BlockSAFU

ADVANCE MANUAL
SMART CONTRACT AUDIT

Project: MuscleX

Website: https://musclex.app/

PASSED

BlockSAFU Score:

85

Contract Address:

0x7D742567bC8d7Bd6Ed3dD94daE31cC1A41ACc5E9

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 to 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

- **Tax rate (owner can set fees over 100%).**

SMART CONTRACT REVIEW

Token Name	**MuscleXNFTStaking**
Contract Address	0x7D742567bC8d7Bd6Ed3dD94daE31cC1A41ACc5E9
Deployer Address	0xC57F463BB52c88d6b6A45eF0FCF833Ce630e0cd1
Owner Address	0xC57F463BB52c88d6b6A45eF0FCF833Ce630e0cd1
Contract Created	Sep-19-2022 10:22:29 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 ChangeXStakingRate(uint newRate) public{
        require(msg.sender == _adminAddress, "Not Authorized");
        XStakingRate = newRate;
    }

    function ChangeXCoinStakingRate(uint newRate) public{
        require(msg.sender == _adminAddress, "Not Authorized");
        xStakingCoinRate = newRate ;
    }

    function ChangeBatXStakingRate(uint newRate) public{
        require(msg.sender == _adminAddress, "Not Authorized");
        batXStakingRate = newRate;
    }

    function ChangeBatXCoinStakingRate(uint newRate) public{
        require(msg.sender == _adminAddress, "Not Authorized");
        batXStakingCoinRate = newRate;
    }

    function ChangeCaptainXStakingRate(uint newRate) public{
      require(msg.sender == _adminAddress, "Not Authorized");
        captainXStakingRate = newRate;
    }
```

```
function ChangeCaptainXCoinStakingRate(uint newRate) public{
  require(msg.sender == _adminAddress, "Not Authorized");
    captainXStakingCoinRate = newRate;
}
```

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. MuscleXNFTStaking Contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7.0;

import "./Ownable.sol";


import "./IERC721.sol";
import "./IERC20.sol";

contract MuscleXNFTStakingBNB is Ownable  {

    IERC721 nftContract;

    IERC20 rewardToken;

    address private _adminAddress;

    event Received(address, uint);


    uint public XStakingRate = 0.15 ether;
    uint  public batXStakingRate = 0.07 ether;
    uint public  captainXStakingRate = 0.07 ether;
    uint public xStakingCoinRate = 60000 * ( 10 ** 18 );
    uint public batXStakingCoinRate  = 30000 * ( 10 ** 18 );
    uint public captainXStakingCoinRate = 30000 * ( 10 ** 18 );
    uint public  mininumStakingDays = 30;

    bool public isStakingActive;

    constructor(address nftAddress, address tokenAddress, address
adminAddress){
        nftContract  = IERC721(nftAddress);
        rewardToken = IERC20(tokenAddress);
        _adminAddress = adminAddress;
        isStakingActive = true;
    }
```

```
    receive() external payable {
        emit Received(msg.sender, msg.value);
    }


    mapping(address=>uint[]) public nftsStakedByUser;
    mapping(uint =>bool) public isNftStaked;
    mapping(uint=>uint) public timeNftWasStaked;
    mapping(uint=>uint) public nftStakingPeroid;
    mapping(uint=>address) public originalOwnerOfNft;
    mapping(uint=>bool) public isNFTRewardBNB;



    function stakeNft(uint nftTokenId, uint stakingPeriod, bool
rewardWithBNB )public{
        bool isUserOwnerOfNft =  nftContract.ownerOf(nftTokenId)
== msg.sender;
        bool canNFtBeStaked = checkIfNFTCanBeStaked(nftTokenId);
        require(isNftStaked[nftTokenId] == false, "Token Currently
Staked in Contract");
        require(isStakingActive == true, "Staking not currently
active, try again later");
        require(stakingPeriod > 0 , "Invalid Staking Period");
        require(isUserOwnerOfNft, "Not Owner of Nft");
        require(stakingPeriod >= mininumStakingDays, "Can't stake
less than the minuim staking days");
        require( canNFtBeStaked == true, "This NFT Can't be staked
" );
        nftContract.transferFrom(msg.sender, address(this),
nftTokenId);
        timeNftWasStaked[nftTokenId] = block.timestamp;
        isNftStaked[nftTokenId] = true;
        isNFTRewardBNB[nftTokenId] = rewardWithBNB;
        uint[] storage currentNftsStaked =
nftsStakedByUser[msg.sender];
        currentNftsStaked.push(nftTokenId);
        nftStakingPeroid[nftTokenId] = stakingPeriod;
        originalOwnerOfNft[nftTokenId] = msg.sender;
    }
```

```
    function checkUserNftStaked(address userAddress) public view
returns (uint[] memory) {
            return nftsStakedByUser[userAddress];
    }

    function getNFTGroupId(uint tokenId) public pure returns
(uint) {

         if(tokenId >= 0 && tokenId <  800){
               return 0;
            }
            else if (tokenId >= 800 && tokenId < 1600){
               return 1;
            }
            else if (tokenId >= 1600 && tokenId < 2000){
               return 2;
            }
            else if(tokenId >=2000 && tokenId < 2300){
               return 3;
            }
            else if (tokenId >= 2300 && tokenId < 2400){
               return 4;
            }else{
               return 5;
            }
    }

    function checkIfNFTCanBeStaked(uint tokenId) public pure
returns (bool){
        uint groupId = getNFTGroupId(tokenId);
        bool canNFtBeStaked = false;
        if(groupId == 5){
            canNFtBeStaked = true;
        }
        else if (groupId == 4){
           canNFtBeStaked = true;

        }
        else if (groupId == 3){
            canNFtBeStaked = true;
        }
```

```
        return canNFtBeStaked;

    }


    function calculateRewards(uint tokenId) public view  returns
(uint) {
        uint groupId = getNFTGroupId(tokenId);
        uint rewardRate;
        if(groupId == 5){
            if (isNFTRewardBNB[tokenId]){
             rewardRate = XStakingRate;
            }
            else{
                rewardRate = xStakingCoinRate;
            }
        }
        else if (groupId == 4){
            if(isNFTRewardBNB[tokenId]){
            rewardRate = batXStakingRate;
            }
            else{
                rewardRate = batXStakingCoinRate;
            }
        }
        else if(groupId  == 3){
            if(isNFTRewardBNB[tokenId]){
                 rewardRate = captainXStakingRate;
            }
            else{
                rewardRate = captainXStakingCoinRate;
            }
        }

        uint timeSpentStaking = block.timestamp -
timeNftWasStaked[tokenId];
        uint numberOfDaysStaked = timeSpentStaking / 86400;
        uint timeDurationForStaking = nftStakingPeroid[tokenId];
        uint reward = numberOfDaysStaked * rewardRate /
timeDurationForStaking ;
        return reward ;
    }
```

```
    function UnstakeAndClaimRewards(uint nftTokenId) public  {
        bool isUserOwnerOfNft =  originalOwnerOfNft[nftTokenId] ==
msg.sender;
        uint timeSpentStaking = block.timestamp -
timeNftWasStaked[nftTokenId];
        uint numberOfDaysStaked = timeSpentStaking / 86400;
        require(isUserOwnerOfNft, "Not Owner of Nft");
        require(isNftStaked[nftTokenId] == true, "Can't Unstake an
nft that is not staked ");
        require(numberOfDaysStaked > 0 , "NFT Can't be Unstaked
Less than 24 HRS of Staking");
        uint stakingreward = calculateRewards(nftTokenId);
        isNftStaked[nftTokenId] = false;
        timeNftWasStaked[nftTokenId] = 0;
        nftStakingPeroid[nftTokenId] = 0;

        uint[] storage previousNftsStaked =
nftsStakedByUser[msg.sender];

        uint[] memory newNftsStakedList = new
uint[](previousNftsStaked.length -1);
        uint newNFtListIndexer = 0;
        for (uint i=0; i < previousNftsStaked.length; i++ ){
            if(previousNftsStaked[i] != nftTokenId){
                newNftsStakedList[newNFtListIndexer] =
previousNftsStaked[i];
                newNFtListIndexer++;
            }
        }

        nftsStakedByUser[msg.sender] = newNftsStakedList;

        address payable addressToRecievePayment =
payable(originalOwnerOfNft[nftTokenId]);

        if(isNFTRewardBNB[nftTokenId]){
        addressToRecievePayment.transfer(stakingreward);
        }else{
            // Implement Erc20 transfer
            rewardToken.transfer(addressToRecievePayment,
stakingreward);
```

```
        }



        originalOwnerOfNft[nftTokenId] = address(0);

        nftContract.transferFrom(address(this), msg.sender,
nftTokenId);

    }

    function EmergencyUnstake(uint nftTokenId) public {
       bool isUserOwnerOfNft =  originalOwnerOfNft[nftTokenId] ==
msg.sender;
       require(isNftStaked[nftTokenId] == true, "Can't Unstake an
nft that is not staked ");
       require(isUserOwnerOfNft, "Not Owner of Nft");


        uint[] storage previousNftsStaked =
nftsStakedByUser[msg.sender];

        uint[] memory newNftsStakedList = new
uint[](previousNftsStaked.length -1);
        uint newNFtListIndexer = 0;
        for (uint i=0; i < previousNftsStaked.length; i++ ){
            if(previousNftsStaked[i] != nftTokenId){
                newNftsStakedList[newNFtListIndexer] =
previousNftsStaked[i];
                newNFtListIndexer++;
            }
        }

        nftsStakedByUser[msg.sender] = newNftsStakedList;

        originalOwnerOfNft[nftTokenId] = address(0);

        isNftStaked[nftTokenId] = false;

        timeNftWasStaked[nftTokenId] = 0;

        nftStakingPeroid[nftTokenId] = 0;
```

```
        nftContract.transferFrom(address(this), msg.sender,
nftTokenId);

    }


    function ChangeXStakingRate(uint newRate) public{
        require(msg.sender == _adminAddress, "Not Authorized");
        XStakingRate = newRate;
    }

    function ChangeXCoinStakingRate(uint newRate) public{
        require(msg.sender == _adminAddress, "Not Authorized");
        xStakingCoinRate = newRate ;
    }

    function ChangeBatXStakingRate(uint newRate) public{
        require(msg.sender == _adminAddress, "Not Authorized");
        batXStakingRate = newRate;
    }

    function ChangeBatXCoinStakingRate(uint newRate) public{
        require(msg.sender == _adminAddress, "Not Authorized");
        batXStakingCoinRate = newRate;
    }




    function ChangeCaptainXStakingRate(uint newRate) public{
      require(msg.sender == _adminAddress, "Not Authorized");
        captainXStakingRate = newRate;
    }

    function ChangeCaptainXCoinStakingRate(uint newRate) public{
      require(msg.sender == _adminAddress, "Not Authorized");
        captainXStakingCoinRate = newRate;
    }

    function ChangeMinimuimStakingDays(uint stakingDays) public{
        require(msg.sender == _adminAddress, "Not Authorized");
```

```
        mininumStakingDays = stakingDays;
    }

    function recieveBNB() public payable  returns (uint){
        return 2;
    }

    function changeAdminAddress(address newAdminAddress) public {
        require(msg.sender == _adminAddress, "Not Authorized");
        _adminAddress = newAdminAddress;
    }

    function ChangeStakingRewardToken(address newTokenAddress)
public {
        require(msg.sender == _adminAddress, "Not Authorized");
        rewardToken = IERC20(newTokenAddress);
    }

    function pauseStaking() public {
      require(msg.sender == _adminAddress, "Not Authorized");
      isStakingActive = false;
    }

    function resumeStaking() public {
      require(msg.sender == _adminAddress, "Not Authorized");
      isStakingActive = true;
    }
}
```

MuscleXCoinStaking Contract

READ CONTRACT (ONLY NEED TO KNOW)

1. xStakingRate
15000000000000000 uint256
(Function for read x staking rate)

2. batXStakingCoinRate
300000000000000000000 uint256
(Function for read batx Staking coin rate)

3. batXStakingRate
70000000000000000 uint256
(Function for read bat x staking rate)

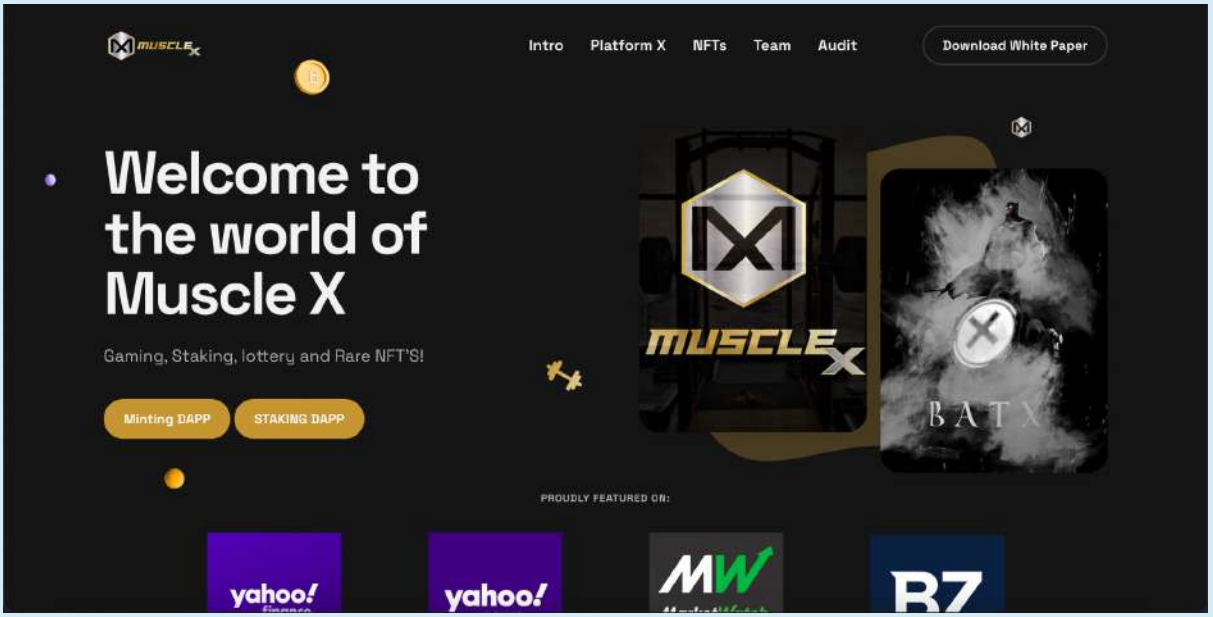
4. owner
0xc57f463bb52c88d6b6a45ef0fcf833ce630e0cd1 address
(Function for read contract owner)

WRITE CONTRACT

1. renounceOwnership
(Function for renounce)

2. 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 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.