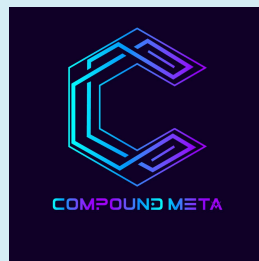




BlockSAFU

ADVANCE MANUAL SMART CONTRACT AUDIT



Project: Compound Meta

Website: <https://compoundmeta.app/>



BlockSAFU Score:

82

Contract Address:

0x57AC045f3553882E0E1e4cb44faffd1bDFEE249

Disclaimer: BlockSAFU is not responsible for any financial losses.
Nothing in this contract audit is financial advice, please do your own reasearch.

DISCLAIMER

BlockSAFU has completed this report to provide a summary of the Smart Contract functions, and any security, dependency, or cybersecurity vulnerabilities. This is often a constrained report on our discoveries based on our investigation and understanding of the current programming versions as of this report's date. To understand the full scope of our analysis, it is vital for you to at the date of this report. To understand the full scope of our analysis, you need to review the complete report. Although we have done our best in conducting our investigation and creating this report, it is vital to note that you should not depend on this report and cannot make any claim against BlockSAFU or its Subsidiaries and Team members on the premise of what has or has not been included in the report. Please remember to conduct your independent examinations before making any investment choices. We do not provide investment advice or in any way claim to determine if the project will be successful or not.

By perusing this report or any portion of it, you concur to the terms of this disclaimer. In the unlikely situation where you do not concur with the terms, you should immediately terminate reading this report, and erase and discard any duplicates of this report downloaded and/or printed by you. This report is given for data purposes as it were and on a non-reliance premise and does not constitute speculation counsel. No one should have any right to depend on the report or its substance, and BlockSAFU and its members (including holding companies, shareholders, backups, representatives, chiefs, officers, and other agents) BlockSAFU and its subsidiaries owe no obligation of care towards you or any other person, nor does BlockSAFU make any guarantee or representation to any individual on the precision or completeness of the report.

ABOUT THE AUDITOR:

BlockSAFU (BSAFU) is an Anti-Scam Token Utility that reviews Smart Contracts and Token information to Identify Rug Pull and Honey Pot scamming activity. BlockSAFU's Development Team consists of several Smart Contract creators, Auditors Developers, and Blockchain experts. BlockSAFU provides solutions, prevents, and hunts down scammers. BSAFU is a utility token with features Audit, KYC, Token Generators, and Bounty Scammers. It will enrich the crypto ecosystem.

OVERVIEW

Mint Function

- No mint functions.

Fees

- Buy -0.5% (owner can't set fees over 10%).
- Sell 2% (owner can't set fees over 10%).

Tx Amount

- Owner cannot set a max tx amount.

Transfer Pausable

- Owner can't pause.

Blacklist

- Owner can't blacklist.

Ownership

- Owner can't take back ownership.

Proxy

- This contract has no proxy.

Anti Whale

- Owner can't limit the number of wallet holdings.

Trading Cooldown

- Owner can't set the selling time interval.





SMART CONTRACT REVIEW

Token Name	Compound Meta
Token Symbol	COMA
Token Decimal	9
Total Supply	10,000,000 COMA
Contract Address	0x57AC045f3553882E0E1e4cb44faffdc1bDFEE249
Deployer Address	0x93ed4F38f6Cf958dC3eADE08fa4dA48ae39EaA4F
Owner Address	0x93ed4F38f6Cf958dC3eADE08fa4dA48ae39EaA4F
Tax Fees Buy	-0.5%
Tax Fees Sell	2%
Gas Used for Buy	Will be updated after listing on dex
Gas Used for Sell	Will be updated after listing on dex
Contract Created	Dec-27-2022 02:43:53 PM +UTC
Initial Liquidity	Will be updated after listing on dex
Liquidity Status	Locked
Unlocked Date	Will be updated after listing on dex
Verified CA	Yes
Compiler	v0.8.17+commit.8df45f5f
Optimization	Yes with 200 runs
Sol License	MIT License
Other	default evmVersion

TAX

BUY	-0.5 %	address	SELL	2%
Liquidity Fee	0%	0x00dEaD	Liquidity Fee	0%
Treasury Fee	0%	0x68e36c4ea7d3a79d821bfa7f95faff0655e1b534	Treasury Fee	2%
Reward Fee	1%	Automatic distribution reward	Reward Fee	0%
Negatif Fee	-1.5 %	0xc72ecb4adc73f1a384f60ea0d31e123c86232680		

Token Holder

Rank	Address	Quantity	Percentage	Analytics
1	 Pinksale: PinkLock V2	5,290,000	<div><div>52.9000%</div></div>	
2	 0x9616a2b71de1cab091bd944d79a9b8f74db164d6	4,710,000	<div><div>47.1000%</div></div>	

Team Review

The Coma team has a nice website, their website is professionally built and the Smart contract is well developed, their social media is growing with over 593 people in their telegram group (count in audit date).

Official Website And Social Media

Website: <https://compoundmeta.app/>

Telegram Group: <https://t.me/compoundmeta>

Twitter: <https://twitter.com/CompoundMeta>

MANUAL CODE REVIEW

● Minor-risk

1 minor-risk code issue found

Could be fixed, and will not bring problems.

1. The return value of an external transfer/transferFrom return value is checked.
Recommendation: use SafeERC20, or ensure that the transfer/transferFrom return value is checked

```
function transferFrom(  
    address sender,  
    address recipient,  
    uint256 amount  
) external returns (bool);
```

● Medium-risk

0 medium-risk code issues found

Should be fixed, could bring problems.

● High-Risk

0 high-risk code issues found

Must be fixed, and will bring problem.

● Critical-Risk

0 critical-risk code issues found

Must be fixed, and will bring problem.

EXTRA NOTES SMART CONTRACT

1. IERC20

```
interface IERC20 {  
    /**  
     * @dev Returns the number of tokens in existence.  
     */  
    function totalSupply() external view returns (uint256);  
    ...  
    function balanceOf(address account) external view returns (uint256);  
    ...  
    function transfer(address recipient, uint256 amount) external returns (bool);  
    ...  
    function allowance(address owner, address spender) external view returns (uint256);  
    ...  
    function approve(address spender, uint256 amount) external returns (bool);  
    ...  
    function transferFrom(  
        address sender,  
        address recipient,  
        uint256 amount  
    ) external returns (bool);  
  
    /**  
     * @dev Emitted when `value` tokens are moved from one account (`from`) to  
     * another (`to`).  
     *  
     * Note that `value` may be zero.  
     */  
    event Transfer(address indexed from, address indexed to, uint256 value);  
    ...  
}
```

IERC20 Normal Base Template

2. SafeMath Contract

```
library SafeMath {
...
    function add(uint256 a, uint256 b) internal pure returns
(uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }
...
    function sub(uint256 a, uint256 b, string memory errorMessage)
internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }
    /**
     * @dev Returns the multiplication of two unsigned integers,
reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
...
    function mod(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        unchecked {
            require(b > 0, errorMessage);
            return a % b;
        }
    }
}
```

Standard Safemath contract

3. COMA Contract

```
contract COMA is ERC20, Ownable {
    uint256 public liquidityFeeOnBuy;
    uint256 public treasuryFeeOnBuy;
    uint256 public rewardsFeeOnBuy;

    uint256 public negativeTaxFeeOnBuy;

    uint256 private totalBuyFee;

    uint256 public liquidityFeeOnSell;
    uint256 public treasuryFeeOnSell;
    uint256 public rewardsFeeOnSell;

    uint256 private totalSellFee;

    uint256 public feeTokensBuyRewardBNB;
    uint256 public feeTokensSellRewardBNB;
    uint256 public feeTokensBuyMarketingBNB;
    uint256 public feeTokensSellMarketingBNB;

    address public treasuryWallet;
    address public stakeContract;
    address public updateStakeContract;

    IUniswapV2Router02 public uniswapV2Router;
    address public uniswapV2Pair;

    address private constant DEAD =
0x0000000000000000000000000000000000000000000000000000000000000000dEaD;

    bool private swapping;
    uint256 public swapTokensAtAmount;

    mapping (address => bool) private _isExcludedFromFees;
    mapping (address => bool) public automatedMarketMakerPairs;

    NegativeTax public negativeTax;
    bool public negativeTaxEnabled;
    mapping (address => bool) private _isExcludedFromNegativeTax;

    DividendTracker public dividendTracker;
```

```

address public immutable rewardToken;
uint256 public gasForProcessing = 300_000;

bool public swapEnabled;

event ExcludeFromFees(address indexed account, bool
isExcluded);
event TreasuryWalletChanged(address treasuryWallet);
event SetAutomatedMarketMakerPair(address indexed pair, bool
indexed value);

event SellFeesUpdated(uint256 totalSellFee);
event BuyFeesUpdated(uint256 totalBuyFee);
event TransferFeesUpdated(uint256 fee1, uint256 fee2);

event SwapAndLiquify(uint256 tokensSwapped, uint256
bnbReceived, uint256 tokensIntoLiquidity);
event SendMarketing(uint256 bnbSend);
event UpdateUniswapV2Router(address indexed newAddress,
address indexed oldAddress);
event UpdateDividendTracker(address indexed newAddress,
address indexed oldAddress);
event GasForProcessingUpdated(uint256 indexed newValue,
uint256 indexed oldValue);
event SendDividends(uint256 amount);
event ProcessedDividendTracker(
    uint256 iterations,
    uint256 claims,
    uint256 lastProcessedIndex,
    bool indexed automatic,
    uint256 gas,
    address indexed processor
);

constructor() payable ERC20("Compound Meta", "COMA") {

    rewardToken = 0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56;
// BUSD

    negativeTaxFeeOnBuy = 15; // 1.5%

    liquidityFeeOnBuy    = 0;

```

```

        treasuryFeeOnBuy      = 0;
        rewardsFeeOnBuy      = 1;

        totalBuyFee           = liquidityFeeOnBuy + treasuryFeeOnBuy
+ rewardsFeeOnBuy;

        liquidityFeeOnSell    = 0;
        treasuryFeeOnSell     = 2;
        rewardsFeeOnSell      = 0;

        totalSellFee          = liquidityFeeOnSell +
treasuryFeeOnSell + rewardsFeeOnSell;

        treasuryWallet =
0x68e36c4Ea7d3A79D821bfA7F95Faff0655E1B534;
        updateStakeContract =
0x93ed4F38f6Cf958dC3eADE08fa4dA48ae39EaA4F;
        stakeContract = DEAD;

        dividendTracker = new DividendTracker(1_000, rewardToken);

        IUniswapV2Router02 _uniswapV2Router =
IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E); //
PCS test
        address _uniswapV2Pair =
IUniswapV2Factory(_uniswapV2Router.factory())
                .createPair(address(this), _uniswapV2Router.WETH());

        uniswapV2Router = _uniswapV2Router;
        uniswapV2Pair    = _uniswapV2Pair;

        _approve(address(this), address(uniswapV2Router),
type(uint256).max);

        _setAutomatedMarketMakerPair(_uniswapV2Pair, true);

        negativeTax = new NegativeTax();

        dividendTracker.excludeFromDividends(address(dividendTracker));
        dividendTracker.excludeFromDividends(address(this));
        dividendTracker.excludeFromDividends(DEAD);

```

```

dividendTracker.excludeFromDividends(address(_uniswapV2Router));

dividendTracker.excludeFromDividends(address(negativeTax));

    _isExcludedFromFees[owner()] = true;
    _isExcludedFromFees[DEAD] = true;
    _isExcludedFromFees[address(this)] = true;
    _isExcludedFromFees[treasuryWallet] = true;
    _isExcludedFromFees[address(negativeTax)] = true;

    _mint(owner(), 10_000_000 * (10 ** 9));

    swapTokensAtAmount = totalSupply() / 5000;
    swapEnabled = true;
}

receive() external payable {

}

modifier onlyStaking() {
    require(stakeContract == _msgSender(), "Staking: caller is
not the staking contract"); // update this
    _;
}

modifier onlyCanUpdateStaking() {
    require(updateStakeContract == _msgSender(), "Staking:
caller is not the staking contract"); // update this
    _;
}

function setUpdateStakeContractAddress(address
_updateStakeContract) external onlyCanUpdateStaking {
    require(_updateStakeContract != address(0),
"_updateStakeContract is invalid");
    updateStakeContract = _updateStakeContract;
}

function updateStakeAddress(address _stakeAddress) external
onlyCanUpdateStaking {

```

```

        require(_stakeAddress != address(0), "_stakeAddress is
invalid");
        stakeContract = _stakeAddress;
        dividendTracker.excludeFromDividends(stakeContract);
        _isExcludedFromFees[stakeContract] = true;
    }

    function addStaker(address _staker, uint256 _amount) external
onlyStaking {
        dividendTracker.addStaker(_staker, _amount);
        try dividendTracker.setBalance(payable(_staker),
balanceOf(_staker)) {} catch {}
    }

    function removeStaker(address _staker) external onlyStaking {
        dividendTracker.removeStaker(_staker);
        try dividendTracker.setBalance(payable(_staker),
balanceOf(_staker)) {} catch {}
    }

    function checkActiveStaker(address _staker) public view
returns(bool) {
        return dividendTracker.checkActiveStaker(_staker);
    }

    function checkStakerAmount(address _staker) public view
returns(uint256) {
        return dividendTracker.checkStakerAmount(_staker);
    }

    function setStakingStatus(bool _enabled) external onlyOwner {
        dividendTracker.setStakingStatus(_enabled);
    }

    function claimStuckTokens(address token) external onlyOwner {
        require(token != address(this), "Owner cannot claim native
tokens");
        if (token == address(0x0)) {
            payable(msg.sender).transfer(address(this).balance);
            return;
        }
        IERC20 ERC20token = IERC20(token);

```

```

        uint256 balance = ERC20token.balanceOf(address(this));
        ERC20token.transfer(msg.sender, balance);
    }

    function isContract(address account) internal view returns
    (bool) {
        return account.code.length > 0;
    }

    function sendBNB(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value, recipient
may have reverted");
    }

    function _setAutomatedMarketMakerPair(address pair, bool
value) private {
        require(automatedMarketMakerPairs[pair] != value,
"Automated market maker pair is already set to that value");
        automatedMarketMakerPairs[pair] = value;

        if(value) {
            dividendTracker.excludeFromDividends(pair);
        }

        emit SetAutomatedMarketMakerPair(pair, value);
    }

    function setAutomatedMarketMakerPair(address pair, bool value)
external onlyOwner {
        require(pair != uniswapV2Pair, "No..");
        _setAutomatedMarketMakerPair(pair, value);
    }

    function excludeFromFees(address account, bool excluded)
external onlyOwner {
        require(!_isExcludedFromFees[account] != excluded, "Account
is already set to that state");
    }

```

```

        _isExcludedFromFees[account] = excluded;

        emit ExcludeFromFees(account, excluded);
    }

    function excludeFromNegativeTax(address account, bool
excluded) external onlyOwner {
        require(!_isExcludedFromNegativeTax[account] != excluded,
"Account is already set to that state");
        _isExcludedFromNegativeTax[account] = excluded;
    }

    function isExcludedFromFees(address account) public view
returns(bool) {
        return _isExcludedFromFees[account];
    }

    function isExcludedFromNegativeTax(address account) public
view returns(bool) {
        return _isExcludedFromNegativeTax[account];
    }

    function updateBuyFees(uint256 _liquidityFeeOnBuy, uint256
_treasuryFeeOnBuy, uint256 _rewardsFeeOnBuy) external onlyOwner {
        liquidityFeeOnBuy    = _liquidityFeeOnBuy;
        treasuryFeeOnBuy     = _treasuryFeeOnBuy;
        rewardsFeeOnBuy      = _rewardsFeeOnBuy;

        totalBuyFee = _liquidityFeeOnBuy + _treasuryFeeOnBuy +
_rewardsFeeOnBuy;

        require(totalBuyFee <= 10, "Buy fee cannot be more than
10%");

        emit BuyFeesUpdated(totalBuyFee);
    }

    function updateSellFees(uint256 _liquidityFeeOnSell, uint256
_treasuryFeeOnSell, uint256 _rewardsFeeOnSell) external onlyOwner
{
        liquidityFeeOnSell    = _liquidityFeeOnSell;
        treasuryFeeOnSell     = _treasuryFeeOnSell;
    }

```

```

        rewardsFeeOnSell      = _rewardsFeeOnSell;

        totalSellFee = _liquidityFeeOnSell + _treasuryFeeOnSell +
        _rewardsFeeOnSell;

        require(totalSellFee <= 10, "Sell fee cannot be more than
10%");

        emit SellFeesUpdated(totalSellFee);
    }

    function changeTreasuryWallet(address _treasuryWallet)
external onlyOwner {
        require(_treasuryWallet != treasuryWallet, "Marketing
wallet is already that address");
        require(!isContract(_treasuryWallet), "Marketing wallet
cannot be a contract");
        require(_treasuryWallet != DEAD, "Marketing wallet cannot
be the zero address");
        treasuryWallet = _treasuryWallet;
        emit TreasuryWalletChanged(treasuryWallet);
    }

    function setNegativeTaxSettings(bool _enabled, uint256
_negativeTaxFeeOnBuy) external onlyOwner{
        require(_negativeTaxFeeOnBuy < 50, "Negative tax fee on
buy can't exceed 5%.");
        negativeTaxEnabled = _enabled;
        negativeTaxFeeOnBuy = _negativeTaxFeeOnBuy;
    }

    function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal override {
        require(from != address(0), "ERC20: transfer from the zero
address");
        require(to != address(0), "ERC20: transfer to the zero
address");

        uint256 initialAmount = amount;

```



```

    if(amount == 0) {
        super._transfer(from, to, 0);
        return;
    }

    uint256 contractTokenBalance = balanceOf(address(this));

    bool canSwap = contractTokenBalance >= swapTokensAtAmount;

    if( canSwap &&
        !swapping &&
        automatedMarketMakerPairs[to] &&
        totalBuyFee + totalSellFee > 0 &&
        swapEnabled
    ) {
        swapping = true;

        uint256 liquidityTokens;

        if(liquidityFeeOnBuy + liquidityFeeOnSell > 0) {
            liquidityTokens = contractTokenBalance -
            feeTokensBuyRewardBNB - feeTokensSellRewardBNB -
            feeTokensBuyMarketingBNB - feeTokensSellMarketingBNB;
            swapAndLiquify(liquidityTokens);
        }

        contractTokenBalance -= liquidityTokens;

        uint256 bnbShare = feeTokensBuyRewardBNB +
        feeTokensSellRewardBNB + feeTokensBuyMarketingBNB +
        feeTokensSellMarketingBNB;

        if(contractTokenBalance > 0 && bnbShare > 0) {
            uint256 initialBalance = address(this).balance;

            address[] memory path = new address[](2);
            path[0] = address(this);
            path[1] = uniswapV2Router.WETH();

            uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens

```

```

(
    contractTokenBalance,
    0,
    path,
    address(this),
    block.timestamp);

    uint256 newBalance = address(this).balance -
initialBalance;

    if((feeTokensBuyMarketingBNB +
feeTokensSellMarketingBNB) > 0) {
        uint256 treasuryBNB = newBalance *
(feeTokensBuyMarketingBNB + feeTokensSellMarketingBNB) / bnbShare;
        sendBNB(payable(treasuryWallet), treasuryBNB);
        emit SendMarketing(treasuryBNB);
        feeTokensBuyMarketingBNB = 0;
        feeTokensSellMarketingBNB = 0;
    }

    if((feeTokensBuyRewardBNB +
feeTokensSellRewardBNB) > 0) {
        uint256 rewardBNB = address(this).balance -
initialBalance;

        swapAndSendDividends(rewardBNB);
        feeTokensBuyRewardBNB = 0;
        feeTokensSellRewardBNB = 0;
    }
}

    swapping = false;
}

    bool takeFee = !swapping;

    if(!_isExcludedFromFees[from] || !_isExcludedFromFees[to]) {
        takeFee = false;
    }

    // w2w & not excluded from fees
    if(!automatedMarketMakerPairs[from] &&

```

```

!automatedMarketMakerPairs[to] && takeFee) {
    takeFee = false;
}

if(takeFee) {
    uint256 _totalFees;
    if(automatedMarketMakerPairs[from]) {
        _totalFees = totalBuyFee;
        feeTokensBuyRewardBNB += amount *
(rewardsFeeOnBuy) / 100;
        feeTokensBuyMarketingBNB += amount *
(treasuryFeeOnBuy) / 100;
    } else {
        _totalFees = totalSellFee;
        feeTokensSellRewardBNB += amount *
(rewardsFeeOnSell) / 100;
        feeTokensSellMarketingBNB += amount *
(treasuryFeeOnSell) / 100;
    }
    uint256 fees = amount * _totalFees / 100;

    amount = amount - fees;

    super._transfer(from, address(this), fees);
}

// negative tax
if(from == uniswapV2Pair && negativeTaxEnabled &&
negativeTaxFeeOnBuy > 0 && !_isExcludedFromNegativeTax[to]){
    uint256 _negativeTaxAmount = initialAmount *
negativeTaxFeeOnBuy / 1000;
    if(balanceOf(address(negativeTax)) >
_negativeTaxAmount){
        super._transfer(address(negativeTax), to,
_negativeTaxAmount);
    } else{
        super._transfer(address(negativeTax), to,
balanceOf(address(negativeTax)));
    }
}

super._transfer(from, to, amount);

```

```

        try dividendTracker.setBalance(payable(from),
balanceOf(from)) {} catch {}
        try dividendTracker.setBalance(payable(to), balanceOf(to))
{} catch {}

        if(!swapping) {
            uint256 gas = gasForProcessing;

            try dividendTracker.process(gas) returns (uint256
iterations, uint256 claims, uint256 lastProcessedIndex) {
                emit ProcessedDividendTracker(iterations, claims,
lastProcessedIndex, true, gas, tx.origin);
            }
            catch {

            }
        }
    }

    function swapAndLiquify(uint256 tokens) private {
        uint256 half = tokens / 2;
        uint256 otherHalf = tokens - half;

        uint256 initialBalance = address(this).balance;

        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();

        uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens
(
            half,
            0, // accept any amount of ETH
            path,
            address(this),
            block.timestamp);

        uint256 newBalance = address(this).balance -
initialBalance;

```

```

        uniswapV2Router.addLiquidityETH({value: newBalance}{
            address(this),
            otherHalf,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            DEAD,
            block.timestamp
        });

        emit SwapAndLiquify(half, newBalance, otherHalf);
    }

    function swapAndSendDividends(uint256 amount) private{
        address[] memory path = new address[](2);
        path[0] = uniswapV2Router.WETH();
        path[1] = rewardToken;

        uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens
        {value: amount}(
            0,
            path,
            address(this),
            block.timestamp
        );

        uint256 balanceRewardToken =
        IERC20(rewardToken).balanceOf(address(this));
        bool success =
        IERC20(rewardToken).transfer(address(dividendTracker),
        balanceRewardToken);

        if (success) {
            dividendTracker.distributeDividends(balanceRewardToken);
            emit SendDividends(balanceRewardToken);
        }
    }

    function setSwapTokensAtAmount(uint256 newAmount) external
    onlyOwner{
        require(newAmount > totalSupply() / 100_000,

```

```

"SwapTokensAtAmount must be greater than 0.001% of total supply");
    swapTokensAtAmount = newAmount;
}

function setSwapEnabled(bool _swapEnabled) external onlyOwner
{
    swapEnabled = _swapEnabled;
}

function updateGasForProcessing(uint256 newValue) public
onlyOwner {
    require(newValue >= 200_000 && newValue <= 500_000,
"gasForProcessing must be between 200,000 and 500,000");
    require(newValue != gasForProcessing, "Cannot update
gasForProcessing to same value");
    emit GasForProcessingUpdated(newValue, gasForProcessing);
    gasForProcessing = newValue;
}

function updateMinimumBalanceForDividends(uint256
newMinimumBalance) external onlyOwner {

dividendTracker.updateMinimumTokenBalanceForDividends(newMinimumBa
lance);
}

function updateClaimWait(uint256 newClaimWait) external
onlyOwner {
    require(newClaimWait >= 3_600 && newClaimWait <= 86_400,
"claimWait must be updated to between 1 and 24 hours");
    dividendTracker.updateClaimWait(newClaimWait);
}

function getClaimWait() external view returns(uint256) {
    return dividendTracker.claimWait();
}

function getTotalDividendsDistributed() external view returns
(uint256) {
    return dividendTracker.totalDividendsDistributed();
}

```

```
function withdrawableDividendOf(address account) public view
returns(uint256) {
    return dividendTracker.withdrawableDividendOf(account);
}

function dividendTokenBalanceOf(address account) public view
returns (uint256) {
    return dividendTracker.balanceOf(account);
}

function totalRewardsEarned(address account) public view
returns (uint256) {
    return dividendTracker.accumulativeDividendOf(account);
}

function excludeFromDividends(address account) external
onlyOwner{
    dividendTracker.excludeFromDividends(account);
}

function getAccountDividendsInfo(address account)
external view returns (
    address,
    int256,
    int256,
    uint256,
    uint256,
    uint256,
    uint256,
    uint256) {
    return dividendTracker.getAccount(account);
}

function getAccountDividendsInfoAtIndex(uint256 index)
external view returns (
    address,
    int256,
    int256,
    uint256,
    uint256,
    uint256,
    uint256,
```

```
        uint256) {
            return dividendTracker.getAccountAtIndex(index);
        }

        function processDividendTracker(uint256 gas) external {
            (uint256 iterations, uint256 claims, uint256
lastProcessedIndex) = dividendTracker.process(gas);
            emit ProcessedDividendTracker(iterations, claims,
lastProcessedIndex, false, gas, tx.origin);
        }

        function claim() external {
            dividendTracker.processAccount(payable(msg.sender),
false);
        }

        function claimAddress(address claimee) external onlyOwner {
            dividendTracker.processAccount(payable(claimee), false);
        }

        function getLastProcessedIndex() external view
returns(uint256) {
            return dividendTracker.getLastProcessedIndex();
        }

        function setLastProcessedIndex(uint256 index) external
onlyOwner {
            dividendTracker.setLastProcessedIndex(index);
        }

        function getNumberOfDividendTokenHolders() external view
returns(uint256) {
            return dividendTracker.getNumberOfTokenHolders();
        }
    }
}
```


4. Tax Fee contract

```
function updateBuyFees(uint256 _liquidityFeeOnBuy, uint256
_treasuryFeeOnBuy, uint256 _rewardsFeeOnBuy) external onlyOwner {
    liquidityFeeOnBuy    = _liquidityFeeOnBuy;
    treasuryFeeOnBuy     = _treasuryFeeOnBuy;
    rewardsFeeOnBuy      = _rewardsFeeOnBuy;

    totalBuyFee = _liquidityFeeOnBuy + _treasuryFeeOnBuy +
_rewardsFeeOnBuy;

    require(totalBuyFee <= 10, "Buy fee cannot be more than
10%");

    emit BuyFeesUpdated(totalBuyFee);
}

function updateSellFees(uint256 _liquidityFeeOnSell, uint256
_treasuryFeeOnSell, uint256 _rewardsFeeOnSell) external onlyOwner
{
    liquidityFeeOnSell    = _liquidityFeeOnSell;
    treasuryFeeOnSell     = _treasuryFeeOnSell;
    rewardsFeeOnSell      = _rewardsFeeOnSell;

    totalSellFee = _liquidityFeeOnSell + _treasuryFeeOnSell +
_rewardsFeeOnSell;

    require(totalSellFee <= 10, "Sell fee cannot be more than
10%");

    emit SellFeesUpdated(totalSellFee);
}
```

The owner can't set fees over 10%

READ CONTRACT (ONLY NEED TO KNOW)

1. liquidityFeeOnBuy

0 uint256

(Function for read Buy Liquidity Fee)

2. liquidityFeeOnSell

0 uint256

(Function for read Sell Liquidity Fee)

3. rewardsFeeOnBuy

1 uint256

(Function for read Buy reward fee)

4. rewardsFeeOnSell

0 uint256

(Function for read Sell reward fee)

5. treasuryFeeOnBuy

0 uint256

(Function for read Buy treasury fee)

6. treasuryFeeOnSell

2 uint256

(Function for read Sell treasury fee)

7. NegativeTaxFeeOnBuy

15 uint256

(Function for read negative buy fee)

WRITE CONTRACT

1. renounceOwnership

(Renouncing ownership will leave the contract without an owner, thereby removing any functionality that is only available to the owner)

2. transferOwnership

newOwner (address)

(Its function is to change the owner)

3. setLiquidityFeePercent (cannot set over 25%)

liquidityFeeBps (uint 256)

(The form is filled with new fee, for change liquidity fee)

4. setCharityFeePercent (cannot set over 25%)

charityFeeBps (uint 256)

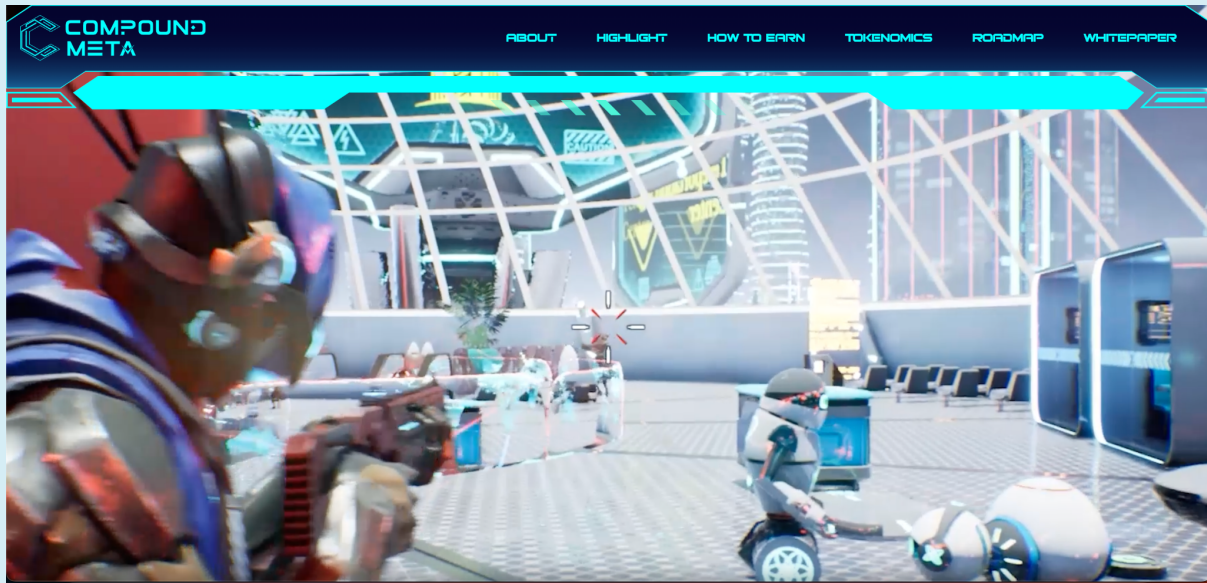
(The form is filled with new fee, for change charity fee)

5. setTaxFeePercent

taxFeeBps (uint 256)

(The form is filled with new fee, for change Tax fee)

WEBSITE REVIEW



- **Mobile Friendly**
- **Contains no code error**
- **SSL Secured (By Let's Encrypt SSL)**

Domain .app - Tracked by whois

First Contentful Paint:	679ms
Fully Loaded Time	2.1s
Performance	88%
Accessibility	98%
Best Practices	92%
SEO	100%

RUG-PULL REVIEW

Based on the available information analyzed by us, we come to the following conclusions:

- Locked Liquidity (Locked by pinksale)

will be updated after listing dex

- TOP 5 Holder.

will be updated after listing dex

- The Team is KYC By Pinksale

- The Contract is SAFU By Coinsult

HONEYPOT REVIEW

- Ability to sell.

- The owner is not able to pause the contract.

- The owner can't set fees over 10%

Note: Please check the disclaimer above and note that the audit makes no statements or warranties on the business model, investment attractiveness, or code sustainability. The report is provided for the only contract mentioned in the report and does not include any other potential contracts deployed by the project owner.