Coinsult

Advanced Manual Smart Contract Audit

Project: DiscDoge

Website: https://www.discdoge.com/

● Low-Risk

6 low-risk code issues found

● Medium-Risk

0 medium-risk code issues found

● High-Risk

0 high-risk code issues found

Contract Address

0x55bD484ba92cb52c9ea393e3Ff55d3E3A4Fb1D87

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	Null Address: 0x000...dEaD	8,000,000,000	80.0000%
2	0x33df49c82a97b267cb6993383a7270fe7ffdeea2	1,900,000,000	19.0000%
3	0xf9a870777b2bbf244145a7d78c6d16476e09e5bb	100,000,000	1.0000%

Source Code

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

https://bscscan.com/address/0x55bd484ba92cb52c9ea393e3ff55d3e3a4fb1d87#code

Manual Code Review

In this audit report we will highlight all these issues:

🟢 Low-Risk

6 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 {
    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");

    _beforeTokenTransfer(sender, recipient, amount);

    _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
    _balances[recipient] = _balances[recipient].add(amount);
    emit Transfer(sender, recipient, amount);
}
```

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.

No zero address validation for some functions

Detect missing zero address validation.

```
function updateUniswapV2Router(address newAddress) public onlyOwner {
    require(newAddress != address(uniswapV2Router), "TOKEN: The router already has that address");
    emit UpdateUniswapV2Router(newAddress, address(uniswapV2Router));
    uniswapV2Router = IUniswapV2Router02(newAddress);
    address _uniswapV2Pair = IUniswapV2Factory(uniswapV2Router.factory())
        .createPair(address(this), uniswapV2Router.WETH());
    uniswapV2Pair = _uniswapV2Pair;
}
```

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.

```
uint256 BNBAmount = address(this).balance;

uint256 BNBAmountForDev = BNBAmount.div(2);
uint256 BNBAmountForMkt = BNBAmount.sub(BNBAmountForDev);

payable(_AppWalletAddress).transfer(BNBAmountForDev);
payable(_MktWalletAddress).transfer(BNBAmountForMkt);

swapping = false;
```

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.

```
uint256 BNBAmount = address(this).balance;

uint256 BNBAmountForDev = BNBAmount.div(2);
uint256 BNBAmountForMkt = BNBAmount.sub(BNBAmountForDev);

payable(_AppWalletAddress).transfer(BNBAmountForDev);
payable(_MktWalletAddress).transfer(BNBAmountForMkt);

swapping = false;
```

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 setFee(
    uint256 _MktFee,
    uint256 _DevFee,
    uint256 _LpFee,
    uint256 _BurnFee,
    uint256 _UsefulShare,
    uint256 _OtherShare
) public onlyOwner {
    MktFee = _MktFee;
    DevFee = _DevFee;
    LpFee = _LpFee;
    BurnFee = _BurnFee;

    UsefulShare = _UsefulShare;
    OtherShare = _OtherShare;

    AllFee = MktFee.add(DevFee).add(LpFee).add(BurnFee);
    AllShare = UsefulShare.add(OtherShare);
}
```

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 calldata) {
    this; // silence state mutability warning without generating bytecode - see https://github.com/e
    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 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

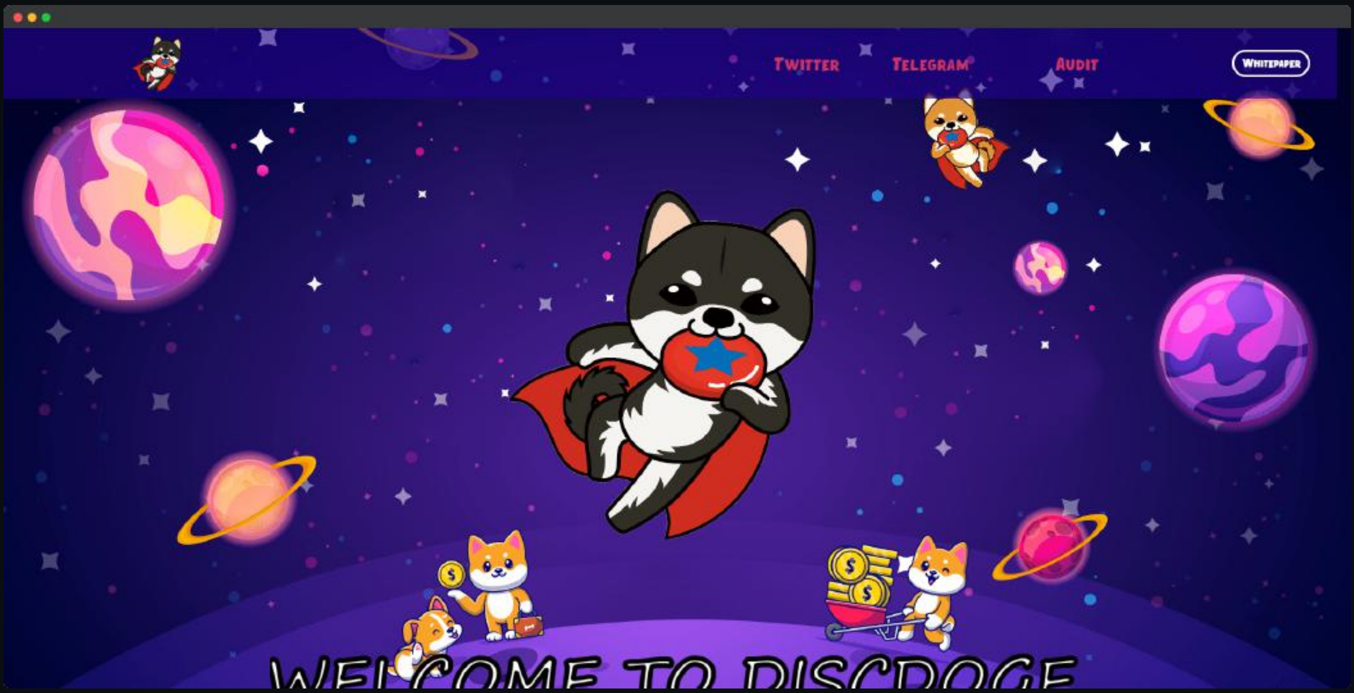
No notes

Contract Snapshot

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

IUniswapV2Router02 public uniswapV2Router;
address public  uniswapV2Pair;

bool private swapping;

address public deadWallet = 0x000000000000000000000000000000000000dEaD;
address public _AppWalletAddress = 0x658BB29e9df2CE396b35E1864846E6d8b0d50c31;
address public _MktWalletAddress = 0xf78627C6499c99783880E52d742DB18e50AC0FBc;
```

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

DiscDoge

Audited by Coinsult.net

Date: 17 June 2022

✓ Advanced Manual Smart Contract Audit