BlockSAFU

ADVANCE MANUAL
SMART CONTRACT AUDIT

Project: DEEP SKYLINE

Website: https://www.wstecserver.com/

BlockSAFU Score:

80

Contract Address:

0x7f97da02976c10A9c5c10a2E19491C050A184035

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

BlockSAFU was commissioned by DES to complete a Smart Contract audit. The objective of the Audit is to achieve the following:

- Review the Project and experience and Development team
- Ensure that the Smart Contract functions are necessary and operate as intended.
- Identify any vulnerabilities in the Smart Contract code.

DISCLAIMER: This Audit is intended to inform about token Contract Risks, the result does not imply an endorsement or provide financial advice in any way, all investments are made at your own risk. (https://blocksafu.com/)

SMART CONTRACT REVIEW

Contract Name	**ArbiPenguinRewards**
Contract Address	0x945a2cF9DD0ab96f75C5b436186B31f6F2750F35
Deployer Address	0x4cB418C5279954db46cfa8d6Af17E0Cc7a8Fd486
Owner Address	0x4cB418C5279954db46cfa8d6Af17E0Cc7a8Fd486
Contract Created	Aug-15-2022 06:01:16 PM +UTC
Verified CA	Yes
Compiler	v0.8.7+commit.e28d00a7
Optimization	Yes with 200 runs
Sol License	MIT License
Other	default evmVersion

OVERVIEW

-

Team Review

The DES team has a nice website, their website is professionally built and the Smart contract is well developed, their social media is growing with over 16,864 people in their telegram group (count in audit date).

Official Website And Social Media

Website: https://www.wstecserver.com/

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

Twitter: https://twitter.com/des_token

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
```

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 {
    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( addr
        ess 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
    );
}
```

IERC20 Normal Base Template

2. Ownable Contract

```
contract Ownable {
    address public owner;

    event OwnershipTransferred(
        address indexed previousOwner,
        address indexed newOwner
    );

    constructor() {
        owner = msg.sender;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(msg.sender == owner, "Ownable: caller is not the
owner");
        _;
    }

    /**
     * @dev Leaves the contract without owner. It will not be
possible to call
     * `onlyOwner` functions anymore. Can only be called by the
current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without
an owner,
     * thereby removing any functionality that is only available to
the owner.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(owner, address(0));
        owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account
(`newOwner`).
     * Can only be called by the current owner.
```

```
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account
(`newOwner`).
     */
    function _transferOwnership(address newOwner) internal
        { require(
            newOwner != address(0),
            "Ownable: new owner is the zero address"
        );
        emit OwnershipTransferred(owner, newOwner);
        owner = newOwner;
    }
}
```

Standard Ownable contract

3. Staked Token Wrapper Contract

```
contract StakedTokenWrapper {
    uint256 public totalSupply;

    mapping(address => uint256) private _balances;
    IERC20 public stakedToken;

    event Staked(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);

    function balanceOf(address account) public view returns
(uint256) {
        return _balances[account];
    }

    string constant _transferErrorMessage = "staked token transfer
failed";

    function stakeFor(address forWhom, uint128 amount) public
payable virtual {
        IERC20 st = stakedToken;
        if (st == IERC20(address(0))) {
            //eth
            unchecked {
                totalSupply += msg.value;
                _balances[forWhom] += msg.value;
            }
        } else {
            require(msg.value == 0, "non-zero eth");
            require(amount > 0, "Cannot stake 0");
            require(
                st.transferFrom(msg.sender, address(this),
amount),
                _transferErrorMessage
            );
            unchecked {
                totalSupply += amount;
                _balances[forWhom] += amount;
            }
        }
        emit Staked(forWhom, amount);
    }
```

```
    function withdraw(uint128 amount) public virtual {
        require(amount <= _balances[msg.sender], "withdraw:
balance is lower");

        unchecked {
            _balances[msg.sender] -= amount;
            totalSupply = totalSupply - amount;
        }
        IERC20 st = stakedToken;
        if (st == IERC20(address(0))) {
            //eth
            (bool success, ) = msg.sender.call{value: amount}("");
            require(success, "eth transfer failure");
        } else {
            require(
                stakedToken.transfer(msg.sender, amount),
                _transferErrorMessage
            );
        }
        emit Withdrawn(msg.sender, amount);
    }
}
```

Staked Token Wrapper Contract

4. Contract ArbiPenguinRewards

```
contract ArbiPenguinRewards is StakedTokenWrapper, Ownable
    { IERC20 public rewardToken;
    uint256 public rewardRate;
    uint64 public periodFinish;
    uint64 public lastUpdateTime;
    uint128 public rewardPerTokenStored;
    uint256[] public shareRate = [2000, 600, 180, 54];
    uint256 public allShareRate = 2000 + 600 + 180 + 54;
    IReferral public referral;
    struct UserRewards {
        uint128 userRewardPerTokenPaid;
        uint128 rewards;
    }
    mapping(address => UserRewards) public userRewards;

    event RewardAdded(uint256 reward);
    event RewardPaid(address indexed user, uint256 reward);

    constructor(
        IERC20 _rewardToken,
        IERC20 _stakedToken,
        address _referral
    ) {
        rewardToken = _rewardToken;
        stakedToken = _stakedToken;
        referral = IReferral(_referral);
    }

    modifier updateReward(address account) {
        uint128 _rewardPerTokenStored = rewardPerToken();
        lastUpdateTime = lastTimeRewardApplicable();
        rewardPerTokenStored = _rewardPerTokenStored;
        userRewards[account].rewards = earned(account);
        userRewards[account].userRewardPerTokenPaid =
_rewardPerTokenStored;
        _;
    }

    function lastTimeRewardApplicable() public view returns
(uint64) {
        uint64 blockTimestamp = uint64(block.timestamp);
```

```
        return blockTimestamp < periodFinish ? blockTimestamp :
periodFinish;
    }

    function rewardPerToken() public view returns (uint128)
        { uint256 totalStakedSupply = totalSupply;
        if (totalStakedSupply == 0) {
            return rewardPerTokenStored;
        }
        unchecked {
            uint256 rewardDuration = lastTimeRewardApplicable() -
                lastUpdateTime;
            return
                uint128(
                    rewardPerTokenStored +
                        (rewardDuration * rewardRate * 1e18) /
                        totalStakedSupply
                );
        }
    }

    function earned(address account) public view returns (uint128)
{
        unchecked {
            return
                uint128(
                    (balanceOf(account) *
                        (rewardPerToken() -

userRewards[account].userRewardPerTokenPaid)) /
                        1e18 +
                        userRewards[account].rewards
                );
        }
    }

    function getLeftEarned(address account) public view returns
(uint256) {
        uint256 allRewards =
            ( uint128(
                (balanceOf(account) *
```

```
                    (rewardPerToken() -

userRewards[account].userRewardPerTokenPaid)) /
                    1e18 +
                    userRewards[account].rewards
            )
        );

        uint256 fee = (allRewards * allShareRate) / 10000;
        unchecked {
            return
                uint128(
                    (balanceOf(account) *
                        (rewardPerToken() -

userRewards[account].userRewardPerTokenPaid)) /
                        1e18 +
                        userRewards[account].rewards -
                        fee
                );
        }
    }

    function stake(uint128 amount) external payable
        { stakeFor(msg.sender, amount);
    }

    function stakeFor(address forWhom, uint128 amount)
        public
        payable
        override
        updateReward(forWhom)
    {
        super.stakeFor(forWhom, amount);
    }

    function withdraw(uint128 amount) public override
updateReward(msg.sender) {
        super.withdraw(amount);
    }

    function exit() external {
```

```
        getReward();
        withdraw(uint128(balanceOf(msg.sender)));
    }

    function  getReward()  public  updateReward(msg.sender)
        { uint256 reward = earned(msg.sender);
        address addr1 = referral.referrers(msg.sender);
        address addr2 = referral.referrers(addr1);
        address addr3 = referral.referrers(addr2);
        address addr4 = referral.referrers(addr3);
        uint256 share1 = (reward * shareRate[0]) / 10000;
        uint256 share2 = (reward * shareRate[1]) / 10000;
        uint256 share3 = (reward * shareRate[2]) / 10000;
        uint256 share4 = (reward * shareRate[3]) / 10000;
        if (addr1 != address(0)) {
            rewardToken.transfer(addr1, share1);
        } else {
            rewardToken.transfer(address(this), share1);
        }

        if (addr2 != address(0)) {
            rewardToken.transfer(addr2, share2);
        } else {
            rewardToken.transfer(address(this), share2);
        }

        if (addr3 != address(0)) {
            rewardToken.transfer(addr3, share3);
        } else {
            rewardToken.transfer(address(this), share3);
        }

        if (addr4 != address(0)) {
            rewardToken.transfer(addr4, share4);
        } else {
            rewardToken.transfer(address(this), share4);
        }

        uint256 leftReward = getLeftEarned(msg.sender);

        if (leftReward > 0) {
            userRewards[msg.sender].rewards = 0;
```

```
            require(
                rewardToken.transfer(msg.sender, leftReward),
                "reward transfer failed"
            );
            emit RewardPaid(msg.sender, leftReward);
        }
    }

    function setRewardParams(uint128 reward, uint64 duration)
        external
        onlyOwner
    {
        unchecked {
            require(reward > 0);

            rewardPerTokenStored = rewardPerToken();
            uint64 blockTimestamp = uint64(block.timestamp);
            uint256 maxRewardSupply =
rewardToken.balanceOf(address(this));
            if (rewardToken == stakedToken) maxRewardSupply -=
totalSupply;
            uint256 leftover = 0;
            if (blockTimestamp >= periodFinish)
                { rewardRate = reward / duration;
            } else {
                uint256 remaining = periodFinish - blockTimestamp;
                leftover = remaining * rewardRate;
                rewardRate = (reward + leftover) / duration;
            }

            require(reward + leftover <= maxRewardSupply, "not
enough tokens");
            lastUpdateTime = blockTimestamp;
            periodFinish = blockTimestamp + duration;

            emit RewardAdded(reward);
        }
    }

    function withdrawReward() external onlyOwner
        { uint256 rewardSupply =
rewardToken.balanceOf(address(this));
```

```
        //ensure funds staked by users can't be transferred out
        if (rewardToken == stakedToken) rewardSupply -=
totalSupply;
        require(rewardToken.transfer(msg.sender, rewardSupply));
        rewardRate = 0;
        periodFinish = uint64(block.timestamp);
    }
}
```

```
function stakeFor(address forWhom, uint128 amount) public payable
virtual {
        IERC20 st = stakedToken;
        if (st == IERC20(address(0))) {
            //eth
            unchecked {
                totalSupply += msg.value;
                _balances[forWhom] += msg.value;
            }
        } else {
            require(msg.value == 0, "non-zero eth");
            require(amount > 0, "Cannot stake 0");
            require(
                st.transferFrom(msg.sender, address(this),
amount),
                _transferErrorMessage
            );
            unchecked {
                totalSupply += amount;
                _balances[forWhom] += amount;
            }
        }
        emit Staked(forWhom, amount);
    }
```

This function for staking

READ CONTRACT (ONLY NEED TO KNOW)

1. balanceOf
account (address)
(The form is filled with the address you want to check balance)

2. earned
account (address)
(The form is filled with the address you want to check earned reward)

3. owner
0xac45ecf9f362eee0ee7f6d1c39bcc51f796abb09 (address)

4. referral
0x3453d7833faea0693f625c88e0a296d78ceec71 (address)

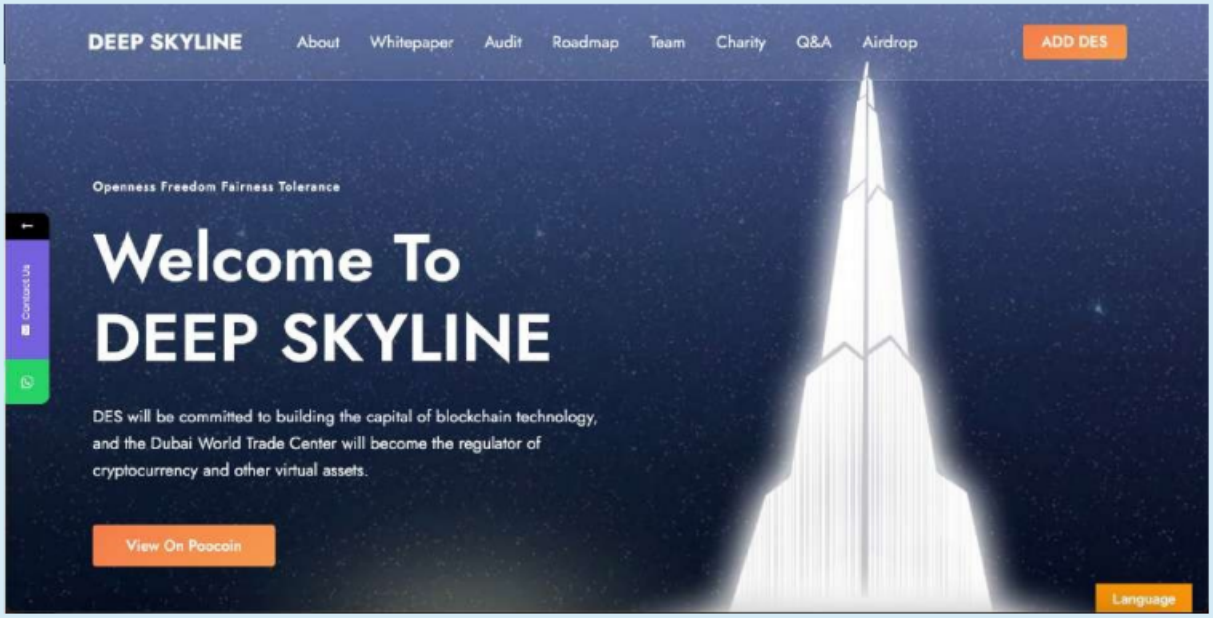
5. rewardToken
0x8753ceb6196467fc64e170f63bc0fe9331f9d3a8 (Des Token) (address)

6. stakedToken
0xb0b125169d4866c821a2d30bf2a204c45fc634fc (Pancake LP)

WRITE CONTRACT

1. getReward
call function
(Call function to trigger get reward from stake)
2. renounceOwnership
(Renouncing ownership will leave the contract without an owner, thereby removing any functionality that is only available to the owner)
3. setRewardParams
reward (uint128)
duration (uint64)
(this function to set reward and duration)
4. stake
stake (payable BNB)
amount (uint128)
(this function for trigger staking function with input value stake and amount for stake)
5. transferOwnership
(this function to transfer ownership from old owner to new owner)
6. Withdraw
amount (uint128)
(this function to withdraw stake token reward)

WEBSITE REVIEW



🟢 **Mobile Friendly**

🟢 **Contains no code error**

🔴 **SSL Not Secured (if dapps not secured, can't open with dapps wallet)**

Web-Tech stack: jQuery, Wordpress

Domain .com (WildWest) - Tracked by whois

First Contentful Paint:	1.2s
Fully Loaded Time	4.4s
Performance	88%
Accessibility	94%
Best Practices	75%
SEO	75%

RUG-PULL REVIEW

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

- The Team No KYC

HONEYPOT REVIEW

- Ability to sell.

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.