BlockSAFU

ADVANCE MANUAL SMART CONTRACT AUDIT

Project: Compound Meta Staking

Website: https://compoundmeta.app/

PASSED

BlockSAFU Score:

82

Contract Address:

0x8edc28E352dC48B7d5C7140f8e5EEbe5F335beAB

DISCLAMER

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. BlockSAFUs 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

- Stake 2% (owner can’t set fees over 5%).
- Withdraw 10% (owner can’t set fees over 20%).

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 Staking**
Contract Address	0x8edc28E352dC48B7d5C7140f8e5EEbe5F335beAB
Deployer Address	0xbF894C13aFbA110C800Ff9398E7155dafd698171
Owner Address	0x93ed4f38f6cf958dc3eade08fa4da48ae39eaa4f
Stake	2%
Withdraw	10%
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-28-2022 02:03:47 AM +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	v0.8.2+commit.661d1103
Optimization	No with 200 runs
Sol License	MIT License
Other	default evmVersion

TAX

Stake	2%		**Withdraw**	10%

Token Holder

Rank	Address	Quantity	Percentage	Analytics
1	Pinksale: PinkLock V2	5,290,000	52.9000%	
2	0x9616a2b71de1cab091bd944d79a9b8f74db164d8	4,710,000	47.1000%	

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

0 minor-risk code issue found

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. CompoundStake Contract

```
contract CompoundStake is Ownable {
      using EnumerableSet for EnumerableSet.AddressSet;
      using SafeMath for uint256;
      using SafeERC20 for IERC20;

      uint256 public constant ACC_PRECISION = 1e12;

      struct UserInfo {
            uint256 amount;
            uint256 rewardDebt;
            uint256 rewardAmount;
            uint256 lastStakeTime;
      }

      // Info of each pool.
      struct PoolInfo {
            IERC20 poolToken;
            uint256 allocPoint;
            uint256 lastRewardTime;
            uint256 accRewardPerShare;
            uint256 amount;
            uint256 lockDuration;
      }

      ERC20 public rewardToken;
      IDividendToken public dividendToken;
      uint256 public tokenPerSecond;
      uint256 public startTime;
      uint256 public stakeFee;
      uint256 public withdrawFee;
      address public rewardAddress;
      uint256 public constant taxFee = 10000;

      PoolInfo[] public poolInfo;
      mapping(uint256 => mapping(address => UserInfo)) public
userInfo;
      mapping(uint256 => EnumerableSet.AddressSet) private
poolUsers;

      uint256 public totalAllocPoint = 0;
```

```
    event Deposit(address indexed user, uint256 indexed pid,
uint256 amount);
    event Withdraw(address indexed user, uint256 indexed pid,
uint256 amount);
    event Harvest(address indexed user, uint256 indexed pid,
uint256 amount);

    constructor(
        address _rewardTokenAddress,
        uint256 _rewardPerBlock,
        uint256 _startTime,
        uint256 _stakeFee,
        uint256 _withdrawFee,
        address _rewardAddress
    ) {
        rewardToken = ERC20(_rewardTokenAddress);
        stakeFee = _stakeFee;
        withdrawFee = _withdrawFee;
        tokenPerSecond = _rewardPerBlock;
        startTime = _startTime;
        rewardAddress = _rewardAddress;
        dividendToken = IDividendToken(_rewardTokenAddress);
    }

    modifier verifyPoolId(uint256 _pid) {
        require(_pid < poolInfo.length, "Pool is not exist");
        _;
    }

    function poolLength() external view returns (uint256) {
        return poolInfo.length;
    }

    function getMultiplier(uint256 _from, uint256 _to) public
pure returns (uint256) {
        return _to.sub(_from);
    }

    function getUserStakeBalance(uint256 _pid, address _user)
external view returns (uint256) {
        return userInfo[_pid][_user].amount;
    }
```

```
	// View function to see pending BSLs on frontend.
	function getRewardAmount(uint256 _pid, address _user)
external view verifyPoolId(_pid) returns (uint256) {
		PoolInfo memory pool = poolInfo[_pid];
		UserInfo memory user = userInfo[_pid][_user];
		uint256 accRewardPerShare = pool.accRewardPerShare;
		uint256 lpSupply = pool.amount;
		if (block.timestamp > pool.lastRewardTime && lpSupply
!= 0) {
			uint256 multiplier =
getMultiplier(pool.lastRewardTime, block.timestamp);
			uint256 rewardAmount =
multiplier.mul(tokenPerSecond).mul(pool.allocPoint).div(totalAlloc
Point);
			accRewardPerShare =
accRewardPerShare.add(rewardAmount.mul(ACC_PRECISION).div(lpSupply
));
		}

		uint256 pendingAmount =
user.amount.mul(accRewardPerShare).div(ACC_PRECISION).sub(user.rew
ardDebt);
		return user.rewardAmount.add(pendingAmount);
	}

	// Update reward variables for all pools. Be careful of gas
spending!
	function massUpdatePools() public {
		for (uint256 pid = 0; pid < poolInfo.length; ++pid) {
			updatePool(pid);
		}
	}

	// Update reward variables of the given pool to be
up-to-date.
	function updatePool(uint256 _pid) public verifyPoolId(_pid)
{
		PoolInfo storage pool = poolInfo[_pid];
		if (block.timestamp <= pool.lastRewardTime) {
			return;
		}
```

```
            uint256 lpSupply = pool.amount;
            if (lpSupply == 0) {
                  pool.lastRewardTime = block.timestamp;
                  return;
            }
            uint256 multiplier = getMultiplier(pool.lastRewardTime,
block.timestamp);
            uint256 rewardAmount =
multiplier.mul(tokenPerSecond).mul(pool.allocPoint).div(totalAlloc
Point);

            pool.accRewardPerShare =
pool.accRewardPerShare.add(rewardAmount.mul(ACC_PRECISION).div(lpS
upply));
            pool.lastRewardTime = block.timestamp;
      }

      function stake(uint256 _pid, uint256 _amount) external
verifyPoolId(_pid)  {
            require(_amount > 0, "amount must be greater than 0");
            updatePool(_pid);

            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][msg.sender];

            // Update last reward
            uint256 pendingAmount =
user.amount.mul(pool.accRewardPerShare).div(ACC_PRECISION).sub(use
r.rewardDebt);
            if (pendingAmount > 0) {
                  user.rewardAmount =
user.rewardAmount.add(pendingAmount);
            }

            // Statistical
            if (user.amount == 0) {
                  poolUsers[_pid].add(msg.sender);
            }

            uint256 amountStake = _amount;
            if(stakeFee > 0){
                  uint256 feeStake =
```

```
amountStake.mul(stakeFee).div(taxFee);
                amountStake = amountStake.sub(feeStake);

pool.poolToken.safeTransferFrom(address(msg.sender),
rewardAddress, feeStake);
           }

           uint256 amountAfter = user.amount.add(amountStake);



           // Add LP
           pool.poolToken.safeTransferFrom(address(msg.sender),
address(this), amountStake);
           pool.amount = pool.amount.add(amountStake);
           user.amount = user.amount.add(amountStake);
           user.rewardDebt =
user.amount.mul(pool.accRewardPerShare).div(ACC_PRECISION);
           user.lastStakeTime = block.timestamp;
           try dividendToken.addStaker(address(msg.sender),
amountAfter) {} catch {}
           emit Deposit(msg.sender, _pid, amountStake);
      }

      function unstake(uint256 _pid) public verifyPoolId(_pid) {
           UserInfo storage user = userInfo[_pid][msg.sender];
           PoolInfo storage pool = poolInfo[_pid];

           uint256 _amount = user.amount;
           require(_amount > 0, "amount is zero");
           //require(user.lastStakeTime.add(pool.lockDuration) <
block.timestamp, "Under locked");


           updatePool(_pid);
           uint256 pendingAmount =
user.amount.mul(pool.accRewardPerShare).div(1e12).sub(user.rewardD
ebt);
           if (pendingAmount > 0) {
                user.rewardAmount =
user.rewardAmount.add(pendingAmount);
           }
```

```
            uint256 amountStake = _amount;
            if(user.lastStakeTime.add(pool.lockDuration) >=
block.timestamp){
                  uint256 feeUnStake =
amountStake.mul(withdrawFee).div(taxFee);
                  amountStake = amountStake.sub(feeUnStake);
                  pool.poolToken.safeTransfer(rewardAddress,
feeUnStake);
            }

            user.lastStakeTime = block.timestamp;
            pool.poolToken.safeTransfer(msg.sender, amountStake);
            user.amount = 0;
            pool.amount = pool.amount.sub(_amount);
            user.rewardDebt =
user.amount.mul(pool.accRewardPerShare).div(ACC_PRECISION);
            poolUsers[_pid].remove(msg.sender);
            try dividendToken.removeStaker(address(msg.sender)) {}
catch {}
            emit Withdraw(msg.sender, _pid, _amount);
      }
      function harvest(uint256 _pid) external verifyPoolId(_pid)
returns (uint256) {
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][msg.sender];

            updatePool(_pid);

            uint256 pendingAmount =
user.amount.mul(pool.accRewardPerShare).div(ACC_PRECISION).sub(use
r.rewardDebt);
            pendingAmount = user.rewardAmount.add(pendingAmount);
            if (pendingAmount > 0) {
                  rewardToken.transfer(msg.sender, pendingAmount);
            }
            user.rewardAmount = 0;
            user.rewardDebt =
user.amount.mul(pool.accRewardPerShare).div(ACC_PRECISION);
            emit Harvest(msg.sender, _pid, pendingAmount);
            return pendingAmount;
      }
```

```
    // getters
    function getPoolUserLength(uint256 _pid) external view
returns (uint256) {
        return poolUsers[_pid].length();
    }

    function getPoolUsers(uint256 _pid) external view returns
(address[] memory) {
        return poolUsers[_pid].values();
    }

    function getPoolUserStakes(uint256 _pid) external view
returns (address[] memory, uint256[] memory) {
        uint256 _len = poolUsers[_pid].length();
        uint256[] memory stakeAmount = new uint256[](_len);
        address[] memory users = poolUsers[_pid].values();
        for (uint256 index = 0; index < _len; index++) {
            stakeAmount[index] =
userInfo[_pid][poolUsers[_pid].at(index)].amount;
        }
        return (users, stakeAmount);
    }

    function getSubPoolUsers(
        uint256 _pid,
        uint256 _from,
        uint256 _length
    ) external view returns (address[] memory) {
        uint256 _poolUserLength = poolUsers[_pid].length();
        if (_from.add(_length) > _poolUserLength) {
            _length = _poolUserLength.sub(_from);
        }
        address[] memory results = new address[](_length);
        for (uint256 index = 0; index < _length; index++) {
            results[index] = poolUsers[_pid].at(_from +
index);
        }
        return results;
    }
    function getSubPoolUserStakes(
        uint256 _pid,
```

```
            uint256 _from,
            uint256 _length
      ) external view returns (address[] memory, uint256[] memory)
{
            uint256 _poolUserLength = poolUsers[_pid].length();
            if (_from.add(_length) > _poolUserLength) {
                  _length = _poolUserLength.sub(_from);
            }
            address[] memory results = new address[](_length);
            uint256[] memory stakeAmount = new uint256[](_length);
            for (uint256 index = 0; index < _length; index++) {
                  results[index] = poolUsers[_pid].at(_from +
index);
                  stakeAmount[index] =
userInfo[_pid][results[index]].amount;
            }
            return (results, stakeAmount);
      }
      function setTokenPerSecond(uint256 _tokenPerSecond, bool
_withUpdate) external onlyOwner {
            if (_withUpdate) {
                  massUpdatePools();
            }
            tokenPerSecond = _tokenPerSecond;
      }
      function setFee(uint256 _stakeFee, uint256 _withdrawFee)
external onlyOwner {
            require(_stakeFee <= 500, "_stakeFee failed");
            require(_withdrawFee <= 2000, "_withdrawFee failed");
            stakeFee = _stakeFee;
            withdrawFee = _withdrawFee;
      }
      function setRewardAddress(address _rewardAddress) external
onlyOwner {
            require(_rewardAddress <= address(0), "_rewardAddress
failed");
            rewardAddress = _rewardAddress;
      }
      function add(
            uint256 _allocPoint,
            IERC20 _poolToken,
            uint256 _lockDuration,
```

```
            bool _withUpdate
      ) external onlyOwner {
            if (_withUpdate) {
                  massUpdatePools();
            }
            uint256 lastRewardTime = block.timestamp > startTime ?
block.timestamp : startTime;
            totalAllocPoint = totalAllocPoint.add(_allocPoint);
            poolInfo.push(
                  PoolInfo({
                        poolToken: _poolToken,
                        allocPoint: _allocPoint,
                        lastRewardTime: lastRewardTime,
                        accRewardPerShare: 0,
                        amount: 0,
                        lockDuration: _lockDuration
                  })
            );
      }
      function set(
            uint256 _pid,
            uint256 _allocPoint,
            bool _withUpdate
      ) external onlyOwner verifyPoolId(_pid) {
            if (poolInfo[_pid].allocPoint != _allocPoint) {
                  if (_withUpdate) {
                        massUpdatePools();
                  }
                  totalAllocPoint =
totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
                  poolInfo[_pid].allocPoint = _allocPoint;
            }
      }
      function setLockDuration(uint256 _pid, uint256
_lockDuration) external onlyOwner verifyPoolId(_pid) {
            if (poolInfo[_pid].lockDuration != _lockDuration) {
                  poolInfo[_pid].lockDuration = _lockDuration;
            }
      }
}
```

READ CONTRACT (ONLY NEED TO KNOW)

1. stakeFee
200 uint256
(Function for read stakeFee)

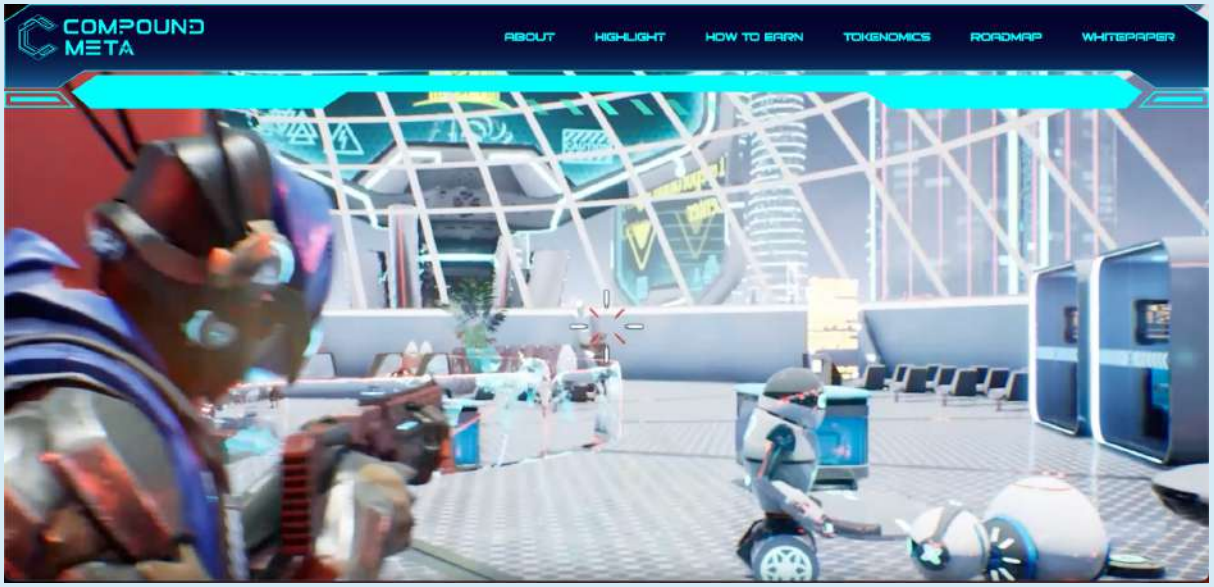
2. withdrawFee
1000 uint256
(Function for read withdrawFee)

WRITE CONTRACT

1. stake
(Function for staking)

2. unstake
newOwner (address)
(Function for unstake)

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 5% for stake and 20% for withdraw

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.