Coinsult

Advanced Manual Smart Contract Audit

Project: MoveEarn
Website: https://www.moveearn.io/

● **Low-Risk**
2 low-risk code issues found

● **Medium-Risk**
2 medium-risk code issues found

● **High-Risk**
0 high-risk code issues found

Contract Address
0xF2f7063932761241afeDb4A514eb5D28901cc86B

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	0x97858fd65f3839acc213bd26dc95a52ce267ee79	5,000,000	100.0000%

Source Code

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

https://bscscan.com/address/0xF2f7063932761241afeDb4A514eb5D28901cc86B#code

Manual Code Review

In this audit report we will highlight all these issues:

🟢 Low-Risk

2 low-risk code
issues found

🟡 Medium-Risk

2 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 (!isAntiWhaleEnded()) {
        if (!presaleAddress[sender]) {
            require(amount &lt;= antiWhaleAmount, &quot;PUMPING_IS_NOT_ALLOWED&quot;);
        }
    }

    if (inSwap) {
        super._transfer(sender, recipient, amount);
        return;
    }

    if (sender != uniswapV2Pair &amp;&amp; recipient != uniswapV2Pair) {
        if (shouldSwapBack()) {
            swapBack();
        }
    }
```

Recommendation

Apply the check-effects-interactions pattern.

Exploit scenario

```
function withdrawBalance(){
    // send userBalance[msg.sender] Ether to msg.sender
    // if mgs.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.

```
// make the swap
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
    tokenAmount,
    0, // accept any amount of ETH
    path,
    receiverEth, // The contract
    block.timestamp
);
```

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.

No zero address validation for some functions

Detect missing zero address validation.

```
Fixed and resolved

function setStakingAddress(address _stakingAddress) external onlyTreasury {
        stakingAddress = _stakingAddress;
    }
```

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`, soBob 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.

```
Fixed and resolved

function sweepBNB(address _to) public onlyTreasury {
     payable(_to).transfer(address(this).balance);
 }
```

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.

```
Fixed and resolved

function sweepToken(address _token, address _to) public onlyTreasury {
    IERC20(_token).transfer(_to, IERC20(_token).balanceOf(address(this)));
}
```

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.

Conformance to Solidity naming conventions

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

```
function DOMAIN_SEPARATOR() external view returns (bytes32);

function PERMIT_TYPEHASH() external pure returns (bytes32);
```

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.

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

Redundant Statements

Detect the usage of redundant statements that have no effect.

```
Fixed and resolved
function _msgData() internal view virtual returns (bytes memory) {
       this;
       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.

🟡 **Medium-Risk:** Should be fixed, could bring problems.

Treasury and Staking Addresses are set to dead wallet – 🟢 Fixed and resolved

```
address public treasuryAddress = 0x0000000000000000000000000000000000000000;
address public stakingAddress = 0x0000000000000000000000000000000000000000;
```

Recommendation

Set the wallets to the appropriate wallets.

🟡 **Medium-Risk:** Should be fixed, could bring problems.

Antiwhale amount not checked – 🟢 Fixed and resolved

```
function setAntiWhaleAmount(uint256 _amount) public onlyTreasury {
    // set anti whale amount also activate antibot
    require(antiWhaleAmount == 0, "Can set antiWhale once"); // antiWhale can only activate once
    antiWhaleAmount = _amount;
    antiWhaleStartTime = block.timestamp - 10; // safty margin 10s
}
```

Recommendation

Since you can only set antiwhale amount once, it should be correct. We recommend to add require statements to ensure it is between a certain window to avoid issues with the contract. Fix: //fix anti whale uint public constant antiWhaleMin = 1; uint public constant antiWhaleMax = 1000; function setAntiWhaleAmount(uint256 _amount) public onlyTreasury { //fix require(_amount >= antiWhaleMin && _amount <= antiWhaleMax, "Invalid Value"); // set anti whale amount also activate antibot require(antiWhaleAmount == 0, "Can set antiWhale once"); // antiWhale can only activate once antiWhaleAmount = _amount; antiWhaleStartTime = block.timestamp – 10; // safty margin 10s
}

Owner privileges

- 🟢 Owner cannot set fees higher than 25%
- 🟢 Owner cannot pause trading
- 🟢 Owner cannot change max transaction amount

⚠️ Treasury wallet can set antiwhaleamount

Extra notes by the team

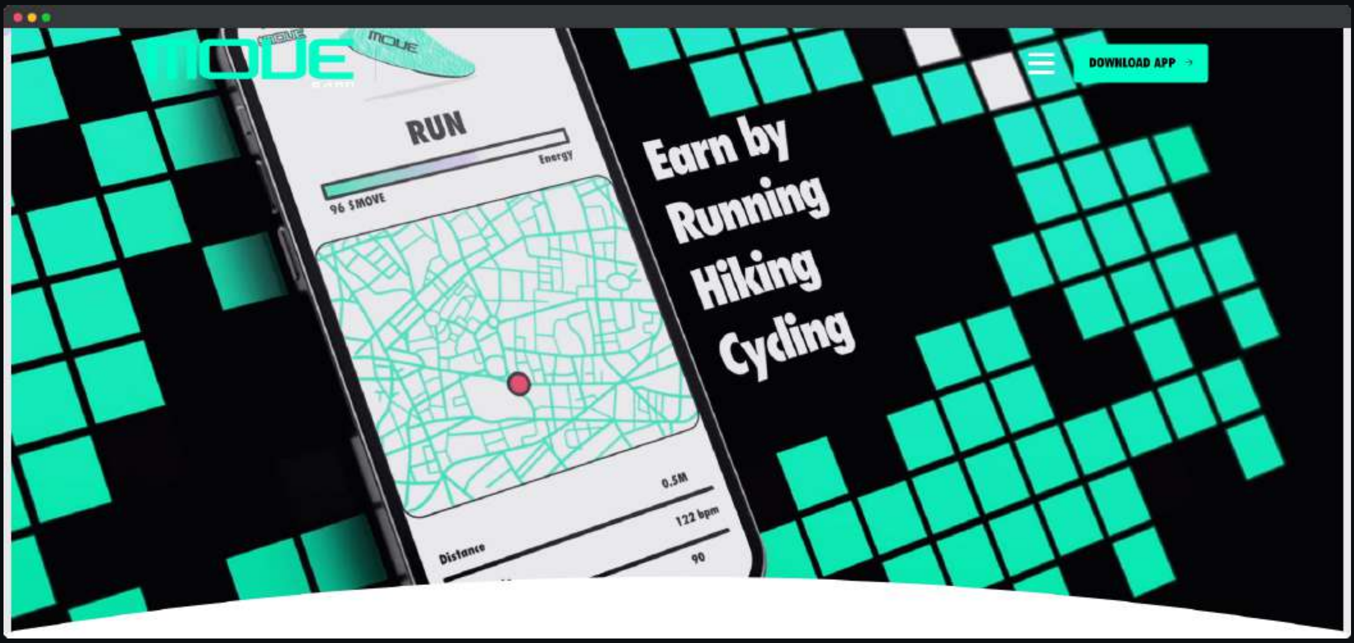
Even when the ownership is renounced, the treasury wallet can still change a lot of the contract.

Contract Snapshot

```
contract MoveEarn is TokenERC20 {
using SafeMath for uint256;

uint256 public constant DECIMALS = 18;

IUniswapV2Router02 public uniswapV2Router;

uint256 public constant maxSupply = 5 * 10**6 * 10**18;

address public uniswapV2Pair;

bool inSwap = false;

modifier swapping() {
    inSwap = true;
    _;
    inSwap = false;
}

constructor() TokenERC20("MOVE Earn", "MOVE", uint8(DECIMALS)) {
    _mint(_msgSender(), maxSupply);
```

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

MoveEarn

Audited by Coinsult.net

Date: 30 May 2022

✓ Advanced Manual Smart Contract Audit