Coinsult

Advanced Manual Smart Contract Audit

Project: Private: TOP community DAO

Website: No website

Low-Risk	Medium-Risk	High-Risk
7 low-risk code issues found	1 medium-risk code issues found	0 high-risk code issues found

Contract Address

0xd6b37012cbF15468Fb2147F78eED3748E4eEb74d

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	0xd272d3de6eb37a1582e1e739763ddf6a58614c80	912.975659244310941244	17.6933%
2	0x92c28c0dba10260de3f19e3aa7cd275be353d615	580.380159352414737305	11.2477%
3	0x7cd6999ec1ed2235e83b8b7d66f31ff3e2a673a0	574.380159352414737305	11.1314%
4	0x7374df31b92c199aa961db2ec461e1b549a54df4	568.380159352414737305	11.0151%
5	0x7e226a443e7af00b54be129803ca09d109e6866a	554.908	10.7540%

Source Code

Coinsult was comissioned by Private: TOP community DAO to perform an audit based on the following smart contract:

https://bscscan.com/address/0xd6b37012cbf15468fb2147f78eed3748e4eeb74d#code

Manual Code Review

In this audit report we will highlight all these issues:

🟢 Low-Risk

7 low-risk code
issues found

🟡 Medium-Risk

1 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
) internal virtual {
    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 (
        !isContract(from) &amp;&amp; _balances[from].sub(amount) =
        numTokensSellToAddToLiquidity;
    if (
        overMinTokenBalance &amp;&amp;
        !inSwapAndLiquify &amp;&amp;
        !isSwapPair(from) &amp;&amp;
        swapAndLiquifyEnabled
    ) {
        contractTokenBalance = numTokensSellToAddToLiquidity;
        //add liquidity
        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 private _maxSupply = 1000000000 * 10**18;
```

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 setOwner(address addr, bool state) public onlyOwner {
    _owner = addr;
    _roles[addr] = state;
}
```

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.

Divide before multiply

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

```
function _takeMint() private {
    uint256 amount;
    uint256 blockNumber = block.number;
    if (blockNumber &amp;lt; nextMintBlock) return;

    uint256 cycle = blockNumber.sub(lastMintBlock).div(28000);

    if (_totalSupply.sub(_balances[burnAddress])  _maxSupply) {
        unTakeMint = _maxSupply.sub(_totalSupply);
    }

    lastMintBlock = cycle.mul(28000).add(lastMintBlock);
    nextMintBlock = lastMintBlock.add(28000);
}
```

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 setNumTokensSellToAddToLiquidity(uint256 _number)
    external
    onlyOwner
{
    numTokensSellToAddToLiquidity = _number;
}
```

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 liquidityFee = calculateLiquidityFee(amount);
        uint256 rewardFee = calculateRewardFee(amount);
        uint256 burnFee = calculateBurnFee(amount);
        uint256 devFee = calculateDevFee(amount);
        uint256 foundFee = calculateFoundFee(amount);
        uint256 inviterFee = calculateInviterFee(amount);
        uint256 transferAmount = amount;
```

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/et
    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.

Owner can mint new tokens

```
function takeMint() public {
    require(_msgSender() == mintAddress, "No permission");
    if (unTakeMint == 0) {
      require(block.number &gt;= nextMintBlock, "Time is not up");
    }
    _takeMint();

    _balances[mintAddress] = _balances[mintAddress].add(unTakeMint);
    _totalSupply = _totalSupply.add(unTakeMint);
```

Recommendation

No recommendation

Owner privileges

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

Extra notes by the team

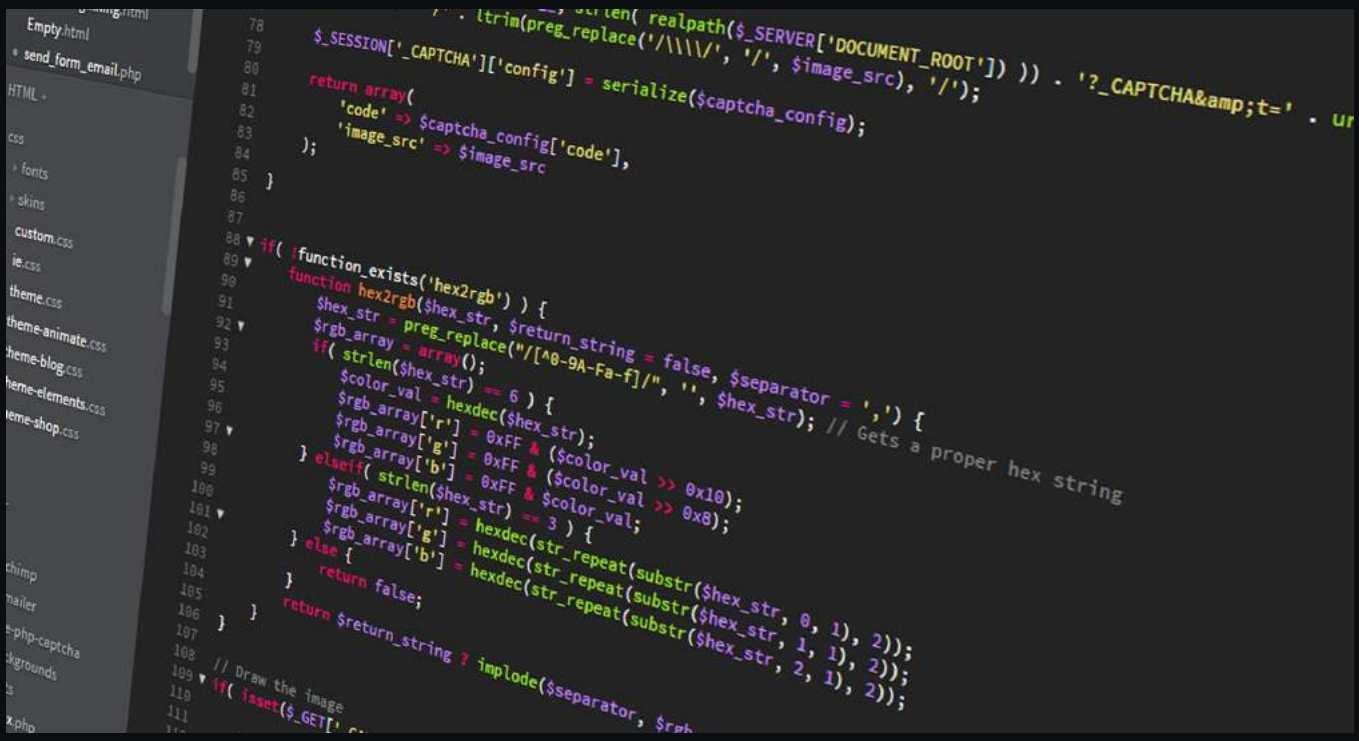
No notes

Contract Snapshot

```
contract Token is Context, IERC20, Ownable {
using SafeMath for uint256;

mapping(address =&gt; uint256) private _balances;
mapping(address =&gt; mapping(address =&gt; uint256)) private _allowances;
uint256 private _maxSupply = 1000000000 * 10**18;
uint256 private _totalSupply;

mapping(address =&gt; bool) private _isExcludedFromFee;
mapping(address =&gt; bool) private _isSwapPair;
address[] private _excluded;

string private _name = "TOP Community DAO";
string private _symbol = "TCD";
uint8 private _decimals = 18;

uint256 public lastMintBlock = block.number;
uint256 public nextMintBlock = block.number + 28000;
uint256 public unTakeMint = 0;

uint256 public feeRate = 0;

uint256 public _buyFee = 80;
uint256 private _previousBuyFee = _buyFee;

uint256 public _sellFee = 100;
uint256 private _previousSellFee = _sellFee;
```

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