



Coinsult

Advanced Manual Smart Contract Audit



Project: Dogeball

Website: <https://www.dogeball.games/>

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

0xFE10FB3526894190f12017E5079005c96DbCD39f

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	0xe00a0a928c83030bf6ec324c495d8eb920312cae	35,000,000,000,000	35.0000%
2	Null Address: 0x000...dEaD	32,000,000,000,000	32.0000%
3	0xf49c49a74871c69590f1d8315caaf8b19e9c84db	24,500,000,000,000	24.5000%
4	0xc4111596e7a8f55c5c76bf1d1c3c6b8cb1ce1b8c	5,000,000,000,000	5.0000%
5	0xb76204aac2bc34a956f3b41e7cdac9f572255c88	2,000,000,000,000	2.0000%

Source Code

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

<https://bscscan.com/address/0xFE10FB3526894190f12017E5079005c96DbCD39f#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) private returns (bool) {

    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");
    require(amount > 0, "Transfer amount must be greater than zero");
    if(inSwapAndLiquify)
    {
        return _basicTransfer(sender, recipient, amount);
    }
    else
    {
        if(!isTxLimitExempt[sender] &&& !isTxLimitExempt[recipient]) {
            require(amount = _minimumTokensBeforeSwap;

            if (overMinimumTokenBalance &&& !inSwapAndLiquify &&& !isMarketPair[sender] &a
            {
                if(swapAndLiquifyByLimitOnly)
                    contractTokenBalance = _minimumTokensBeforeSwap;
                swapAndLiquify(contractTokenBalance);
            }

            balances[sender] = balances[sender].sub(amount, "Insufficient Balance");
```

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 getTime() public view returns (uint256) {  
    return 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.

```
function setMarketingWalletAddress(address newAddress) external onlyOwner() {  
    marketingWalletAddress = payable(newAddress);  
}
```

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.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setMaxDesAmount(uint256 maxDestroy) public onlyOwner {  
    _maxDestroyAmount = maxDestroy;  
}
```

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.

```
uint256 public _buyLiquidityFee = 2;  
uint256 public _buyMarketingFee = 3;  
uint256 public _buyTeamFee = 4;  
uint256 public _buyDestroyFee = 0;
```

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.

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

Extra notes by the team

No notes

Contract Snapshot

```
contract TokenTool is Context, IERC20, Ownable {

    using SafeMath for uint256;
    using Address for address;

    string private _name;
    string private _symbol;
    uint8 private _decimals;
    address payable public marketingWalletAddress;
    address payable public teamWalletAddress;
    address public deadAddress = 0x00000000000000000000000000000000dEaD;

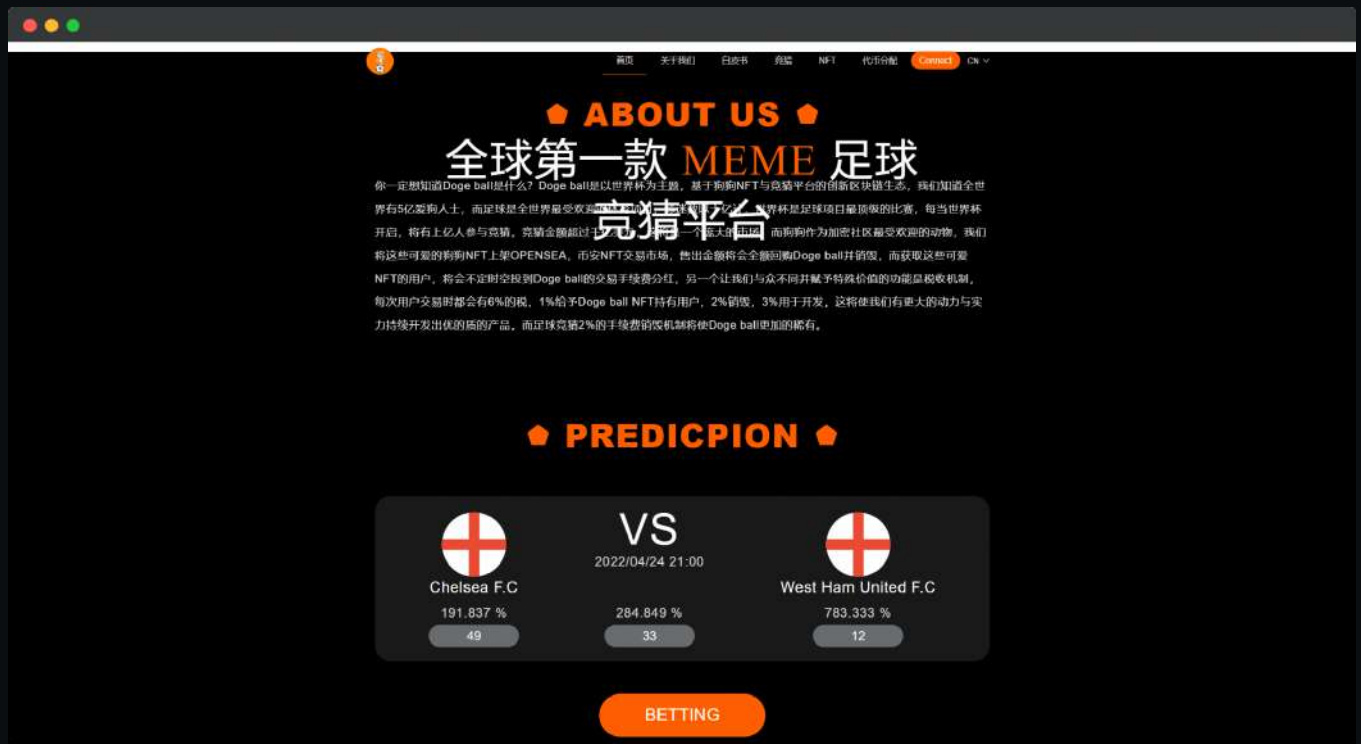
    mapping (address => uint256) _balances;
    mapping (address => mapping (address => uint256)) private _allowances;

    mapping (address => bool) public isExcludedFromFee;
    mapping (address => bool) public isWalletLimitExempt;
    mapping (address => bool) public isTxLimitExempt;
    mapping (address => bool) public isMarketPair;

    uint256 public _buyLiquidityFee = 2;
    uint256 public _buyMarketingFee = 3;
    uint256 public _buyTeamFee = 4;
    uint256 public _buyDestroyFee = 0;
```

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

