



# Coinsult

## Advanced Manual Smart Contract Audit



**Project:** Running man

**Website:** <https://www.runningmancoin.com/>

**Low-Risk**

7 low-risk code  
issues found

**Medium-Risk**

0 medium-risk code  
issues found

**High-Risk**

0 high-risk code  
issues found

**Contract Address**

0x2b79BF838d002696C97EDc83C9021527d1fD1d93

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	0xe4da7e796e7e8f76f217dc0da6d1346988d41bca	576,000,000,000,000	57.6000%
2	PinkSale: PinkLock	424,000,000,000,000	42.4000%

# Source Code

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

<https://bscscan.com/address/0x2b79BF838d002696C97EDc83C9021527d1fD1d93#code>

# Manual Code Review

In this audit report we will highlight all these issues:

## Low-Risk

7 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 from,
    address to,
    uint256 amount
) internal override {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");
    require(!_isEnemy[from] &&& !_isEnemy[to], 'Enemy address');

    if(amount == 0) {
        super._transfer(from, to, 0);
        return;
    }

    uint256 contractTokenBalance = balanceOf(address(this));

    bool canSwap = contractTokenBalance >= swapTokensAtAmount;

    if( canSwap &&&
        !swapping &&&
        !automatedMarketMakerPairs[from] &&&
        from != owner() &&&
```

## 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.

## Too many digits

Literals with many digits are difficult to read and review.

```
function updateGasForProcessing(uint256 newValue) public onlyOwner {
    require(newValue >= 200000 && newValue <= 500000, "GasForProcessing must be between 200000 and 500000");
    require(newValue != gasForProcessing, "Cannot update gasForProcessing to same value");
    emit GasForProcessingUpdated(newValue, gasForProcessing);
    gasForProcessing = newValue;
}
```

## Recommendation

Use: Ether suffix, Time suffix, or The scientific notation

## Exploit scenario

```
contract MyContract{
    uint 1_ether = 1000000000000000000;
}
```

While 1\_ether looks like 1 ether, it is 10 ether. As a result, it's likely to be used incorrectly.

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

## No zero address validation for some functions

Detect missing zero address validation.

```
rewardToken = rewardAddr_;
```

## Recommendation

Check that the new address is not zero.

## Exploit scenario

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

Bob calls updateOwner without specifying the newOwner, so Bob loses ownership of the contract.

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

## Functions that send Ether to arbitrary destinations

Unprotected call to a function sending Ether to an arbitrary address.

```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {  
    // approve token transfer to cover all possible scenarios  
    _approve(address(this), address(uniswapV2Router), tokenAmount);  
    // add the liquidity  
    uniswapV2Router.addLiquidityETH{value: ethAmount}(  
        address(this),  
        tokenAmount,  
        0, // slippage is unavoidable  
        0, // slippage is unavoidable  
        address(0),  
        block.timestamp  
    );  
}
```

## Recommendation

Ensure that an arbitrary user cannot withdraw unauthorized funds.

## Exploit scenario

```
contract ArbitrarySend{  
    address destination;  
    function setDestination(){  
        destination = msg.sender;  
    }  
  
    function withdraw() public{  
        destination.transfer(this.balance);  
    }  
}
```

Bob calls setDestination and withdraw. As a result he withdraws the contract's balance.

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

## Unchecked transfer

The return value of an external transfer/transferFrom call is not checked.

```
function swapAndSendToFee(uint256 tokens) private {
    uint256 initialCAKEBalance = IERC20(rewardToken).balanceOf(address(this));
    swapTokensForCake(tokens);
    uint256 newBalance = (IERC20(rewardToken).balanceOf(address(this))).sub(initialCAKEBalance);
    IERC20(rewardToken).transfer(_marketingWalletAddress, newBalance);
    AmountMarketingFee = AmountMarketingFee - tokens;
}
```

## Recommendation

Use SafeERC20, or ensure that the transfer/transferFrom return value is checked.

## Exploit scenario

```
contract Token {
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool success);
}
contract MyBank{
    mapping(address => uint) balances;
    Token token;
    function deposit(uint amount) public{
        token.transferFrom(msg.sender, address(this), amount);
        balances[msg.sender] += amount;
    }
}
```

Several tokens do not revert in case of failure and return false. If one of these tokens is used in MyBank, deposit will not revert if the transfer fails, and an attacker can call deposit for free..



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

## Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setSwapTokensAtAmount(uint256 amount) public onlyOwner {
    swapTokensAtAmount = amount;
}
```

## 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.

## Conformance to Solidity naming conventions

Allow `_` at the beginning of the `mixed_case` match for private variables and unused parameters.

```
mapping(address => bool) public _isEnemy;
```

## Recommendation

Follow the Solidity naming convention.

## Rule exceptions

- Allow constant variable name/symbol/decimals to be lowercase (ERC20).
- Allow `_` at the beginning of the `mixed_case` match for private variables and unused parameters.

## Owner privileges

- Owner cannot pause trading
- Owner cannot change max transaction amount
- Owner can set fees higher than 25%
- Owner can exclude from fees
- Owner can blacklist addresses

## Extra notes by the team

No notes

# Contract Snapshot

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

    IUniswapV2Router02 public uniswapV2Router;
    address public uniswapV2Pair;

    bool private swapping;

    TokenDividendTracker public dividendTracker;

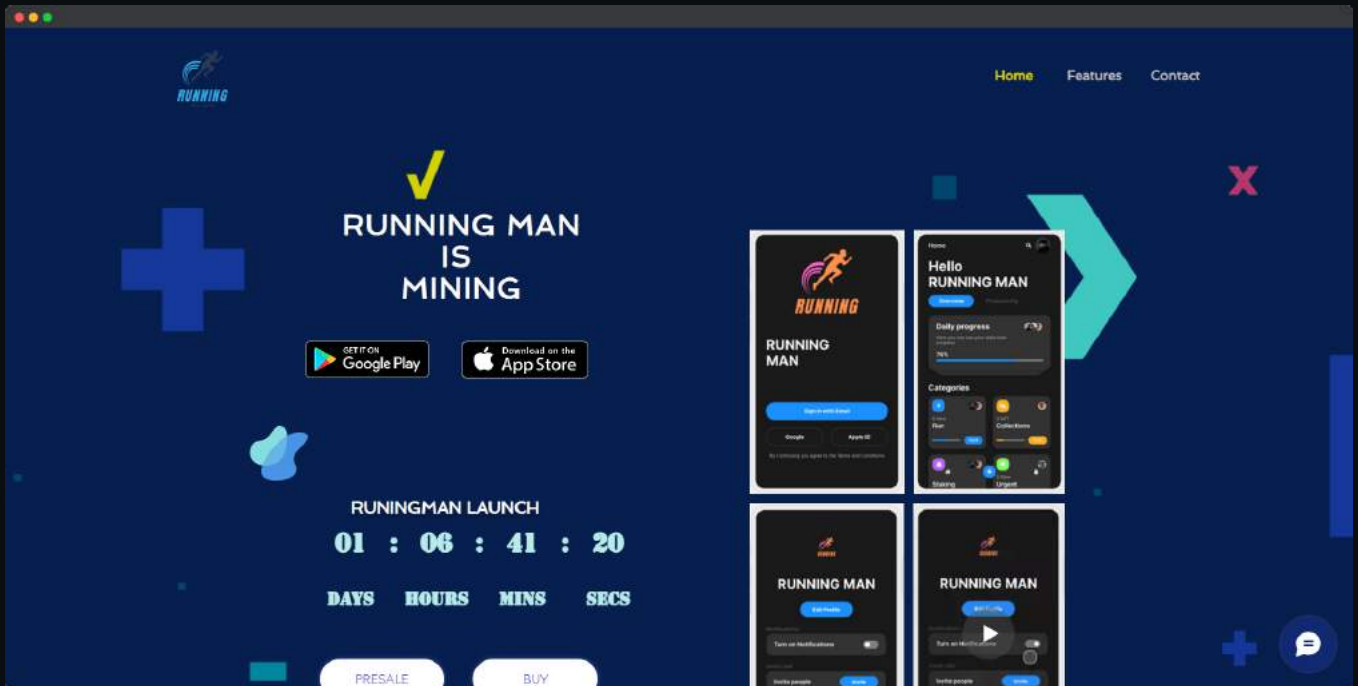
    address public rewardToken;

    uint256 public swapTokensAtAmount;

    uint256 public buyTokenRewardsFee;
    uint256 public sellTokenRewardsFee;
    uint256 public buyLiquidityFee;
    uint256 public sellLiquidityFee;
    uint256 public buyMarketingFee;
    uint256 public sellMarketingFee;
    uint256 public buyDeadFee;
    uint256 public sellDeadFee;
    uint256 public AmountLiquidityFee;
    uint256 public AmountTokenRewardsFee;
    uint256 public AmountMarketingFee;
```

# 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

