SALUS SECURITY

JUNE 2023

CODE SECURITY ASSESSMENT

BITCHAIN

Overview

Project Summary

- Name: BitChain
- Address: 0xc730e6eD9586351B802498d27a18F41E78f759Fb
- Platform: BNB Smart Chain
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	BitChain
Version	v1
Type	Solidity
Dates	June 13 2023
Logs	June 13 2023

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	3
Total Low-Severity issues	2
Total informational issues	7
Total	13

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Swapping from Uniswap V2 can be front-runned due to lack of slippage protection	High	Business Logic	Pending
2	Token transfer may fail when totalTax is 0 and isSwapBackEnabled is true	Medium	Business Logic	Pending
3	Lack of expiration timestamp check	Medium	Business Logic	Pending
4	Centralization risk	Medium	Centralization	Pending
5	Inconsistencies in setting of swapTokensAtAmount	Low	Business Logic	Pending
6	Unchecked return value from sendBNB	Low	Data Validation	Pending
7	Different condition checks before calling swapBack() in different functions	Informational	Data Validation	Pending
8	Redundant code	Informational	Redundancy	Pending
9	Variable denominator can be marked as constant instead of immutable	Informational	Gas Optimization	Pending
10	Missing isContract check for pair parameter	Informational	Data Validation	Pending
11	Inconsistency tax between code and documentation	Informational	Configuration	Pending
12	Missing two-step transfer ownership pattern	Informational	Business Logic	Pending
13	Use call instead of transfer for native tokens transfer	Informational	Business Logic	Pending

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Swapping from Uniswap V2 can be front-runned due to lack of slippage protection	
Severity: High	Category: Business Logic
Target: - BitChain.sol	

Description

Swapping BitChain(BIT) tokens in swapBack function through Uniswap V2 router can be front-runned resulting in loss of received native tokens for the protocol. The vulnerability arises because the argument amountOutMin in the swapExactTokensForETHSupportingFeeOnTransferTokens function is set to 0.

BitChain.sol:L1214-L1220

```
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        swapBackAmount,
        0,
        path,
        address(this),
        block.timestamp
)
```

This means that any amount is acceptable. This can result in an attacker sandwiching the transaction to extract profit which will be a loss to the protocol.

Recommendation

Consider setting some value for amountOutMin. This can be calculated from oracles. Please refrain from using spot price for calculating the amountOutMin because spot prices can also be manipulated by the attacker.

2. Token transfer may fail when totalTax is 0 and isSwapBackEnabled is true

Severity: Medium	Category: Business Logic
Target: - BitChain.sol	

Description

The swapBack function is called during token transfers when contractTokenBalance reaches the threshold and isSwapBackEnabled is true.

BitChain.sol:L1153-L1164

```
uint256 contractTokenBalance = balanceOf(address(this));

bool canSwap = contractTokenBalance >= swapTokensAtAmount;

if (
        canSwap &&
        !swapping &&
        !_isAutomatedMarketMakerPair[from] &&
        isSwapBackEnabled
) {
        swapBack();
}
```

In the swapBack function, the native currency obtained from the swapExactTokensForETHSupportingFeeOnTransferTokens function will be distributed according to the respective proportions of marketingTokenAmount and teamTokenAmount. The marketingTokenAmount and teamTokenAmount are increased only when transfers occur between addresses that are not excluded from fees.

BitChain.sol:L1202-L1240

```
function swapBack() internal inSwap {
        ...

        uint256 contractTokenBalance = balanceOf(address(this));

        uint256 totalTax = marketingTokenAmount + teamTokenAmount;

        uint256 swapBackAmount = contractTokenBalance;

        try
            uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
                swapBackAmount,
                0,
                path,
                address(this),
                block.timestamp
            )
        {} catch {
            return;
        }

        uint256 newBalance = address(this).balance;
        uint256 marketingBNB = (newBalance * marketingTokenAmount) / totalTax;
        uint256 teamBNB = newBalance - marketingBNB;

        ...
}
```

As the BitChain contract itself is exempt from fees, malicious users can transfer tokens directly to the contract and let its balance reach the threshold to trigger the swapBack function. Since the marketingTokenAmount and teamTokenAmount are not increased, the totalTax will be 0, causing the token transfer to fail.

Recommendation

Consider adding a check for totalTax. Additionally, as the native currency obtained from the swap needs to be distributed, consider defining a default allocation ratio for marketing and team when totalTax is 0.

3. Lack of expiration timestamp check	
Severity: Medium	Category: Business Logic
Target: - BitChain.sol	

Description

The process of exchanging ERC20 tokens lacks a transaction expiration check, which may lead to tokens being sold at a price lower than the market price at the moment of a swap.

BitChain.sol:L1214-L1220

```
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        swapBackAmount,
        0,
        path,
        address(this),
        block.timestamp
)
```

Setting the deadline parameter to block.timestamp basically disables the transaction expiration check because the deadline will be set to whatever timestamp the block including the transaction is minted at. The value of the deadline parameter has to be determined outside of the transaction.

Recommendation

Consider a reasonable value to the deadline argument. For example, Uniswap sets it to 30 minutes on the Ethereum mainnet and 5 minutes on L2 networks. Also, consider letting the owner and/or the DAO change the value when on-chain conditions change and may require a different value.

Also due to the fact that swapBack() is implemented in _transfer(), it will be triggered almost any time and it is not a user deal to input the deadline. Consider removing swapBack() from _transfer() and keeping manualSwapBack() or implementing another architect decision.

4. Centralization risk	
Severity: Medium	Category: Centralization

4. Centralization risk
Target: - BitChain.sol

Description

There is a privileged owner role in the BitChain contract.

The owner of the BitChain contract can change the tax or wallet configuration and claim stuck tokens. Should the owner's private key be compromised, an attacker could set marketingWallet and teamWallet to an address he controls and set all tax configurations to the maximum. If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

Consider transferring the privileged role to multi-sig account.

5. Inconsistencies in setting of swapTokensAtAmount	
Severity: Low	Category: Business Logic
Target: - BitChain.sol	

Description

In the following require statement, `totalSupply() / 100_000` will be 0.001% of the total supply. However, the error message states 0.00001%, which is inconsistent.

BitChain.sol:L1083-L1086

```
require(
    amount >= totalSupply() / 100_000,
    "Amount must be equal or greater than 0.00001% of Total Supply"
);
```

Recommendation

Consider fixing the mismatch between the error message and implementation.

6. Unchecked return value from sendBNB	
Severity: Low	Category: Data Validation
Target: - BitChain.sol	

Description

The return value of sendBNB is not checked. Therefore, if sendBNB fails silently, it may need an additional transaction to claim stuck tokens for the owner when the issue is noticed.

BitChain.sol:L1230-L1236

```
if (marketingBNB > 0) {
        sendBNB(marketingWallet, marketingBNB);
}

if (teamBNB > 0) {
        sendBNB(teamWallet, teamBNB);
}
```

Recommendation

Consider checking the return value of sendBNB and emitting an event if it fails. This will help the team detect failed sendBNB transactions.

2.3 Informational Findings

7. Different condition checks before calling swapBack() in different functions	
Severity: Informational	Category: Data Validation
Target: - BitChain.sol	

Description

In _transfer function, there are some condition checks before calling the swapBack function.

BitChain.sol:L1157-L1164

```
if (
    canSwap &&
    !swapping &&
    !_isAutomatedMarketMakerPair[from] &&
    isSwapBackEnabled
) {
    swapBack();
}
```

While in the manualSwapBack function, only the balance of the contract is checked. So if the owner set isSwapBackEnabled to false, it is still possible to call swapBack() through manualSwapBack().

BitChain.sol:L1256-L1262

```
function manualSwapBack() external {
        uint256 contractTokenBalance = balanceOf(address(this));

        require(contractTokenBalance > 0, "Cant Swap Back 0 Token!");

        swapBack();
}
```

Recommendation

Consider adding appropriate checks to the manualSwapBack function to improve code consistency

8. Redundant code	
Severity: Informational	Category: Redundancy
Target: - BitChain.sol	

Description

BitChain.sol:L1207-L1211

```
uint256 contractTokenBalance = balanceOf(address(this));
...
uint256 swapBackAmount = contractTokenBalance;
```

There is a redundant local variable contractTokenBalance in the swapBack function because it is not used after being assigned to swapBackAmount.

Recommendation

Consider using `uint256 swapBackAmount = balanceOf(address(this));`.

9. Variable denominator can be marked as constant instead of immutable	
Severity: Informational	Category: Gas Optimization
Target: - BitChain.sol	

Description

BitChain.sol:L874

```
uint256 public immutable denominator;
```

The variable denominator is marked as immutable instead of constant. It is set to 10000 in the constructor. Since it's assigned a literal number, it can be changed to constant and be set at compile time.

Recommendation

Consider changing the denominator from immutable to constant and removing the assignment in the constructor, i.e. `uint256 public constant denominator = 10_000`.

10. Missing isContract check for pair parameter	
Severity: Informational	Category: Data Validation
Target: - BitChain.sol	

Description

When setting the configuration of an address type, such as marketingWallet and teamWallet, an isContract check is included. However, the isContract check is missing for the pair parameter.

BitChain.sol:L1100-L1115

```
function setAutomatedMarketMakerPair(
        address pair,
        bool status
) external onlyOwner {
        require(
        pair != uniswapV2Pair,
        "Current Pair address cannot be modified"
        );
        require(
        _isAutomatedMarketMakerPair[pair] != status,
        "Pair address is already the value of 'status'"
        );
        _isAutomatedMarketMakerPair[pair] = status;

        emit UpdateAutomatedMarketMakerPair(pair, status);
}
```

Recommendation

Consider adding a isContract check for the pair parameter to improve code consistency.

11. Inconsistency tax between code and documentation	
Severity: Informational	Category: Configuration
Target: - BitChain.sol	

Description

The tokenomics document on the website states that the buy and sell tax should be 6%, but the code only defines it as 5% in the constructor.

BitChain.sol:L915-L921

```
marketingTaxBuy = 400;
marketingTaxSell = 400;
marketingTaxTransfer = 400;

teamTaxBuy = 100;
teamTaxSell = 100;
teamTaxTransfer = 100;
```

For each transfer, when takeFee is true, a fee will be deducted according to marketingTax and teamTax, which is 5% for current configuration.

BitChain.sol:L1140-L1200

```
function _transfer(
       address from,
       address to,
       uint256 amount
) internal override {
       ...

       if (takeFee) {
           uint256 tempMarketingAmount;
           uint256 tempTeamAmount;

           if (_isAutomatedMarketMakerPair[from]) {
              tempMarketingAmount = (amount * marketingTaxBuy) / denominator;
              tempTeamAmount = (amount * teamTaxBuy) / denominator;
           } else if (_isAutomatedMarketMakerPair[to]) {
              tempMarketingAmount = (amount * marketingTaxSell) / denominator;
              tempTeamAmount = (amount * teamTaxSell) / denominator;
           } else {
              ...
           }
           marketingTokenAmount += tempMarketingAmount;
           teamTokenAmount += tempTeamAmount;

           uint256 fees = tempMarketingAmount + tempTeamAmount;

           if (fees > 0) {
              amount -= fees;
              super._transfer(from, address(this), fees);
           }
       }

       ...
}
```

Recommendation

Consider fixing the mismatch between the configuration and documentation.

12. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - BitChain.sol	

Description

The BitChain contract inherits from the Ownable contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2Step contract from OpenZeppelin instead.

13. Use call instead of transfer for native tokens transfer	
Severity: Informational	Category: Business logic
Target: - BitChain.sol	

Description

The transfer function is not recommended for sending native tokens due to its 2300 gas unit limit which may not work with smart contract wallets or multi-sig. Instead, call can be used to circumvent the gas limit.

BitChain.sol:L971-L974

```
if (token == address(0x0)) {
     payable(msg.sender).transfer(address(this).balance);
     return;
}
```

Recommendation

Consider using call instead of transfer for sending native token.

Example:

```
payable(msg.sender).call{value: address(this).balance}("");
```

Appendix

Appendix 1 - Files in Scope

This audit covered the following file from address 0xc730e6eD9586351B802498d27a18F41E78f759Fb:

File	SHA-1 hash
BitChain.sol	95f75d7f94d62cf933dd97c8b67063fc5f8ea0c9