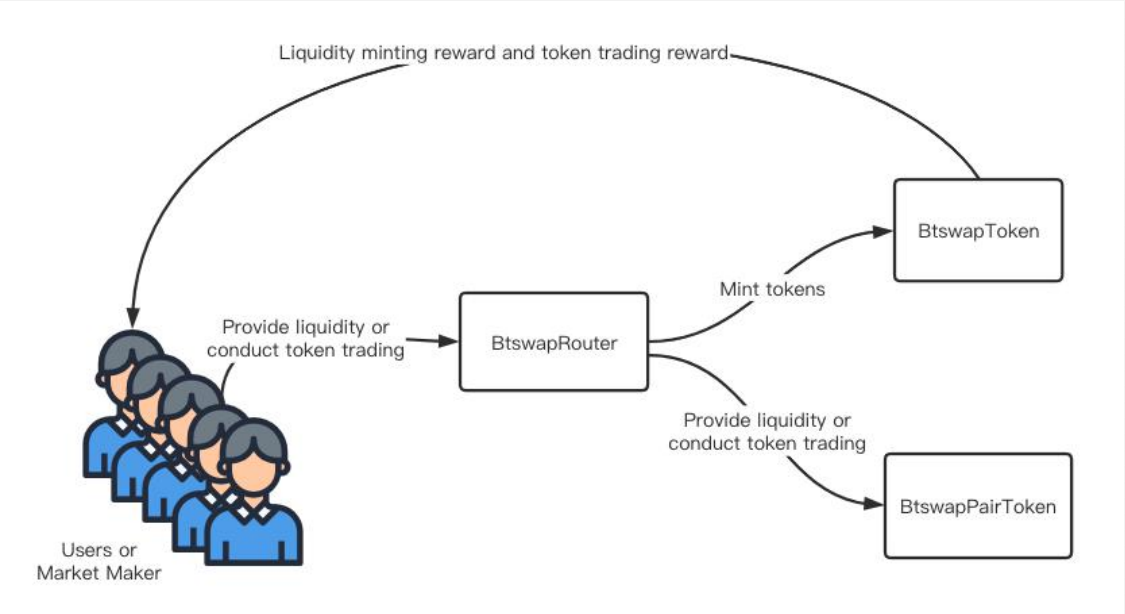
```
contracts
├── Migrations.sol
├── access
│   ├── BlacklistedRole.sol
│   ├── MinterRole.sol
│   ├── PauserRole.sol
│   └── WhitelistedRole.sol
├── common
│   ├── AddressResolver.sol
│   ├── Airdrop.sol
│   ├── ApprovalStorage.sol
│   ├── DailyLimit.sol
│   ├── Deadline.sol
│   ├── Destructible.sol
│   ├── EIP712.sol
│   ├── Logger.sol
│   ├── MappingStorage.sol
│   ├── Ownable.sol
│   ├── Pausable.sol
│   ├── Proxy.sol
│   ├── Proxyable.sol
│   ├── ReentrancyGuard.sol
│   ├── Resolver.sol
│   └── Storage.sol
├── erc20
│   ├── ERC20.sol
│   ├── ERC20Proxy.sol
│   ├── ERC20Proxyable.sol
│   └── ERC20Storage.sol
├── interface
│   └── IERC20.sol
├── library
│   ├── Address.sol
│   ├── Array.sol
│   ├── Roles.sol
```

```
|   ├── SafeERC20.sol
|   ├── SafeMath.sol
|   └── SignedSafeMath.sol
└── test
    ├── access
    |   ├── TestBlacklistedRole.sol
    |   ├── TestMinterRole.sol
    |   ├── TestPauserRole.sol
    |   └── TestWhitelistedRole.sol
    ├── common
    |   ├── TestAddressResolver.sol
    |   ├── TestDailyLimit.sol
    |   ├── TestDeadline.sol
    |   ├── TestLogger.sol
    |   ├── TestOwnable.sol
    |   ├── TestPausable.sol
    |   ├── TestProxy.sol
    |   ├── TestProxyable.sol
    |   ├── TestReentrancyGuard.sol
    |   └── TestStorage.sol
    ├── erc20
    |   ├── TestERC20.sol
    |   └── TestERC20Proxyable.sol
    └── library
        ├── TestAddress.sol
        ├── TestArray.sol
        ├── TestRoles.sol
        ├── TestSafeERC20.sol
        ├── TestSafeMath.sol
        └── TestSignedSafeMath.sol
```

2.3 Contract Structure

Btswap constructs a token exchange protocol based on the mathematical model of constant product, and any user can create a token pair for any white list token through the BtswapFactory contract. In addition, Btswap builds a layer of routing protocol over the exchange protocol. Users can conduct token trading and add liquidity through the BtswapRouter contract, participating in liquidity mining and transaction mining at the same time,

and obtain the tokens of Btswap platform. The overall structure of the contract is as follows:



3 Audit Result

3.1 Critical Vulnerabilities

Critical severity issues can have a major impact on the security of smart contracts, and it is highly recommended to fix critical severity vulnerability.

3.1.1 Process bypassing

(1) When using the BtswapRouter contract to add liquidity, the addLiquidity function is used to add liquidity and generate the corresponding share, but the removable liquidity can be directly burned on Btswappair, bypassing the BtswapRouter, taking liquidity back but continuing to maintain the BtswapToken reward.

```
function addLiquidity(
```

```
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    ) external ensure(deadline) returns (uint256 amountA, uint256 amountB, uint256 liquidity) {
        (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired, amountBDesired, amountAMin,
amountBMin);
        address pair = pairFor(tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IBtswapPair(pair).mint(to);
        IBtswapToken(BT).liquidity(msg.sender, pair);
    }
```

```
function burn(address to) external lock returns (uint256 amount0, uint256 amount1) {
        // gas savings
        (uint112 _reserve0, uint112 _reserve1,) = getReserves();
        // gas savings
        address _token0 = token0;
        // gas savings
        address _token1 = token1;
        uint256 balance0 = IERC20(_token0).balanceOf(address(this));
        uint256 balance1 = IERC20(_token1).balanceOf(address(this));
        uint256 liquidity = balanceOf[address(this)];

        bool feeOn = _mintFee(_reserve0, _reserve1);
        // gas savings, must be defined here since totalSupply can update in _mintFee
        uint256 _totalSupply = totalSupply;
        // using balances ensures pro-rata distribution
        amount0 = liquidity.mul(balance0) / _totalSupply;
        // using balances ensures pro-rata distribution
        amount1 = liquidity.mul(balance1) / _totalSupply;
        require(amount0 > 0 && amount1 > 0, 'BtswapPair: INSUFFICIENT_LIQUIDITY_BURNED');
        _burn(address(this), liquidity);
        _safeTransfer(_token0, to, amount0);
        _safeTransfer(_token1, to, amount1);
```

```
        balance0 = IERC20(_token0).balanceOf(address(this));
        balance1 = IERC20(_token1).balanceOf(address(this));

        _update(balance0, balance1, _reserve0, _reserve1);
        // reserve0 and reserve1 are up-to-date
        if (feeOn) kLast = uint256(reserve0).mul(reserve1);
        emit Burn(msg.sender, amount0, amount1, to);
    }
```

Fix status: Fixed

3.2 High Risk Vulnerabilities

High severity issues can affect the normal operation of smart contracts, and it is highly recommended to fix high severity vulnerability.

The audit has shown no high severity vulnerability.

3.3 Medium Risk Vulnerabilities

Medium severity issues can affect the operation of a smart contract, and it is recommended to fix medium severity vulnerability.

3.3.1 Function flaw

(1) In the BtswapRouter contract, when the swap function is called, since the passed path is an array, when one of the addresses in the path is in the whitelist and the other is not in the whitelist, and the whitelist token does not establish a transaction pair with ETH , When the swap function of BtswapToken is called, the liquidity mining reward cannot be calculated, and the user's mining share cannot be updated correctly.

```
function swap(address account, address input, uint256 amount, address output) public onlyMinter returns (bool) {
        require(account != address(0), "BtswapToken: taker swap account is the zero address");
        require(input != address(0), "BtswapToken: taker swap input is the zero address");
```

```
        require(output != address(0), "BtswapToken: taker swap output is the zero address");

        if (!isWhitelisted(input) || !isWhitelisted(output)) {
            return false;
        }

        uint256 quantity = weth(input, amount);
        if (quantity <= 0) {
            return false;
        }

        taker.timestamp = block.timestamp;
        taker.quantity = takerQuantityOfPool().add(quantity);

        User storage user = taker.users[account];
        user.timestamp = block.timestamp;
        user.quantity = takerQuantityOf(account).add(quantity);

        return true;
    }
```

```
function weth(address token) public view returns (uint256) {
        uint256 price = 0;

        if (WETH == token) {
            price = SafeMath.wad();
        }
        else if (IBtswapFactory(factory).getPair(token, WETH) != address(0)) {
            price = IBtswapPair(IBtswapFactory(factory).getPair(token, WETH)).price(token);
        }
        else {
            uint256 length = IBtswapWhitelistedRole(factory).getWhitelistedsLength();
            for (uint256 index = 0; index < length; index++) {
                address base = IBtswapWhitelistedRole(factory).whitelisteds(index);
                if (IBtswapFactory(factory).getPair(token, base) != address(0) && IBtswapFactory(factory).getPair(base,
WETH) != address(0)) {
                    uint256 price0 = IBtswapPair(IBtswapFactory(factory).getPair(token, base)).price(token);
                    uint256 price1 = IBtswapPair(IBtswapFactory(factory).getPair(base, WETH)).price(base);
                    price = price0.wmul(price1);
```

```
                    break;
                }
            }
        }

        return price;
    }
```

Fixed status: After confirmation with the project party, it has been considered here that when setting whitelist, the token will be configured to the ETH pair at the same time, which will not happen.

(2) When the _swap function is called in the BtswapRouter contract, the swap function of the BtswapToken contract will be called to add liquidity to the user, but the whitelist check in the swap function of the BtswapToken contract is used or (||) judgment, when the input is a non-whitelisted token but output is a whitelisted token, the user's mining share will not be updated correctly.

```
function _swap(uint256[] memory amounts, address[] memory path, address _to) internal {
        for (uint256 i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = IBtswapFactory(factory).sortTokens(input, output);
            uint256 amountInput = amounts[i];
            uint256 amountOut = amounts[i + 1];
            IBtswapToken(BT).swap(msg.sender, input, amountInput, output);
            (uint256 amount0Out, uint256 amount1Out) = input == token0 ? (uint256(0), amountOut) : (amountOut,
uint256(0));
            address to = i < path.length - 2 ? pairFor(output, path[i + 2]) : _to;
            IBtswapPair(pairFor(input, output)).swap(
                amount0Out, amount1Out, to, new bytes(0)
            );
        }
    }
```

```
function swap(address account, address input, uint256 amount, address output) public onlyMinter returns (bool) {
```

```
        require(account != address(0), "BtswapToken: taker swap account is the zero address");
        require(input != address(0), "BtswapToken: taker swap input is the zero address");
        require(output != address(0), "BtswapToken: taker swap output is the zero address");

        if (!isWhitelisted(input) || !isWhitelisted(output)) {
            return false;
        }

        uint256 quantity = weth(input, amount);
        if (quantity <= 0) {
            return false;
        }

        taker.timestamp = block.timestamp;
        taker.quantity = takerQuantityOfPool().add(quantity);

        User storage user = taker.users[account];
        user.timestamp = block.timestamp;
        user.quantity = takerQuantityOf(account).add(quantity);

        return true;
    }
```

Fix status: After confirmation with the project side, this is the design requirement

3.4 Low Risk Vulnerabilities

Low severity issues can affect smart contracts operation in future versions of code. We recommend the project party to evaluate and consider whether these problems need to be fixed.

3.4.1 Missing check

(1) The createPair function in the BtswapFactory contract uses or (||) to determine the whitelisted currency. Malicious user can combine the whitelisted currency and the

non-whitelisted currency to bypass the whitelist check to create a trading pair.

```
function createPair(address tokenA, address tokenB) external returns (address pair) {
        (address token0, address token1) = sortTokens(tokenA, tokenB);
        require(isWhitelisted(token0) || isWhitelisted(token1), 'BtswapFactory: TOKEN_UNAUTHORIZED');
        // single check is sufficient
        require(getPair[token0][token1] == address(0), 'BtswapFactory: PAIR_EXISTS');
        bytes memory bytecode = type(BtswapPair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        IBtswapPair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        // populate mapping in the reverse direction
        getPair[token1][token0] = pair;
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }
```

Fix status: After confirmation with the project side, this is the design requirement

(2) In the BtswapPairToken contract, there is no mandatory verification during initialize that the router cannot be address(0), which will cause the same pair to fail to be created, and the router cannot be reset through the contract.

```
function initialize(address _router, address _token0, address _token1) external {
        // sufficient check
        require(msg.sender == factory, "BtswapPairToken: FORBIDDEN");
        router = _router;
        token0 = _token0;
        token1 = _token1;
    }
```

Fix status: After confirmation with the project side, the contract adopts automatic deployment,

the intermediate interval will be very short, router will be initialized immediately, and this error will not occur.

3.4.2 Process bypassing

(1) Users can directly exchange tokens for BtswapPairToken without following the logic of BtswapRouter, but users cannot update the mining share.

```
function swap(uint256 amount0Out, uint256 amount1Out, address to, bytes calldata data) external lock {
        require(amount0Out > 0 || amount1Out > 0, 'BtswapPair: INSUFFICIENT_OUTPUT_AMOUNT');
        // gas savings
        (uint112 _reserve0, uint112 _reserve1,) = getReserves();
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'BtswapPair: INSUFFICIENT_LIQUIDITY');

        uint256 balance0;
        uint256 balance1;
        {// scope for _token{0,1}, avoids stack too deep errors
            address _token0 = token0;
            address _token1 = token1;
            require(to != _token0 && to != _token1, 'BtswapPair: INVALID_TO');
            // optimistically transfer tokens
            if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out);
            // optimistically transfer tokens
            if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out);
            if (data.length > 0) IBtswapCallee(to).bitswapCall(msg.sender, amount0Out, amount1Out, data);
            balance0 = IERC20(_token0).balanceOf(address(this));
            balance1 = IERC20(_token1).balanceOf(address(this));
        }
        uint256 amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
        uint256 amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
        require(amount0In > 0 || amount1In > 0, 'BtswapPair: INSUFFICIENT_INPUT_AMOUNT');
        {// scope for reserve{0,1}Adjusted, avoids stack too deep errors
            uint256 balance0Adjusted =
balance0.mul(1e4).sub(amount0In.mul(IBtswapFactory(factory).feeRateNumerator()));
            uint256 balance1Adjusted =
balance1.mul(1e4).sub(amount1In.mul(IBtswapFactory(factory).feeRateNumerator()));
```

```
            require(balance0Adjusted.mul(balance1Adjusted) >= uint256(_reserve0).mul(_reserve1).mul(1e8), 'BtswapPair:
K');
        }

        _update(balance0, balance1, _reserve0, _reserve1);
        emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
    }
```

Fix status: Fixed.

3.5 Enhancement Suggestion

The enhancement suggestion is the optimization proposal for the project, and the project party self-assess and considers whether these problems need to be optimized.

3.5.1 Accidentally obtain tokens

(1) An interface can be added to withdraw accidentally obtained tokens in the contract.

Fix status: After confirming with the project party, ETH will not be accidentally deposited at present. As for the tokens, it will increase the centralized owner logic and affect the decentralized design, so it will not be modified temporarily.

3.5.2 Risk control of fake tokens

(1) Be wary of fake token issues. Real token can be listed on the front-end display to prevent user asset loss (refer to Uniswap)

Fix status: Accepted, the front end will pay attention to adding this feature. We also have a token list to indicate that those tokens are approved by us.

4 Audit conclusion

4.1 Critical Vulnerabilities

- Process bypassing

4.2 Medium Risk vulnerabilities

- Function flaw

4.3 Low Risk vulnerabilities

- Missing check
- Process bypassing

4.4 Enhancement Suggestion

- Accidentally obtain tokens
- Risk control of fake tokens

4.5 Conclusion

Audit Result : Passed

Audit Number : 0X002008210001

Audit Date : Aug. 21, 2020

Audit Team : SlowMist Security Team

Summary conclusion: The are six security issues found during the audit. One critical risk issue, two medium risk issues, and three low risk issues. We also provide two enhancement suggestions. After communication and

feedback with the Btswap team, confirms that the risks found in the audit process are within the tolerable range.

5 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 base on these. For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of the smart contract, and is not responsible for it. 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 this report (referred to as "the provided information"). If the provided information is missing, tampered, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom.

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account