Coinsult

Advanced Manual Smart Contract Audit

Project: Boring Doge

Website: http://boringdoge.xyz/

● Low-Risk	● Medium-Risk	● High-Risk
7 low-risk code issues found	0 medium-risk code issues found	0 high-risk code issues found

Contract Address

0x12fEbf8fA1EE2D1CbC365b2f01A4088ebB91F9B2

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	0xbcb0a750f9b34d956235052be98e095550407378	60,000,000,000	60.0000%
2	Null Address: 0x000...dEaD	40,000,000,000	40.0000%

Source Code

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

https://bscscan.com/address/0x12fEbf8fA1EE2D1CbC365b2f01A4088ebB91F9B2#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 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(!isbotBlackList[sender], "account is bot");

    if(inSwapAndLiquify)
    {
        return _basicTransfer(sender, recipient, amount);
    }
    else
    {
        if(!isTxLimitExempt[sender] &amp;&amp; !isTxLimitExempt[recipient]) {
            require(amount = minimumTokensBeforeSwap;

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

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.

Too many digits

Literals with many digits are difficult to read and review.

```
uint256 public _walletMax =     100000000000 * 10**_decimals;
```

Recommendation

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

Exploit scenario

```
contract MyContract{
    uint 1_ether = 10000000000000000000;
}
```

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.

```
function burnBNB(address payable burnAddress) external onlyOwner {
    burnAddress.transfer(address(this).balance);
}
```

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.

```
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
        liqWalletAddress,
        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.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setBuyTaxes(uint256 newLiquidityTax, uint256 newMarketingTax) external onlyOwner() {
    _buyLiquidityFee = newLiquidityTax;
    _buyMarketingFee = newMarketingTax;

    _totalTaxIfBuying = _buyLiquidityFee.add(_buyMarketingFee);
}
```

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 _walletMax =     100000000000 * 10**_decimals;
```

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/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 can change max transaction amount
- 🟡 Owner can set fees higher than 25%
- 🟡 Owner can exclude from fees
- 🟡 Owner can pause the contract
- 🔴 Owner can blacklist addresses

⚠️ Owner can set wallet limit

Extra notes by the team

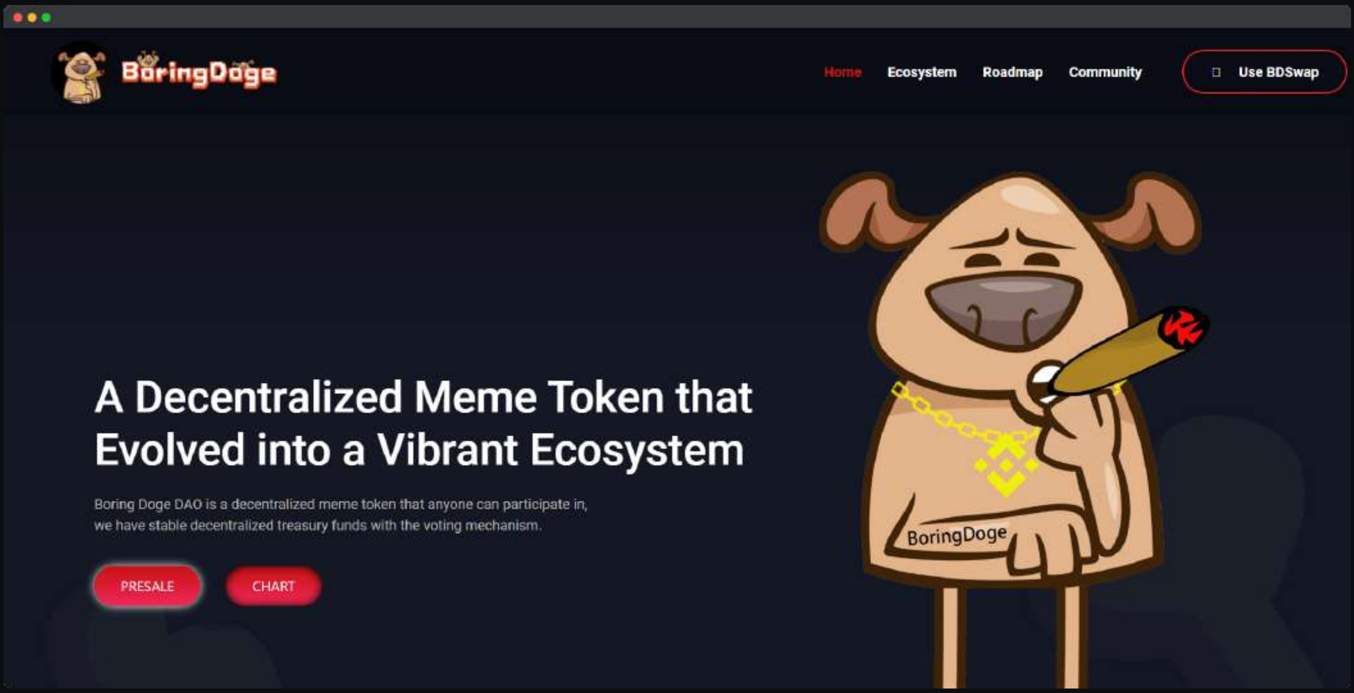
No notes

Contract Snapshot

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

using SafeMath for uint256;
using Address for address;

string private _name = "BoringDogeDAO";
string private _symbol = "BoringDoge";
uint8 private _decimals = 18;

address payable public marketingWalletAddress = payable(0xF4094BEf48364a662f8FA3B6a7dbC11E8A4E21C7);
address payable public liqWalletAddress = payable(0xD2e877c70Bc7e32e2b49c5b7Ef505614B0b8d706);

address public immutable deadAddress = 0x000000000000000000000000000000000000dEaD;
```

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.



A Decentralized Meme Token that Evolved into a Vibrant Ecosystem

Boring Doge DAO is a decentralized meme token that anyone can participate in, we have stable decentralized treasury funds with the voting mechanism.

● Mobile Friendly

● Does not contain jQuery errors

● SSL Secured

● No major spelling errors

Project Overview

● Not KYC verified by Coinsult