Coinsult

Advanced Manual Smart Contract Audit

Project: The tree token

Website: https://thetreetoken.io/

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

Contract Address

0x976242a0f07F4C5211299aa66B72F4cf938c1Fdb

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	0x608427c75256f800f3eba463c00c695ce5dd51ac	333,333,333	100.0000%

Source Code

Coinsult was comissioned by The tree token to perform an audit based on the following smart contract:

https://bscscan.com/address/0x976242a0f07f4c5211299aa66b72f4cf938c1fdb#code

Manual Code Review

In this audit report we will highlight all these issues:

Low-Risk

9 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) private open(from, to)
{
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");
    require(amount &gt; 0, "Transfer amount must be greater than zero");

    if(!_isExemptFromTxLimit[from] &amp;&amp; !_isExemptFromTxLimit[to])
    {
        require(amount = _minimumTokensBeforeSwap;

    if (!inSwapAndLiquify &amp;&amp; swapAndLiquifyEnabled &amp;&amp; from != uniswapV2Pair)
        if (overMinimumTokenBalance)
        {
            contractTokenBalance = _minimumTokensBeforeSwap;
            swapAndLiquify(contractTokenBalance);
        }
    }

    bool takeFee = true;
    //if any account belongs to _isExcludedFromFee account then remove the fee
    if(_isExcludedFromFee[from] || _isExcludedFromFee[to])
    {
```

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.

```
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 setMarketingAddress(address _Address) external onlyOwner()
{
    marketingAddress = payable(_Address);
}
```

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 swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap
{
    swapTokensForEth(contractTokenBalance);

    uint256 _totalFees = _marketingFee+_charityFee+_stakingFee+_lotteryFee;

    uint256 newBalance = address(this).balance;

    uint256 marketingShare  = newBalance.mul(_marketingFee).div(_totalFees);
    marketingAddress.transfer(marketingShare);

    uint256 charityShare  = newBalance.mul(_charityFee).div(_totalFees);
    charityAddress.transfer(charityShare);

    uint256 stakingShare  = newBalance.mul(_stakingFee).div(_totalFees);
    stakingAddress.transfer(stakingShare);

    uint256 lotteryShare  = newBalance.sub(marketingShare).sub(charityShare).sub(stakingShar
    lotteryAddress.transfer(lotteryShare);
}
```

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.

Divide before multiply

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

```
uint256 public _maxTxAmount = _tTotal.div(100).mul(5); //5%
```

Recommendation

Consider ordering multiplication before division.

Exploit scenario

```
contract A {
    function f(uint n) public {
        coins = (oldSupply / n) * interest;
    }
}
```

If `n` is greater than `oldSupply`, `coins` will be zero. For example, with `oldSupply = 5; n = 10, interest = 2`, coins will be zero. If `(oldSupply * interest / n)` was used, `coins` would have been 1. In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

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

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setMaxTxAmount(uint256 _mount) external onlyOwner()
{
    require(_mount&gt;_tTotal.div(1000), "Too low Txn limit"); // Min 0.1%
    _maxTxAmount = _mount;
}
```

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 private _previousCharityFee = _charityFee;
```

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://gith
    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.

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

Costly operations inside a loop

Costly operations inside a loop might waste gas, so optimizations are justified.

```
function includeInReward(address account) external onlyOwner() {
    require(_isExcluded[account], "Account is already excluded");
    for (uint256 i = 0; i &lt; _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

Recommendation

Use a local variable to hold the loop computation result.

Exploit scenario

```
contract CostlyOperationsInLoop{

    function bad() external{
        for (uint i=0; i < loop_count; i++){
            state_variable++;
        }
    }

    function good() external{
      uint local_variable = state_variable;
      for (uint i=0; i < loop_count; i++){
        local_variable++;
      }
      state_variable = local_variable;
    }
}
```

Incrementing `state_variable` in a loop incurs a lot of gas because of expensive `SSTORE`s, which might lead to an `out-of-gas`.

Owner privileges

- Owner cannot set fees higher than 25%
- Owner can change max transaction amount
- Owner can exclude from fees
- Owner can pause the contract

Extra notes by the team

No notes

Contract Snapshot

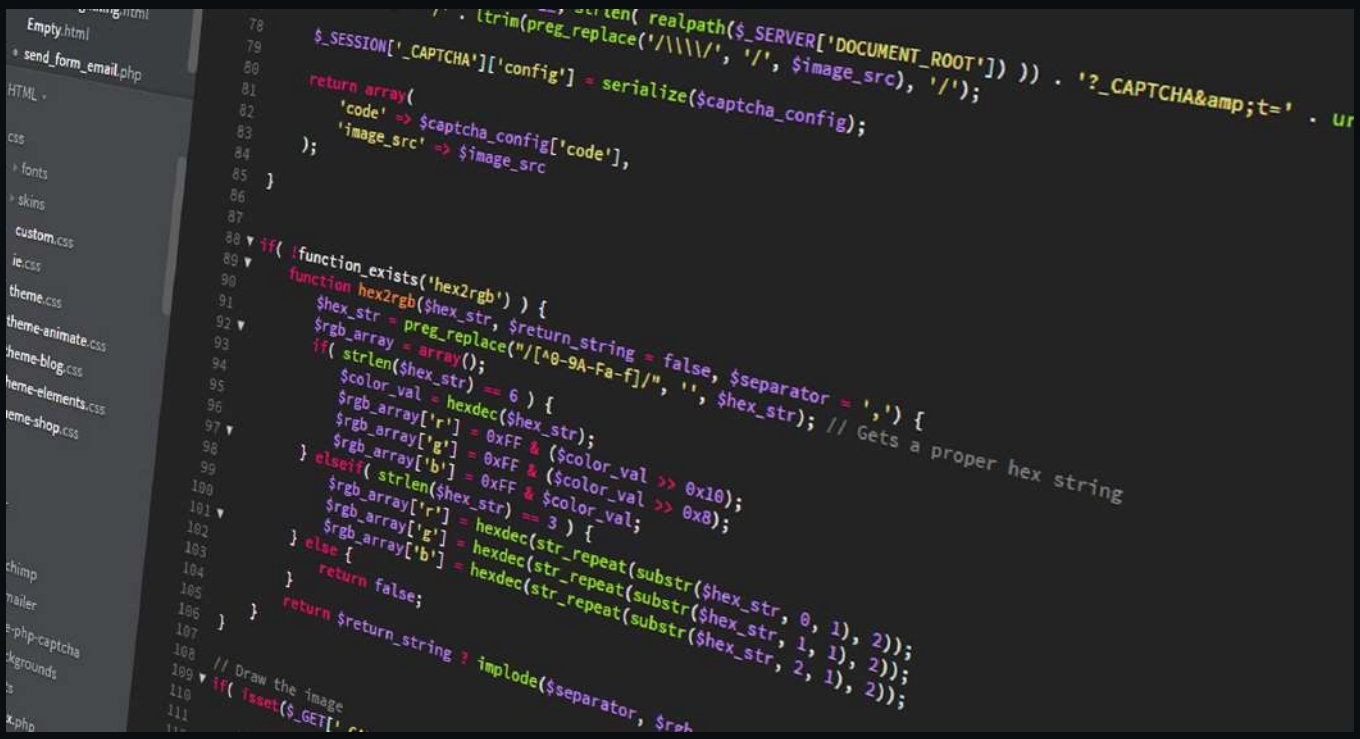
```
contract TreeToken is Context, IBEP20, LockToken
{
    using SafeMath for uint256;
    using Address for address;

    address payable public marketingAddress = payable(0x608427C75256F800f3EBa463c00c695Ce5DD
    address payable public charityAddress = payable(0x7A824cf7903AbA6f7240d6D15138C232530Da3
    address payable public stakingAddress = payable(0xa7C017194C164DF4aB1362a94a9ebb204718cd
    address payable public lotteryAddress = payable(0x94BDF3A04D676171D34fF9094d894729e242e9

    mapping (address =&gt; uint256) private _rOwned;
    mapping (address =&gt; uint256) private _tOwned;
    mapping (address =&gt; mapping (address =&gt; uint256)) private _allowances;
    mapping (address =&gt; bool) private _isExcludedFromWhale;
    mapping (address =&gt; bool) private _isExcludedFromFee;
    mapping (address =&gt; bool) private _isExcluded;
    mapping (address =&gt; bool) private _isExemptFromTxLimit;
    address[] private _excluded;
```

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

- KYC verified by Coinsult