BlockSAFU

ADVANCE MANUAL
SMART CONTRACT AUDIT

Project: Bitnou Master Chef

Website: https://bitnou.com/

PASSED

BlockSAFU Score:

92

Contract Address:

0xe5639b53654B0BF0316C32421ca6DF51A427d20A

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

- Buy 0% (No fees).
- Sell 0% (No fees).

Tx Amount

- Owner cannot set a max tx amount.

Transfer Pausable

- Owner cannot pause.

Blacklist

- Owner cannot blacklist.

Ownership

- Owner cannot take back ownership.

Proxy

- This contract has no proxy.

Anti Whale

- Owner cannot limit the number of wallet holdings.

Trading Cooldown

- Owner cannot set the selling time interval.

SMART CONTRACT REVIEW

Token Name	**BNOUSafe**
Contract Address	0xe5639b53654B0BF0316C32421ca6DF51A427d20A
Deployer Address	0x6b2a856A8954aa86eA66f9729597f4078D03e7a9
Owner Address	0x47a4ea43c6cf05e2541a76903f06d4b24fa4cc81
Gas Used for Buy	*will be updated after the DEX listing*
Gas Used for Sell	*will be updated after the DEX listing*
Contract Created	Sep-06-2022 03:37:40 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	v0.8.15+commit.e14f2714
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 Bnou team has a nice website, their website is professionally built and the Smart contract is well developed, their social media is growing with over 18 people in their telegram group (count in audit date).

Official Website And Social Media

Website: https://bitnou.com/

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

Discord: https://discord.com/invite/5Qb4bM7zYA

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. Burn manual

```
function burnBnou(bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }

        uint256 multiplier = block.number.sub(lastBurnedBlock);
        uint256 pendingBnouToBurn =
multiplier.mul(bnouPerBlockToBurn());

        // SafeTransfer BNOU
        _safeTransfer(burnAdmin, pendingBnouToBurn);
        lastBurnedBlock = block.number;
    }
```

Please burn to 0x..0000 or 0x..dead address

🔴 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. IBEP20

```
interface IBEP20 {
    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);
    event Transfer(address indexed from, address indexed to,
uint256 value);
    event Approval(address indexed owner, address indexed spender,
uint256 value);
}
```

IBEP20 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. Master Chef Contract

```
contract MasterChef is Ownable, ReentrancyGuard {
    using SafeMath for uint256;
    using SafeBEP20 for IBEP20;

    /// @notice Info of each MCV2 user.
    /// `amount` LP token amount the user has provided.
    /// `rewardDebt` Used to calculate the correct amount of
rewards. See explanation below.
    ///
    /// We do some fancy math here. Basically, any point in time,
the amount of BNOUs
    /// entitled to a user but is pending to be distributed is:
    ///
    ///   pending reward = (user share * pool.accBnouPerShare) -
user.rewardDebt
    ///
    ///   Whenever a user deposits or withdraws LP tokens to a
pool. Here's what happens:
    ///   1. The pool's `accBnouPerShare` (and `lastRewardBlock`)
gets updated.
    ///   2. User receives the pending reward sent to his/her
address.
    ///   3. User's `amount` gets updated. Pool's
`totalBoostedShare` gets updated.
    ///   4. User's `rewardDebt` gets updated.
    struct UserInfo {
        uint256 amount;
        uint256 rewardDebt;
        uint256 boostMultiplier;
    }

    /// @notice Info of each MCV2 pool.
    /// `allocPoint` The amount of allocation points assigned to
the pool.
    ///     Also known as the amount of "multipliers". Combined
with `totalXAllocPoint`, it defines the % of
    ///     BNOU rewards each pool gets.
    /// `accBnouPerShare` Accumulated BNOUs per share, times 1e12.
    /// `lastRewardBlock` Last block number that pool update
action is executed.
```

```
    /// `isRegular` The flag to set pool is regular or special.
See below:
    ///     In MasterChef V2 farms are "regular pools". "special
pools", which use a different sets of
    ///     `allocPoint` and their own `totalSpecialAllocPoint`
are designed to handle the distribution of
    ///     the BNOU rewards to all the BitnouSwap products.
    /// `totalBoostedShare` The total amount of user shares in
each pool. After considering the share boosts.
    struct PoolInfo {
        uint256 accBnouPerShare;
        uint256 lastRewardBlock;
        uint256 allocPoint;
        uint256 totalBoostedShare;
        bool isRegular;
    }

    /// @notice Address of BNOU contract.
    IBEP20 public immutable BNOU;
    /// @notice Address of BNOUSafe contract.
    IBNOUSafe public immutable bnouSafe;
    /// @notice Address of project developer.
    address public devAddr;
    /// @notice The only address can withdraw all the burn BNOU.
    address public burnAdmin;
    /// @notice The contract handles the share boosts.
    address public boostContract;

    /// @notice Info of each MCV2 pool.
    PoolInfo[] public poolInfo;
    /// @notice Address of the LP token for each MCV2 pool.
    IBEP20[] public lpToken;

    /// @notice Info of each pool user.
    mapping(uint256 => mapping(address => UserInfo)) public
userInfo;
    /// @notice The whitelist of addresses allowed to deposit in
special pools.
    mapping(address => bool) public whiteList;

    /// @notice Total regular allocation points. Must be the sum
of all regular pools' allocation points.
```

```
    uint256 public totalRegularAllocPoint;
    /// @notice Total special allocation points. Must be the sum
of all special pools' allocation points.
    uint256 public totalSpecialAllocPoint;
    ///  @notice 100 bnous per block in MCV1
    uint256 public constant MASTERCHEF_BNOU_PER_BLOCK = 100 *
1e18;
    uint256 public constant ACC_BNOU_PRECISION = 1e18;

    /// @notice Basic boost factor, none boosted user's boost
factor
    uint256 public constant BOOST_PRECISION = 100 * 1e10;
    /// @notice Hard limit for maxmium boost factor, it must
greater than BOOST_PRECISION
    uint256 public constant MAX_BOOST_PRECISION = 200 * 1e10;
    /// @notice total bnou rate = toBurn + toRegular + toSpecial
    uint256 public constant BNOU_RATE_TOTAL_PRECISION = 1e12;
    /// @notice BNOU distribute % for burn
    uint256 public bnouRateToBurn = 10_000_000_000;
    /// @notice BNOU distribute % for regular farm pool
    uint256 public bnouRateToRegularFarm = 300_000_000_000;
    /// @notice BNOU distribute % for special pools
    uint256 public bnouRateToSpecialFarm = 690_000_000_000;

    uint256 public lastBurnedBlock;

    uint256 public startBlock;

    event AddPool(uint256 indexed pid, uint256 allocPoint, IBEP20
indexed lpToken, bool isRegular);
    event SetPool(uint256 indexed pid, uint256 allocPoint);
    event UpdatePool(uint256 indexed pid, uint256 lastRewardBlock,
uint256 lpSupply, uint256 accBnouPerShare);
    event Deposit(address indexed user, uint256 indexed pid,
uint256 amount);
    event Withdraw(address indexed user, uint256 indexed pid,
uint256 amount);
    event EmergencyWithdraw(address indexed user, uint256 indexed
pid, uint256 amount);

    event UpdateBnouRate(uint256 burnRate, uint256
regularFarmRate, uint256 specialFarmRate);
```

```
    event UpdateBurnAdmin(address indexed oldAdmin, address
indexed newAdmin);
    event UpdateWhiteList(address indexed user, bool isValid);
    event DevAddressUpdate(address indexed oldDev, address indexed
newDev);
    event UpdateBoostContract(address indexed boostContract);
    event UpdateBoostMultiplier(address indexed user, uint256 pid,
uint256 oldMultiplier, uint256 newMultiplier);
    event UpdateStartBlock(address indexed caller, uint256
_oldstartBlock, uint256 _newstartBlock);

    constructor(
        IBEP20 _BNOU,
        address _bnouSafe,
        uint256 _startBlock,
        address _initializer
    ) {
        BNOU = _BNOU;
        bnouSafe = IBNOUSafe(_bnouSafe);
        startBlock = _startBlock;
        lastBurnedBlock = _startBlock;
        _transferOwnership(_initializer);
    }

    /**
     * @dev Throws if caller is not the boost contract.
     */
    modifier onlyBoostContract() {
        require(boostContract == msg.sender, "Ownable: caller is
not the boost contract");
        _;
    }

    /// @notice Returns the number of MCV2 pools.
    function poolLength() public view returns (uint256 pools) {
        pools = poolInfo.length;
    }

    /// @notice Add a new pool. Can only be called by the owner.
    /// DO NOT add the same LP token more than once. Rewards will
be messed up if you do.
    /// @param _allocPoint Number of allocation points for the new
```

```
pool.
    /// @param _lpToken Address of the LP BEP-20 token.
    /// @param _isRegular Whether the pool is regular or special.
LP farms are always "regular". "Special" pools are
    /// @param _withUpdate Whether call "massUpdatePools"
operation.
    /// only for BNOU distributions within BitnouSwap products.
    function add(
        uint256 _allocPoint,
        IBEP20 _lpToken,
        bool _isRegular,
        bool _withUpdate
    ) external onlyOwner {
        require(_lpToken.balanceOf(address(this)) >= 0, "None
BEP20 tokens");
        // stake BNOU token will cause staked token and reward
token mixed up,
        // may cause staked tokens withdraw as reward token,never
do it.
        require(_lpToken != BNOU, "BNOU token can't be added to
farm pools");

        if (_withUpdate) {
            massUpdatePools();
        }

        uint256 _lastRewardBlock = block.number > startBlock ?
block.number : startBlock;

        if (_isRegular) {
            totalRegularAllocPoint =
totalRegularAllocPoint.add(_allocPoint);
        } else {
            totalSpecialAllocPoint =
totalSpecialAllocPoint.add(_allocPoint);
        }
        lpToken.push(_lpToken);

        poolInfo.push(
            PoolInfo({
        allocPoint: _allocPoint,
        lastRewardBlock: _lastRewardBlock,
```

```
            accBnouPerShare: 0,
            isRegular: _isRegular,
            totalBoostedShare: 0
            })
            );
            emit AddPool(lpToken.length.sub(1), _allocPoint, _lpToken,
_isRegular);
    }

    /// @notice Update the given pool's BNOU allocation point. Can
only be called by the owner.
    /// @param _pid The id of the pool. See `poolInfo`.
    /// @param _allocPoint New number of allocation points for the
pool.
    /// @param _withUpdate Whether call "massUpdatePools"
operation.
    function set(
        uint256 _pid,
        uint256 _allocPoint,
        bool _withUpdate
    ) external onlyOwner {
        // No matter _withUpdate is true or false, we need to
execute updatePool once before set the pool parameters.
        updatePool(_pid);

        if (_withUpdate) {
            massUpdatePools();
        }

        if (poolInfo[_pid].isRegular) {
            totalRegularAllocPoint =
totalRegularAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPo
int);
        } else {
            totalSpecialAllocPoint =
totalSpecialAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPo
int);
        }
        poolInfo[_pid].allocPoint = _allocPoint;
        emit SetPool(_pid, _allocPoint);
    }
```

```
    /// @notice View function for checking pending BNOU rewards.
    /// @param _pid The id of the pool. See `poolInfo`.
    /// @param _user Address of the user.
    function pendingBnou(uint256 _pid, address _user) external
view returns (uint256) {
        PoolInfo memory pool = poolInfo[_pid];
        UserInfo memory user = userInfo[_pid][_user];
        uint256 accBnouPerShare = pool.accBnouPerShare;
        uint256 lpSupply = pool.totalBoostedShare;

        if (block.number > pool.lastRewardBlock && lpSupply != 0)
{
            uint256 multiplier =
block.number.sub(pool.lastRewardBlock);

            uint256 bnouReward =
multiplier.mul(bnouPerBlock(pool.isRegular)).mul(pool.allocPoint).
div(
                (pool.isRegular ? totalRegularAllocPoint :
totalSpecialAllocPoint)
            );
            accBnouPerShare =
accBnouPerShare.add(bnouReward.mul(ACC_BNOU_PRECISION).div(lpSuppl
y));
        }

        uint256 boostedAmount =
user.amount.mul(getBoostMultiplier(_user,
_pid)).div(BOOST_PRECISION);
        return
boostedAmount.mul(accBnouPerShare).div(ACC_BNOU_PRECISION).sub(use
r.rewardDebt);
    }

    /// @notice Update bnou reward for all the active pools. Be
careful of gas spending!
    function massUpdatePools() public {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            PoolInfo memory pool = poolInfo[pid];
            if (pool.allocPoint != 0) {
                updatePool(pid);
```

```
            }
        }
    }

    /// @notice Calculates and returns the `amount` of BNOU per
block.
    /// @param _isRegular If the pool belongs to regular or
special.
    function bnouPerBlock(bool _isRegular) public view returns
(uint256 amount) {
        if (_isRegular) {
            amount =
MASTERCHEF_BNOU_PER_BLOCK.mul(bnouRateToRegularFarm).div(BNOU_RATE
_TOTAL_PRECISION);
        } else {
            amount =
MASTERCHEF_BNOU_PER_BLOCK.mul(bnouRateToSpecialFarm).div(BNOU_RATE
_TOTAL_PRECISION);
        }
    }

    /// @notice Calculates and returns the `amount` of BNOU per
block to burn.
    function bnouPerBlockToBurn() public view returns (uint256
amount) {
        amount =
MASTERCHEF_BNOU_PER_BLOCK.mul(bnouRateToBurn).div(BNOU_RATE_TOTAL_
PRECISION);
    }

    /// @notice Update reward variables for the given pool.
    /// @param _pid The id of the pool. See `poolInfo`.
    /// @return pool Returns the pool that was updated.
    function updatePool(uint256 _pid) public returns (PoolInfo
memory pool) {
        pool = poolInfo[_pid];
        if (block.number > pool.lastRewardBlock) {
            uint256 lpSupply = pool.totalBoostedShare;
            uint256 totalAllocPoint = (pool.isRegular ?
totalRegularAllocPoint : totalSpecialAllocPoint);

            if (lpSupply > 0 && totalAllocPoint > 0) {
```

```
                uint256 multiplier =
block.number.sub(pool.lastRewardBlock);
                uint256 bnouReward =
multiplier.mul(bnouPerBlock(pool.isRegular)).mul(pool.allocPoint).
div(
                    totalAllocPoint
                );
                pool.accBnouPerShare =
pool.accBnouPerShare.add((bnouReward.mul(ACC_BNOU_PRECISION).div(l
pSupply)));
            }
            pool.lastRewardBlock = block.number;
            poolInfo[_pid] = pool;
            emit UpdatePool(_pid, pool.lastRewardBlock, lpSupply,
pool.accBnouPerShare);
        }
    }

    /// @notice Deposit LP tokens to pool.
    /// @param _pid The id of the pool. See `poolInfo`.
    /// @param _amount Amount of LP tokens to deposit.
    function deposit(uint256 _pid, uint256 _amount) external
nonReentrant {
        PoolInfo memory pool = updatePool(_pid);
        UserInfo storage user = userInfo[_pid][msg.sender];

        require(
            pool.isRegular || whiteList[msg.sender],
            "MasterChefV2: The address is not available to deposit
in this pool"
        );

        uint256 multiplier = getBoostMultiplier(msg.sender, _pid);

        if (user.amount > 0) {
            settlePendingBnou(msg.sender, _pid, multiplier);
        }

        if (_amount > 0) {
            uint256 before =
lpToken[_pid].balanceOf(address(this));
            lpToken[_pid].safeTransferFrom(msg.sender,
```

```
address(this), _amount);
            _amount =
lpToken[_pid].balanceOf(address(this)).sub(before);
            user.amount = user.amount.add(_amount);

            // Update total boosted share.
            pool.totalBoostedShare =
pool.totalBoostedShare.add(_amount.mul(multiplier).div(BOOST_PRECI
SION));
        }

        user.rewardDebt =
user.amount.mul(multiplier).div(BOOST_PRECISION).mul(pool.accBnouP
erShare).div(
            ACC_BNOU_PRECISION
        );
        poolInfo[_pid] = pool;

        emit Deposit(msg.sender, _pid, _amount);
    }

    /// @notice Withdraw LP tokens from pool.
    /// @param _pid The id of the pool. See `poolInfo`.
    /// @param _amount Amount of LP tokens to withdraw.
    function withdraw(uint256 _pid, uint256 _amount) external
nonReentrant {
        PoolInfo memory pool = updatePool(_pid);
        UserInfo storage user = userInfo[_pid][msg.sender];

        require(user.amount >= _amount, "withdraw: Insufficient");

        uint256 multiplier = getBoostMultiplier(msg.sender, _pid);

        settlePendingBnou(msg.sender, _pid, multiplier);

        if (_amount > 0) {
            user.amount = user.amount.sub(_amount);
            lpToken[_pid].safeTransfer(msg.sender, _amount);
        }

        user.rewardDebt =
user.amount.mul(multiplier).div(BOOST_PRECISION).mul(pool.accBnouP
```

```
erShare).div(
            ACC_BNOU_PRECISION
        );
        poolInfo[_pid].totalBoostedShare =
poolInfo[_pid].totalBoostedShare.sub(
            _amount.mul(multiplier).div(BOOST_PRECISION)
        );

        emit Withdraw(msg.sender, _pid, _amount);
    }

    /// @notice Withdraw without caring about the rewards.
EMERGENCY ONLY.
    /// @param _pid The id of the pool. See `poolInfo`.
    function emergencyWithdraw(uint256 _pid) external nonReentrant
{
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];

        uint256 amount = user.amount;
        user.amount = 0;
        user.rewardDebt = 0;
        uint256 boostedAmount =
amount.mul(getBoostMultiplier(msg.sender,
_pid)).div(BOOST_PRECISION);
        pool.totalBoostedShare = pool.totalBoostedShare >
boostedAmount ? pool.totalBoostedShare.sub(boostedAmount) : 0;

        // Note: transfer can fail or succeed if `amount` is zero.
        lpToken[_pid].safeTransfer(msg.sender, amount);
        emit EmergencyWithdraw(msg.sender, _pid, amount);
    }

    /// @notice Send BNOU pending for burn to `burnAdmin`.
    /// @param _withUpdate Whether call "massUpdatePools"
operation.
    function burnBnou(bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }

        uint256 multiplier = block.number.sub(lastBurnedBlock);
```

```
        uint256 pendingBnouToBurn =
multiplier.mul(bnouPerBlockToBurn());

        // SafeTransfer BNOU
        _safeTransfer(burnAdmin, pendingBnouToBurn);
        lastBurnedBlock = block.number;
    }

    /// @notice Update the % of BNOU distributions for burn,
regular pools and special pools.
    /// @param _burnRate The % of BNOU to burn each block.
    /// @param _regularFarmRate The % of BNOU to regular pools
each block.
    /// @param _specialFarmRate The % of BNOU to special pools
each block.
    /// @param _withUpdate Whether call "massUpdatePools"
operation.
    function updateBnouRate(
        uint256 _burnRate,
        uint256 _regularFarmRate,
        uint256 _specialFarmRate,
        bool _withUpdate
    ) external onlyOwner {
        require(
            _burnRate > 0 && _regularFarmRate > 0 &&
_specialFarmRate > 0,
            "MasterChefV2: Bnou rate must be greater than 0"
        );
        require(
            _burnRate.add(_regularFarmRate).add(_specialFarmRate)
== BNOU_RATE_TOTAL_PRECISION,
            "MasterChefV2: Total rate must be 1e12"
        );
        if (_withUpdate) {
            massUpdatePools();
        }
        // burn bnou base on old burn bnou rate
        burnBnou(false);

        bnouRateToBurn = _burnRate;
        bnouRateToRegularFarm = _regularFarmRate;
        bnouRateToSpecialFarm = _specialFarmRate;
```

```
        emit UpdateBnouRate(_burnRate, _regularFarmRate,
_specialFarmRate);
    }

    /// @notice Update burn admin address.
    /// @param _newAdmin The new burn admin address.
    function updateBurnAdmin(address _newAdmin) external onlyOwner
{
        require(_newAdmin != address(0), "MasterChefV2: Burn admin
address must be valid");
        require(_newAdmin != burnAdmin, "MasterChefV2: Burn admin
address is the same with current address");
        address _oldAdmin = burnAdmin;
        burnAdmin = _newAdmin;
        emit UpdateBurnAdmin(_oldAdmin, _newAdmin);
    }

    /// @notice Update whitelisted addresses for special pools.
    /// @param _user The address to be updated.
    /// @param _isValid The flag for valid or invalid.
    function updateWhiteList(address _user, bool _isValid)
external onlyOwner {
        require(_user != address(0), "MasterChefV2: The white list
address must be valid");

        whiteList[_user] = _isValid;
        emit UpdateWhiteList(_user, _isValid);
    }

    /// @notice Update boost contract address and max boost
factor.
    /// @param _newBoostContract The new address for handling all
the share boosts.
    function updateBoostContract(address _newBoostContract)
external onlyOwner {
        require(
            _newBoostContract != address(0) && _newBoostContract
!= boostContract,
            "MasterChefV2: New boost contract address must be
valid"
        );
```

```
        boostContract = _newBoostContract;
        emit UpdateBoostContract(_newBoostContract);
    }

    // Update developer address by the previous developer address.
    function updateDevAddress(address _devAddr) external
onlyOwner{
        address previousDevAddr = devAddr;
        devAddr = _devAddr;
        emit DevAddressUpdate(previousDevAddr, _devAddr);
    }

    /// @notice Update user boost factor.
    /// @param _user The user address for boost factor updates.
    /// @param _pid The pool id for the boost factor updates.
    /// @param _newMultiplier New boost multiplier.
    function updateBoostMultiplier(
        address _user,
        uint256 _pid,
        uint256 _newMultiplier
    ) external onlyBoostContract nonReentrant {
        require(_user != address(0), "MasterChefV2: The user
address must be valid");
        require(poolInfo[_pid].isRegular, "MasterChefV2: Only
regular farm could be boosted");
        require(
            _newMultiplier >= BOOST_PRECISION && _newMultiplier <=
MAX_BOOST_PRECISION,
            "MasterChefV2: Invalid new boost multiplier"
        );

        PoolInfo memory pool = updatePool(_pid);
        UserInfo storage user = userInfo[_pid][_user];

        uint256 prevMultiplier = getBoostMultiplier(_user, _pid);
        settlePendingBnou(_user, _pid, prevMultiplier);

        user.rewardDebt =
user.amount.mul(_newMultiplier).div(BOOST_PRECISION).mul(pool.accB
nouPerShare).div(
            ACC_BNOU_PRECISION
```

```
        );
        pool.totalBoostedShare =
pool.totalBoostedShare.sub(user.amount.mul(prevMultiplier).div(BOO
ST_PRECISION)).add(
            user.amount.mul(_newMultiplier).div(BOOST_PRECISION)
        );
        poolInfo[_pid] = pool;
        userInfo[_pid][_user].boostMultiplier = _newMultiplier;

        emit UpdateBoostMultiplier(_user, _pid, prevMultiplier,
_newMultiplier);
    }

    /// @notice Get user boost multiplier for specific pool id.
    /// @param _user The user address.
    /// @param _pid The pool id.
    function getBoostMultiplier(address _user, uint256 _pid)
public view returns (uint256) {
        uint256 multiplier =
userInfo[_pid][_user].boostMultiplier;
        return multiplier > BOOST_PRECISION ? multiplier :
BOOST_PRECISION;
    }

    /// @notice Settles, distribute the pending BNOU rewards for
given user.
    /// @param _user The user address for settling rewards.
    /// @param _pid The pool id.
    /// @param _boostMultiplier The user boost multiplier in
specific pool id.
    function settlePendingBnou(
        address _user,
        uint256 _pid,
        uint256 _boostMultiplier
    ) internal {
        UserInfo memory user = userInfo[_pid][_user];

        uint256 boostedAmount =
user.amount.mul(_boostMultiplier).div(BOOST_PRECISION);
        uint256 accBnou =
boostedAmount.mul(poolInfo[_pid].accBnouPerShare).div(ACC_BNOU_PRE
CISION);
```

```
        uint256 pending = accBnou.sub(user.rewardDebt);
        // SafeTransfer BNOU
        _safeTransfer(_user, pending);
    }

    /// @notice Safe Transfer BNOU.
    /// @param _to The BNOU receiver address.
    /// @param _amount transfer BNOU amounts.
    function _safeTransfer(address _to, uint256 _amount) internal
{
        bnouSafe.safeBNOUTransfer(devAddr, _amount.div(10));
        bnouSafe.safeBNOUTransfer(_to, _amount);
    }

    // updateStartBlock, can only update before starting
    function updateStartBlock(uint256 _startBlock) public
onlyOwner {
           require(startBlock > block.number, "updateStartBlock:
cannot update when farming has started");
        startBlock = _startBlock;
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            PoolInfo memory pool = poolInfo[pid];
            pool.lastRewardBlock = _startBlock;
            poolInfo[pid] = pool;
        }

        lastBurnedBlock = _startBlock;

           emit UpdateStartBlock(msg.sender, startBlock,
_startBlock);
    }
}
```

4. Bnou Master Chef

```
function burnBnou(bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }

        uint256 multiplier = block.number.sub(lastBurnedBlock);
        uint256 pendingBnouToBurn =
multiplier.mul(bnouPerBlockToBurn());

        // SafeTransfer BNOU
        _safeTransfer(burnAdmin, pendingBnouToBurn);
        lastBurnedBlock = block.number;
    }
```

Burn not to 0x….000 or 0x….dead, but amount send to Burn admin (address) -Warning

READ CONTRACT (ONLY NEED TO KNOW)

1. ACC_BNOU_PRECISION
10000000000000000 uint256
(Function for read bnou precision)

2. BNOU
0x221e4c3bbabc3e33a8be082c1f96037a9761a9f2 address
(Function for read bnou address)

3. bnouSafe
0x56f600852a0f3758f0bc38e047f988923e86771d address
(Function for read bnou safe address)

4. owner
0x47a4ea43c6cf05e2541a76903f06d4b24fa4cc81 address
(Function for read owner address)

5. totalRegularAllocPoint
8000 uint256
(Function for read total regular allocation point)

6. totalSpecialAllocPoint
57600 uint256
(Function for read total special allocation point)

WRITE CONTRACT

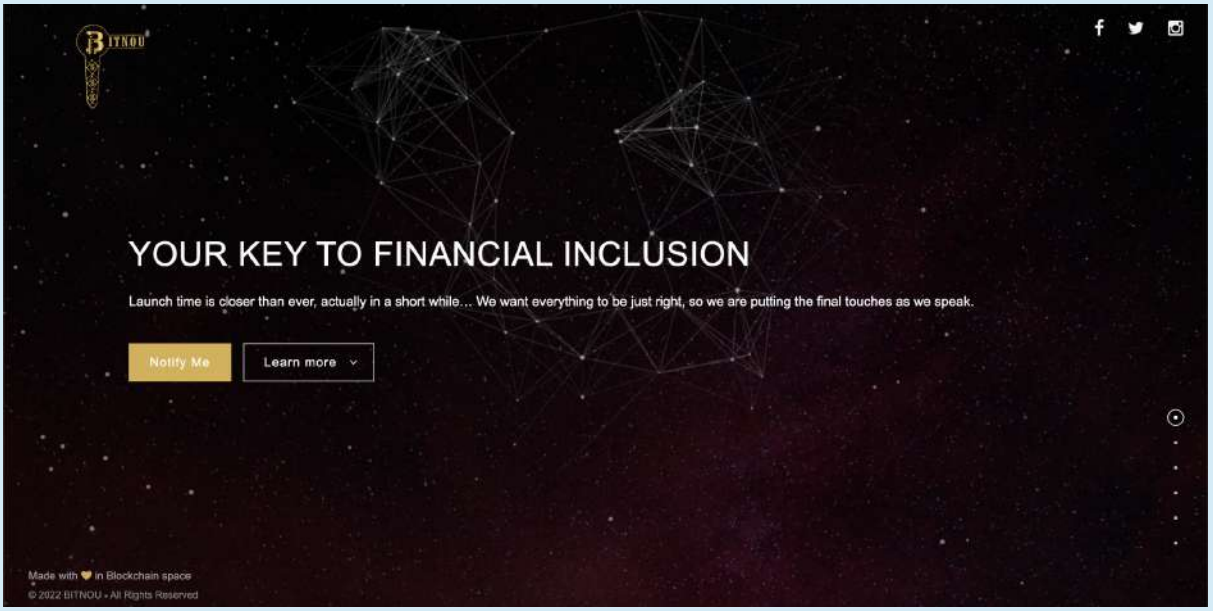
1. renounceOwnership
(Renouncing ownership will leave the contract without an owner, thereby removing any functionality that is only available to the owner)

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

3. updateBurnAdmin
_newAdmin (address)
(Its function for set burn admin)

4. updateDevAddress
_devAddr (address)
(its function for set dev address)

WEBSITE REVIEW



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

Web-Tech stack: Apache, Bootstrap, Animate css

Domain .com (Hostgator) - Tracked by whois

First Contentful Paint:	1.6s
Fully Loaded Time	5.6s
Performance	46%
Accessibility	79%
Best Practices	58%
SEO	80%

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 DEX listing*)

- TOP 5 Holder.

  (*Will be updated after DEX listing*)

- The Team KYC by Blocksafu

HONEYPOT REVIEW

- Ability to sell.
- The owner is not able to pause the contract.
- The owner can’t set fees

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.