Coinsult

# Advanced Manual Smart Contract Audit

**Project:** MVCC Token

**Website:** https://mvcc.co/

| Low-Risk | Medium-Risk | High-Risk |
| --- | --- | --- |
| 5 low-risk code issues found | 0 medium-risk code issues found | 0 high-risk code issues found |

**Contract Address**

0x80fB0Ff63B89CcdAEd3f6198aD86898845681C89

# 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 | 0x1c13a1a23094c4cb1c92df2fa8f6651d60660266 | 10,000,000 | 100.0000% |

# Source Code

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

https://bscscan.com/address/0x80fB0Ff63B89CcdAEd3f6198aD86898845681C89#code

# Manual Code Review

In this audit report we will highlight all these issues:

## 🟢 Low-Risk

5 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
) internal {
    require(from != address(0), "BEP20: transfer from the zero address");
    require(to != address(0), "BEP20: transfer to the zero address");
    require(amount &gt; 0, "Transfer amount must be greater than zero");

    uint256 contractTokenBalance = balanceOf(address(this));

    bool isOverMinTokenBalance = contractTokenBalance &gt;= swapThreshold;
    if (
        isOverMinTokenBalance &amp;&amp;
        !_inSwapAndLiquify &amp;&amp;
        !_isExcludedFromAutoLiquidity[from] &amp;&amp;
        _swapAndLiquifyEnabled
    ) {
        swapAndLiquify(contractTokenBalance);
    }

    bool takeFee = true;
```

### 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 constant _tTotal = 10000000 * 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 setReferralFeeReceiver(address newReferralFeeReceiver) external onlyOwner {
    require(newReferralFeeReceiver != address(0), "Address zero");
    referralFeeReceiver = newReferralFeeReceiver;

    emit SetReferralFeeReceiver(newReferralFeeReceiver);
}

function setSuperNodeFeeReceiver(address newSuperNodeFeeReceiver) external onlyOwner {
    require(newSuperNodeFeeReceiver != address(0), "Address zero");
    superNodeFeeReceiver = newSuperNodeFeeReceiver;

    emit SetSuperNodeFeeReceiver(newSuperNodeFeeReceiver);
}
```

### 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.

### Unchecked transfer

The return value of an external transfer/transferFrom call is not checked.

```
function rescueToken(address tokenAddress, address to) external onlyOwner {
    uint256 contractBalance = IERC20(tokenAddress).balanceOf(address(this));
    IERC20(tokenAddress).transfer(to, contractBalance);
}
```

### 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.

### Costly operations inside a loop

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

```
function setIncludeInReward(address account) external onlyOwner {
    require(account != address(0), "Address zero");
    require(_isExcluded[account], "Account is not 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;
        }
    }

    emit SetIncludeInReward(account);
}
```

### 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 `SSTOREs`, which might lead to an `out-of-gas`.

## Owner privileges

- Owner cannot set fees higher than 25%
- Owner cannot pause trading
- Owner cannot change max transaction amount
- Owner can exclude from fees
- ⚠ Owner can change uniswap router

## Extra notes by the team

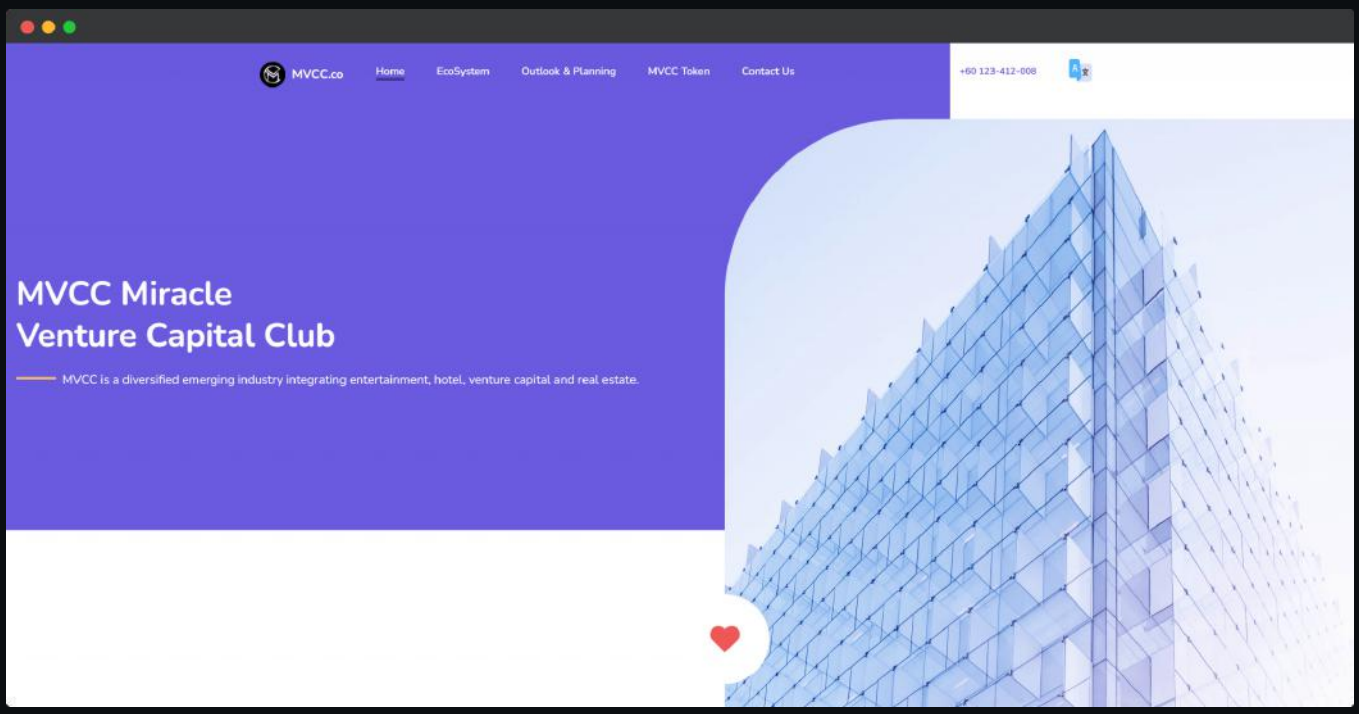
No notes

# Contract Snapshot

```
contract MVCC is Context, IERC20, Ownable {
string private constant _name = "MVCC TOKEN";
string private constant _symbol = "MVCC";
uint8 private constant _decimals = 18;

uint256 private constant MAX = ~uint256(0);
uint256 private constant _tTotal = 10000000 * 10**18;
uint256 private _rTotal = (MAX - (MAX % _tTotal));
uint256 private _tFeeTotal;

address[] private _excluded;
address public referralFeeReceiver;
address public superNodeFeeReceiver;

uint256 public referralFee = 5;
uint256 public superNodeFee = 2;
uint256 public liquidityFee = 2;
uint256 public taxFee = 1; // reflection
uint256 public swapThreshold = _tTotal * 1/1000; //0.1%;

// auto liquidity
bool public _swapAndLiquifyEnabled = true;
bool _inSwapAndLiquify;

IPancakeRouter02 public _pancakeRouter;
address public _pancakePair;

mapping (address =&gt; uint256) private _rOwned;
mapping (address =&gt; uint256) private _tOwned;
mapping (address =&gt; bool) private _isExcludedFromFee;
mapping (address =&gt; bool) private _isExcluded;
mapping (address =&gt; mapping (address =&gt; uint256)) private _allowances;
mapping (address =&gt; bool) public _isExcludedFromAutoLiquidity;

modifier lockTheSwap {
    _inSwapAndLiquify = true;
    _;
```

## 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