BlockSAFU

ADVANCE MANUAL SMART CONTRACT AUDIT

Project: MuscleX

Website: https://musclex.app/

PASSED

BlockSAFU Score:

85

Contract Address:

0x327c37704816df1f2fFe35f45bDF5661095b08B6

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

Tax

- Unstake Bronze 30% (owner can set fees over 100%).
- Unstake Silver 15% (owner can set fees over 100%).
- Unstake Gold 8% (owner can set fees over 100%).

SMART CONTRACT REVIEW

Token Name	**MuscleXCoinStaking**
Contract Address	0x327c37704816df1f2fFe35f45bDF5661095b08B6
Deployer Address	0xC57F463BB52c88d6b6A45eF0FCF833Ce630e0cd1
Owner Address	0xc57f463bb52c88d6b6a45ef0fcf833ce630e0cd1
Contract Created	Sep-19-2022 10:13:26 AM +UTC
Initial Liquidity	*will be updated after the DEX listing*
Liquidity Status	Locked
Unlocked Date	*will be updated after the DEX listing*
Verified CA	Yes
Compiler	v0.8.7+commit.e28d00a7
Optimization	Yes with 200 runs
Sol License	MIT License
Top 5 Holders	*will be updated after the DEX listing*
Other	default evmVersion

Team Review

The Muscle X team has a nice website, their website is professionally built and the Smart contract is well developed, their social media is growing with over 17,771 people in their telegram group (count in audit date).

Official Website And Social Media

Website: https://musclex.app/

Telegram Group: https://t.me/MuscleXOfficial

Twitter: https://twitter.com/MuscleXOfficial

MANUAL CODE REVIEW

● Minor-risk

0 minor-risk code issue found

Could be fixed, and will not bring problems.

● Medium-risk

1 medium-risk code issues found

Should be fixed, could bring problems.

1. Add limit on change tax rate (add limit can't set over 25%)

```
function ChangeTaxForGold(uint newTaxRate) public {
            require(ownersAddress == msg.sender, "User Not
Authorized");
            taxForGold = newTaxRate;
    }

    function ChangeTaxForSilver(uint newTaxRate) public {
            require(ownersAddress == msg.sender, "User Not
Authorized");
            taxForSilver = newTaxRate;
    }

    function ChangeTaxForBronze(uint newTaxRate) public {
            require(ownersAddress == msg.sender, "User Not
Authorized");
            taxForBronze = newTaxRate;
    }
```

● High-Risk

0 high-risk code issues found

Must be fixed, and will bring problems.

● Critical-Risk

0 critical-risk code issues found

Must be fixed, and will bring problems.

EXTRA NOTES SMART CONTRACT

1. MuscleXCoinStaking Contract

```
// SPDX-License-Identifier: MIT

import "./Ownable.sol";

pragma solidity ^0.8.4;

contract   MuscleXCoinStaking is Ownable {
    IBEP20 public rewardsToken;
    IBEP20 public stakingToken;

    uint256 public totalStaked;

    mapping(address => uint256) public stakingBalance;

    mapping(address => bool) public hasStaked;

    mapping(address => bool) public isStakingAtm;

    mapping(address=>uint) public numberOfDaysContract;

    uint stakingPeriod = 365;

    address[] public stakers;

    uint public rewardRateForXtier = 800;

    uint public rewardRateForGold  =  700;

    uint public rewardRateForSilver = 600;

    uint public rewardRateForBronze = 500;

    uint public taxForGold = 8;

    uint public taxForSilver = 15;

    uint public taxForBronze = 30;

    address payable ownersAddress;
    mapping(address=>uint) public lastTimeUserStaked;

    mapping(address=>uint) public accumulatedRewards;


    constructor(address _stakingToken, address _rewardsToken,
address administratorAddress) {
        stakingToken = IBEP20(_stakingToken);
        rewardsToken = IBEP20(_rewardsToken);
        ownersAddress = payable(administratorAddress);
    }


    function stake(uint _amount, uint numberOfDaysToStake) public
{
    totalStaked = totalStaked + _amount;

    bool isStakingPeriodValid = false;

    if(numberOfDaysToStake == 30 ){
        isStakingPeriodValid = true;
    }
    else if (numberOfDaysToStake == 21 ){
        isStakingPeriodValid = true;
    }

    else if (numberOfDaysToStake == 14 ){
        isStakingPeriodValid = true;
    }

    else if (numberOfDaysToStake == 7 ){
        isStakingPeriodValid = true;
    }

    require(isStakingPeriodValid == true, "Staking Time not
supported");

        stakingToken.transferFrom(msg.sender, address(this),
_amount *(10**18));
        if (hasStaked[msg.sender] == false) {
            stakers.push(msg.sender);
            hasStaked[msg.sender] = true;
        }

        if (isStakingAtm[msg.sender] == true ){
            require(numberOfDaysContract[msg.sender] ==
numberOfDaysToStake, "Sorry You need to be on the same APY as
before to stake more tokens " );
                    uint userRewards =
calculateUserRewards(msg.sender);
                    accumulatedRewards[msg.sender]  = userRewards;
        }else{
            accumulatedRewards[msg.sender] = 0;
        }

             stakingBalance[msg.sender] =
stakingBalance[msg.sender] + _amount;
             numberOfDaysContract[msg.sender] =
numberOfDaysToStake;
            lastTimeUserStaked[msg.sender] = block.timestamp;

            isStakingAtm[msg.sender] = true;
    }

    function calculateUserRewards (address userAddress) public
view returns(uint){
            if(isStakingAtm[userAddress] == true){
                uint numberOfDaysStaked =
calculateNumberOfDaysStaked(userAddress);
            uint userBalance = stakingBalance[userAddress] * (10
**18 );

            uint rewardRate;

            if(numberOfDaysContract[userAddress] == 30){
                rewardRate = rewardRateForXtier;
            }
            else if (numberOfDaysContract[userAddress] == 21 ){
                rewardRate = rewardRateForGold;
            }

            else if (numberOfDaysContract[userAddress] == 14  ){
                rewardRate = rewardRateForSilver;
            }
            else if (numberOfDaysContract[userAddress] == 7 ){
                rewardRate = rewardRateForBronze;
            }

            for(uint i = 0; i< numberOfDaysStaked; i++){
                userBalance  = userBalance + userBalance *
rewardRate / 100 / stakingPeriod;
            }

            return accumulatedRewards[userAddress] +  userBalance
- (stakingBalance[userAddress] * (10**18)) ;
            }else{
                return 0;
            }
    }

    function calculateExitFee (uint daysStaked, address
userAddress)public view returns(uint){

        uint exitFee = 0;

        uint numbersOfDaysStaked  = daysStaked;

        if(numberOfDaysContract[userAddress] == 21 &&
numbersOfDaysStaked < 21  ){
          exitFee = taxForGold;
        }
        else if (numberOfDaysContract[userAddress] == 14 &&
numbersOfDaysStaked < 14  ){
            exitFee = taxForSilver;
        }
        else if (numberOfDaysContract[msg.sender] == 7 &&
numbersOfDaysStaked < 7 ){
            exitFee = taxForBronze;
        }

        return exitFee;
    }

  function calculateNumberOfDaysStaked(address userAddress) public
view returns(uint){
      if(isStakingAtm[userAddress] == true ){
          uint lastTimeStaked = lastTimeUserStaked[userAddress];
          uint remainingTime = block.timestamp  - lastTimeStaked;
          uint remainingDays  = remainingTime / 86400;
          return remainingDays;
      }else{
          return 0;
      }
  }
    function claimReward(uint amount) external {
        uint reward = calculateUserRewards(msg.sender);

        uint numberOfDaysStaked =
calculateNumberOfDaysStaked(msg.sender);

        require(amount <= stakingBalance[msg.sender], "Can't
unstake more than your balance");

        bool canUserUnStake = false;

        if(numberOfDaysContract[msg.sender] == 30 ){
            if (numberOfDaysStaked >  29 ){
                canUserUnStake = true;
            }else {
                canUserUnStake = false;
            }
        }else{
            canUserUnStake = true;
        }

        require(canUserUnStake == true, "Can't unstake as an X
Tier Staker under 30 days ");

        require(numberOfDaysStaked > 0 , "Can't unstake in less
than a day");

        require(reward > 0, "Rewards is too small to be claimed");

        uint percentageOfRewardsToSend = amount * 100 /
stakingBalance[msg.sender] ;

        uint rewardsToPay = reward * percentageOfRewardsToSend /
```

```
100;

        uint totalToBePayed = amount  + (rewardsToPay / (10**18));

        uint percentageOfTaxToPay =
calculateExitFee(numberOfDaysStaked, msg.sender);

        require(rewardsToken.balanceOf(address(this)) / (10**18)
- totalToBePayed   >= totalStaked, "Contract Balance too Low");

        uint taxToPay = (rewardsToPay * percentageOfTaxToPay ) /
100;

        rewardsToken.transfer(msg.sender, rewardsToPay - taxToPay
);

        stakingToken.transfer(msg.sender, amount * (10**18));

        totalStaked = totalStaked  - amount;

        if (rewardsToPay >= accumulatedRewards[msg.sender]){
            accumulatedRewards[msg.sender] = 0;
        }else{
            accumulatedRewards[msg.sender] -=  rewardsToPay;
        }

        if(amount >= stakingBalance[msg.sender]){
        stakingBalance[msg.sender]  = 0;
        isStakingAtm[msg.sender] =  false;
        }else{
            stakingBalance[msg.sender] -= amount;
        }

    }

    function changeAdminAddress(address payable newAdminAddress)
public payable{
     require(msg.sender == ownersAddress, "UnAuthorized to take
this action");
        ownersAddress = newAdminAddress;
    }
    function ChangeRewardsForXTier(uint newRewardRate) public {
    require(ownersAddress == msg.sender, "User Not Authorized");
    rewardRateForXtier =  newRewardRate;

    }

    function ChangeRewardsForGold(uint newRewardRate) public {
    require(ownersAddress == msg.sender, "User Not Authorized");
    rewardRateForGold =  newRewardRate;

    }

    function ChangeRewardsForSilver(uint newRewardRate) public {
    require(ownersAddress == msg.sender, "User Not Authorized");
    rewardRateForSilver =  newRewardRate;

    }

    function ChangeRewardsForBronze(uint newRewardRate) public {
    require(ownersAddress == msg.sender, "User Not Authorized");
    rewardRateForBronze =  newRewardRate;

    }


    function ChangeTaxForGold(uint newTaxRate) public {
            require(ownersAddress == msg.sender, "User Not
Authorized");
            taxForGold = newTaxRate;
    }

    function ChangeTaxForSilver(uint newTaxRate) public {
            require(ownersAddress == msg.sender, "User Not
Authorized");
            taxForSilver = newTaxRate;
    }

    function ChangeTaxForBronze(uint newTaxRate) public {
            require(ownersAddress == msg.sender, "User Not
Authorized");
            taxForBronze = newTaxRate;
    }
```

```
function EmergencyUnstake() public {
    require(isStakingAtm[msg.sender] == true, "You currently don't
have any tokens staked");
    stakingToken.transfer(msg.sender, stakingBalance[msg.sender] *
(10**18));
    stakingBalance[msg.sender]  = 0;
    isStakingAtm[msg.sender] =  false;
    accumulatedRewards[msg.sender] = 0;
}



    function getTotalStaked() public view returns(uint){
                return totalStaked;
        }

    function getUserStakingBalance(address userAddress) public
view returns (uint){
            return stakingBalance[userAddress];
        }

    function getRewardRateXTier() public view returns (uint){
        return rewardRateForXtier;
        }

        function getRewardRateGold() public view returns (uint){
        return rewardRateForGold;
        }

        function getRewardRateSilver() public view returns (uint){
        return rewardRateForSilver;
        }

        function getRewardRateBronze() public view returns (uint){
        return rewardRateForBronze;
        }

    function changeStakingDays(uint newStakingDays) public {
        require(msg.sender == ownersAddress, "Not Authorized");
            stakingPeriod = newStakingDays;
    }
```

```
}

interface IBEP20 {
    function totalSupply() external view returns (uint);

    function balanceOf(address account) external view returns
(uint);

    function transfer(address recipient, uint amount) external
returns (bool);

    function allowance(address owner, address spender) external
view returns (uint);

    function approve(address spender, uint amount) external
returns (bool);

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

    event Transfer(address indexed from, address indexed to, uint
value);
    event Approval(address indexed owner, address indexed spender,
uint value);
}
```

MuscleXCoinStaking Contract

READ CONTRACT (ONLY NEED TO KNOW)

1. getRewardRateBronze
500 uint256
(Function for read reward rate for bronze tier)

2. getRewardRateGold
700 uint256
(Function for read reward rate for gold tier)

3. getRewardRateSilver
600 uint256
(Function for read reward rate for silver tier)

4. getRewardRateXTier
800 uint256
(Function for read reward rate for x tier)

5. rewardsToken
0x22e88b8abecc7e510c98d55991c626d67cdc52ea address
(Shows staking rewards)

6. stakingToken
0x22e88b8abecc7e510c98d55991c626d67cdc52ea address
(Shows staking token address)

WRITE CONTRACT

1. ChangeRewardsForBronze
newRewardRate (uint256)
(The form is filled with new rate for bronze tier)

2. ChangeRewardsForSilver
newRewardRate (uint256)
(The form is filled with new rate for silver tier)

3. ChangeRewardsForGold
newRewardRate (uint256)
(The form is filled with new rate for gold tier)

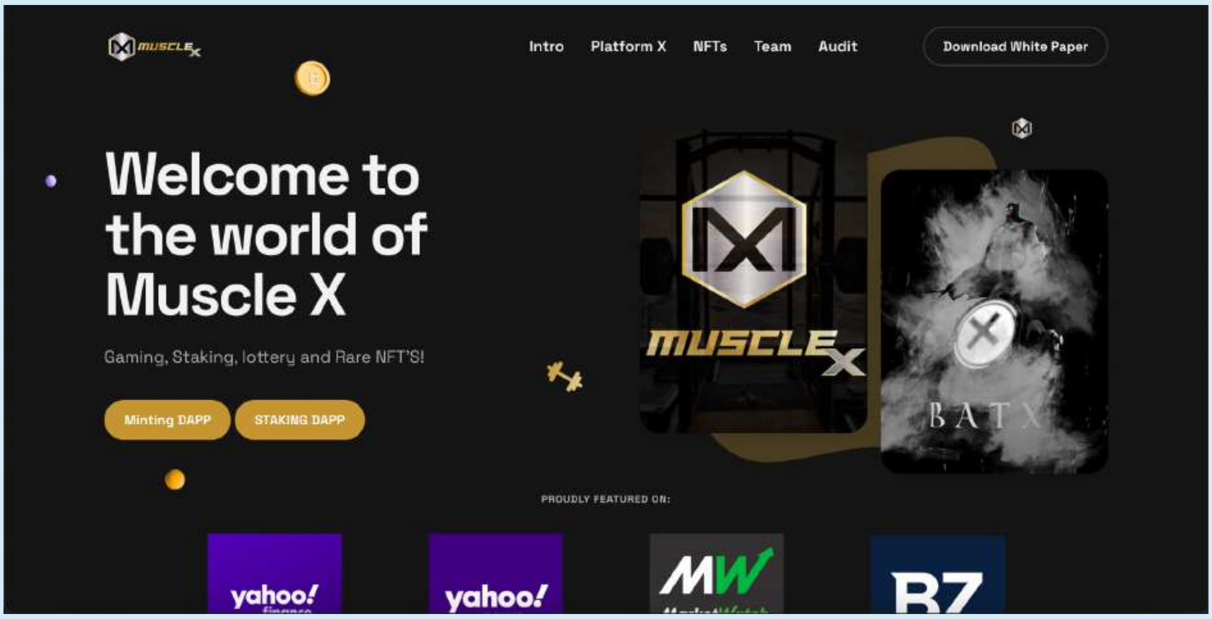
4. ChangeRewardsForXTier
newRewardRate (uint256)
(The form is filled with new rate for x tier)

5. changeAdminAddress
changeAdminAddress (payable amount)
newAdminAddress (address)
(The form is filled with amount and new admin address)

6. changeStakingDays
newStakingDays (uint256)
(The form is filled with new staking days)

6. transferOwnership
newOwner (address)
(Its function is to change the owner)

WEBSITE REVIEW



● Mobile Friendly

● Contains no code error

● SSL Secured (By GoDaddy SSL)

Web-Tech stack: Plesk,UIKit,Swiper Slider

Domain .app (GoDaddy) - Tracked by whois

First Contentful Paint:	857ms
Fully Loaded Time	2.7s
Performance	81%
Accessibility	85%
Best Practices	75%
SEO	91%

RUG-PULL REVIEW

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

- 🟢 The team is KYC By PinkSale

REVOKE REVIEW

- 🟢 Ability to unstake.
- 🟢 The owner is not able to pause the contract.
- 🟡 The owner can set tax up to 100%

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.