



Coinsult

Advanced Manual Smart Contract Audit



Project: RunLix

Website: <http://runlix.com>

Low-Risk

4 low-risk code
issues found

Medium-Risk

0 medium-risk code
issues found

High-Risk

0 high-risk code
issues found

Contract Address

0x002c059ce521Fa3333ba4C771042dAcB6a93DCFD

Disclaimer: Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

Disclaimer

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

Tokenomics

Rank	Address	Quantity (Token)	Percentage
1	0x46636e41ab2253495c36d20d28a4ba4b6bdc32a2	2,500,000,000	100.0000%

Source Code

Coinsult was comissioned by RunLix to perform an audit based on the following smart contract:

<https://bscscan.com/address/0x002c059ce521fa3333ba4c771042dacb6a93dcfd#code>

Manual Code Review

In this audit report we will highlight all these issues:

Low-Risk

4 low-risk code
issues found

Medium-Risk

0 medium-risk code
issues found

High-Risk

0 high-risk code
issues found

The detailed report continues on the next page...

● **Low-Risk:** Could be fixed, will not bring problems.

Contract contains Reentrancy vulnerabilities

Additional information: This combination increases risk of malicious intent. While it may be justified by some complex mechanics (e.g. rebase, reflections, buyback).

More information: Slither

```
function _transfer(
    address sender,
    address recipient,
    uint256 amount
) internal virtual override {
    if (antiBotTime > block.timestamp &&
        amount > antiBotAmount &&
        bots[sender]){
        revert("Anti Bot");
    }

    uint256 transferFeeRate = recipient == uniswapV2Pair
        ? sellFeeRate
        : (sender == uniswapV2Pair ? buyFeeRate : 0);
    if (transferFeeRate > 0) {
        uint256 _fee = amount.mul(transferFeeRate).div(100);
        super._transfer(sender, address(this), _fee); // TransferFee
        amount = amount.sub(_fee);
    }

    uint256 contractTokenBalance = balanceOf(address(this));
    bool overMinimumTokenBalance = contractTokenBalance >= minimumTokensBeforeSwap;
```

Recommendation

Apply the check-effects-interactions pattern.

Exploit scenario

```
function withdrawBalance(){
    // send userBalance[msg.sender] Ether to msg.sender
    // if msg.sender is a contract, it will call its fallback function
    if( ! (msg.sender.call.value(userBalance[msg.sender]))( ) ){
        throw;
    }
    userBalance[msg.sender] = 0;
}
```

Bob uses the re-entrancy bug to call withdrawBalance two times, and withdraw more than its initial deposit to the contract.

● **Low-Risk:** Could be fixed, will not bring problems.

Avoid relying on `block.timestamp`

`block.timestamp` can be manipulated by miners.

```
function antiBot(uint256 amount) external onlyOwner {
    require(amount > 0, "not accept 0 value");
    require(!antiBotEnabled);

    antiBotAmount = amount * 10**18;
    antiBotTime = block.timestamp.add(antiBotDuration);
    antiBotEnabled = true;
}
```

Recommendation

Do not use `block.timestamp`, `now` or `blockhash` as a source of randomness

Exploit scenario

```
contract Game {

    uint reward_determining_number;

    function guessing() external{
        reward_determining_number = uint256(block.blockhash(10000)) % 10;
    }
}
```

Eve is a miner. Eve calls `guessing` and re-orders the block containing the transaction. As a result, Eve wins the game.

● **Low-Risk:** Could be fixed, will not bring problems.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setTransferFeeRate(uint256 _sellFeeRate, uint256 _buyFeeRate) public onlyOwner {  
    sellFeeRate = _sellFeeRate;  
    buyFeeRate = _buyFeeRate;  
}
```

Recommendation

Emit an event for critical parameter changes.

Exploit scenario

```
contract C {  
  
    modifier onlyAdmin {  
        if (msg.sender != owner) throw;  
        _;  
    }  
  
    function updateOwner(address newOwner) onlyAdmin external {  
        owner = newOwner;  
    }  
}
```

updateOwner() has no event, so it is difficult to track off-chain changes in the buy price.

● **Low-Risk:** Could be fixed, will not bring problems.

Redundant Statements

Detect the usage of redundant statements that have no effect.

```
function _msgData() internal view virtual returns (bytes memory) {  
    this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2318  
    return msg.data;  
}
```

Recommendation

Remove redundant statements if they congest code but offer no value.

Exploit scenario

```
contract RedundantStatementsContract {  
  
    constructor() public {  
        uint; // Elementary Type Name  
        bool; // Elementary Type Name  
        RedundantStatementsContract; // Identifier  
    }  
  
    function test() public returns (uint) {  
        uint; // Elementary Type Name  
        assert; // Identifier  
        test; // Identifier  
        return 777;  
    }  
}
```

Each commented line references types/identifiers, but performs no action with them, so no code will be generated for such statements and they can be removed.

Owner privileges

- Owner cannot change max transaction amount
- Owner can set fees higher than 25%
- Owner can pause the contract
- Owner can blacklist addresses
- ⚠ Owner can set minimum amount of tokens to sell
- ⚠ Owner can enable antibot

Extra notes by the team

No notes

Contract Snapshot

```
contract RunLix is Ownable, ERC20 {
    using SafeMath for uint256;

    mapping (address => bool) private bots;
    uint256 public maxSupply = 2500 * 10**6 * 10**18;

    IUniswapV2Router02 public uniswapV2Router;
    address public uniswapV2Pair;
    address payable private DevAddress;
    uint256 public sellFeeRate = 0;
    uint256 public buyFeeRate = 0;
    uint256 public minimumTokensBeforeSwap = 20000 * 10**18;

    bool public swapAndLiquifyEnabled = false;
    bool inSwapAndLiquify;
```

Website Review

Coinsult checks the website completely manually and looks for visual, technical and textual errors. We also look at the security, speed and accessibility of the website. In short, a complete check to see if the website meets the current standard of the web development industry.

- Mobile Friendly
- Does not contain jQuery errors
- SSL Secured
- No major spelling errors

Project Overview

● Not KYC verified by Coinsult

AUDITED
BY COINSULT.NET

