BlockSAFU

ADVANCE MANUAL
SMART CONTRACT AUDIT

Project: Dark Web3

Website: https://darkweb3.io/

BlockSAFU Score:

97

Contract Address:

0xC6775a74a0113E1Efe7b1CA21f9a2Ef8C86eF3C4

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 Dark Web3 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

Token Name	**Dark Web3**
Token Symbol	**DARK**
Token Decimal	18
Total Supply	100,000,000 **DARK**
Contract Address	0xC6775a74a0113E1Efe7b1CA21f9a2Ef8C86eF3C4
Deployer Address	0x12a7cA11520f86fb46E0A1a790AEb731333BB5C5
Owner Address	0x12a7cA11520f86fb46E0A1a790AEb731333BB5C5
Tax Fees Buy	0%
Tax Fees Sell	5%
Transfer Fee	5%
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-03-2022 01:03:31 PM +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	v0.8.16+commit.07a7930e
Optimization	Yes with 5000 runs
Sol License	MIT License
Top 5 Holders	*will be updated after the DEX listing*
Other	default evmVersion

TAX

BUY	0%	**SELL**	5%	**TRANSFER**	5%

OVERVIEW

Mint Function

- No mint functions.

Fees

- Buy 0% (owner can't set fees over 8%, and 10% for total buy+sell).
- Sell 5% (owner can't set fees over 8%,and 10% for total buy+sell).
- Transfer 5% (owner can't set fees over 8%)

Tx Amount

- Owner cannot set a max tx amount.

Transfer Pausable

- Owner cannot pause.

Blacklist

- Owner cannot set a 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.

Token Holder

Rank	Address	Quantity	Percentage	Analytics
1	0x9d252calfbddfb169e2f790fdd159b5dc024710b	54,950,000	54.9500%	
2	Pinksale: PinkLock V2	44,000,000	44.0000%	
3	0x12a7ca11520f86fb46e0a1a790aeb731333bb5c5	1,050,000	1.0500%	

[Download CSV Export]

Team Review

The Dark Web3 team has a nice website, their website is professionally built and the Smart contract is well developed, their social media is growing with over 13 people in their telegram group (count in audit date).

Official Website And Social Media

Website: https://darkweb3.io/

Telegram Group: https://t.me/darkweb3_official

Twitter: https://twitter.com/darkweb3io

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
    ) external returns (bool);
```

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 amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns
(uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to
`recipient`.
     *
     * Returns a boolean value indicating whether the operation
succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external
returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender`
will be
     * allowed to spend on behalf of `owner` through
{transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are
called.
     */
    function allowance(address owner, address spender) external
view returns (uint256);
```

```
    /**
     * @dev Sets `amount` as the allowance of `spender` over the
caller's tokens.
     *
     * Returns a boolean value indicating whether the operation
succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this
method brings the risk
     * that someone may use both the old and the new allowance by
unfortunate
     * transaction ordering. One possible solution to mitigate this
race
     * condition is to first reduce the spender's allowance to 0
and set the
     * desired value afterwards:
     *
https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external
returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient`
using the
     * allowance mechanism. `amount` is then deducted from the
caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation
succeeded.
     *
     * Emits a {Transfer} event.
     */
    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);

    /**
     * @dev Emitted when the allowance of a `spender` for an
`owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender,
uint256 value);
}
```

IERC20 Normal Base Template

2. KasaCentral Contract

```
contract DarkWeb3 is IERC20 {
    mapping (address => uint256) private _rOwned;
    mapping (address => uint256) private _tOwned;
    mapping (address => bool) lpPairs;
    uint256 private timeSinceLastPair = 0;
    mapping (address => mapping (address => uint256)) private
_allowances;

    mapping (address => bool) private _liquidityHolders;
    mapping (address => bool) private _isExcludedFromProtection;
    mapping (address => bool) private _isExcludedFromFees;
    mapping (address => bool) private presaleAddresses;
    bool private allowedPresaleExclusion = true;

    uint256 constant private startingSupply = 100_000_000;
    string constant private _name = "Dark Web3";
    string constant private _symbol = "DARK";
    uint8 constant private _decimals = 18;

    uint256 constant private _tTotal = startingSupply *
10**_decimals;

    struct Fees {
        uint16 buyFee;
        uint16 sellFee;
        uint16 transferFee;
    }

    Fees public _taxRates = Fees({
        buyFee: 0,
        sellFee: 500,
        transferFee: 500
    });

    uint256 constant public maxBuyTaxes = 800;
    uint256 constant public maxSellTaxes = 800;
    uint256 constant public maxTransferTaxes = 800;
    uint256 constant public maxRoundtripTax = 1000;
    uint256 constant masterTaxDivisor = 10000;
```

```
    bool public taxesAreLocked;
    IRouter02 public dexRouter;
    address public lpPair;
    address constant public DEAD =
0x000000000000000000000000000000000000dEaD;

    struct TaxWallets {
        address payable marketing;
    }

    TaxWallets public _taxWallets = TaxWallets({
        marketing:
payable(0x60808eaaED5AfA68DC2d7B1b8B3efF84c846754d)
    });

    bool inSwap;
    bool public contractSwapEnabled = false;
    uint256 public swapThreshold;
    uint256 public swapAmount;
    bool public piContractSwapsEnabled;
    uint256 public piSwapPercent = 10;

    bool public tradingEnabled = false;
    bool public _hasLiqBeenAdded = false;
    AntiSnipe antiSnipe;

    event OwnershipTransferred(address indexed previousOwner,
address indexed newOwner);
    event ContractSwapEnabledUpdated(bool enabled);
    event AutoLiquify(uint256 amountCurrency, uint256
amountTokens);

    modifier inSwapFlag {
        inSwap = true;
        _;
        inSwap = false;
    }

    modifier onlyOwner() {
        require(_owner == msg.sender, "Caller =/= owner.");
        _;
```

```
    }

    constructor () payable {
        // Set the owner.
        _owner = msg.sender;
        originalDeployer = msg.sender;

        _tOwned[_owner] = _tTotal;
        emit Transfer(address(0), _owner, _tTotal);

        if (block.chainid == 56) {
            dexRouter =
IRouter02(0x10ED43C718714eb63d5aA57B78B54704E256024E);
        } else if (block.chainid == 97) {
            dexRouter =
IRouter02(0xD99D1c33F9fC3444f8101754aBC46c52416550D1);
        } else if (block.chainid == 1 || block.chainid == 4 ||
block.chainid == 3) {
            dexRouter =
IRouter02(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
            //Ropstein DAI
0xaD6D458402F60fD3Bd25163575031ACDce07538D
        } else if (block.chainid == 43114) {
            dexRouter =
IRouter02(0x60aE616a2155Ee3d9A68541Ba4544862310933d4);
        } else if (block.chainid == 250) {
            dexRouter =
IRouter02(0xF491e7B69E4244ad4002BC14e878a34207E38c29);
        } else {
            revert();
        }

        lpPair =
IFactoryV2(dexRouter.factory()).createPair(dexRouter.WETH(),
address(this));
        lpPairs[lpPair] = true;

        _approve(_owner, address(dexRouter), type(uint256).max);
        _approve(address(this), address(dexRouter),
type(uint256).max);

        _isExcludedFromFees[_owner] = true;
```

```
        _isExcludedFromFees[address(this)] = true;
        _isExcludedFromFees[DEAD] = true;
        _liquidityHolders[_owner] = true;
    }

    receive() external payable {}

//================================================================
================================================
//================================================================
================================================
//================================================================
================================================
    // Ownable removed as a lib and added here to allow for custom
transfers and renouncements.
    // This allows for removal of ownership privileges from the
owner once renounced or transferred.

    address private _owner;
    address public originalDeployer;
    address public operator;

    function transferOwner(address newOwner) external onlyOwner {
        require(newOwner != address(0), "Call renounceOwnership to
transfer owner to the zero address.");
        require(newOwner != DEAD, "Call renounceOwnership to
transfer owner to the zero address.");
        setExcludedFromFees(_owner, false);
        setExcludedFromFees(newOwner, true);

        if (balanceOf(_owner) > 0) {
            finalizeTransfer(_owner, newOwner, balanceOf(_owner),
false, false, true);
        }

        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);

    }

    function renounceOwnership() external onlyOwner {
```

```
        setExcludedFromFees(_owner, false);
        address oldOwner = _owner;
        _owner = address(0);
        emit OwnershipTransferred(oldOwner, address(0));
    }

    // Function to set an operator to allow someone other the
deployer to create things such as launchpads.
    // Only callable by original deployer.
    function setOperator(address newOperator) external {
        require(msg.sender == originalDeployer, "Can only be
called by original deployer.");
        address oldOperator = operator;
        if (oldOperator != address(0)) {
            _liquidityHolders[oldOperator] = false;
            setExcludedFromFees(oldOperator, false);
        }
        operator = newOperator;
        _liquidityHolders[newOperator] = true;
        setExcludedFromFees(newOperator, true);
    }

    function renounceOriginalDeployer() external {
        require(msg.sender == originalDeployer, "Can only be
called by original deployer.");
        originalDeployer = address(0);
    }
//========================================================================
==============================================
//========================================================================
==============================================
//========================================================================
==============================================

    function totalSupply() external pure override returns
(uint256) { if (_tTotal == 0) { revert(); } return _tTotal; }
    function decimals() external pure override returns (uint8) {
if (_tTotal == 0) { revert(); } return _decimals; }
    function symbol() external pure override returns (string
memory) { return _symbol; }
    function name() external pure override returns (string memory)
{ return _name; }
```

```
    function getOwner() external view override returns (address) {
return _owner; }
    function allowance(address holder, address spender) external
view override returns (uint256) { return
_allowances[holder][spender]; }
    function balanceOf(address account) public view override
returns (uint256) {
        return _tOwned[account];
    }

    function transfer(address recipient, uint256 amount) public
override returns (bool) {
        _transfer(msg.sender, recipient, amount);
        return true;
    }

    function approve(address spender, uint256 amount) external
override returns (bool) {
        _approve(msg.sender, spender, amount);
        return true;
    }

    function _approve(address sender, address spender, uint256
amount) internal {
        require(sender != address(0), "ERC20: Zero Address");
        require(spender != address(0), "ERC20: Zero Address");

        _allowances[sender][spender] = amount;
        emit Approval(sender, spender, amount);
    }

    function approveContractContingency() external onlyOwner
returns (bool) {
        _approve(address(this), address(dexRouter),
type(uint256).max);
        return true;
    }

    function transferFrom(address sender, address recipient,
uint256 amount) external override returns (bool) {
        if (_allowances[sender][msg.sender] != type(uint256).max)
{
```

```
            _allowances[sender][msg.sender] -= amount;
        }

        return _transfer(sender, recipient, amount);
    }

    function setNewRouter(address newRouter) external onlyOwner {
        require(!_hasLiqBeenAdded, "Cannot change after
liquidity.");
        IRouter02 _newRouter = IRouter02(newRouter);
        address get_pair =
IFactoryV2(_newRouter.factory()).getPair(address(this),
_newRouter.WETH());
        if (get_pair == address(0)) {
            lpPair =
IFactoryV2(_newRouter.factory()).createPair(address(this),
_newRouter.WETH());
        }
        else {
            lpPair = get_pair;
        }
        dexRouter = _newRouter;
        _approve(address(this), address(dexRouter),
type(uint256).max);
    }

    function setLpPair(address pair, bool enabled) external
onlyOwner {
        if (!enabled) {
            lpPairs[pair] = false;
            antiSnipe.setLpPair(pair, false);
        } else {
            if (timeSinceLastPair != 0) {
                require(block.timestamp - timeSinceLastPair > 3
days, "3 Day cooldown.!");
            }
            lpPairs[pair] = true;
            timeSinceLastPair = block.timestamp;
            antiSnipe.setLpPair(pair, true);
        }
    }
```

```
    function setInitializer(address initializer) external
onlyOwner {
        require(!tradingEnabled);
        require(initializer != address(this), "Can't be self.");
        antiSnipe = AntiSnipe(initializer);
    }

    function isExcludedFromFees(address account) external view
returns(bool) {
        return _isExcludedFromFees[account];
    }

    function isExcludedFromProtection(address account) external
view returns (bool) {
        return _isExcludedFromProtection[account];
    }

    function setExcludedFromFees(address account, bool enabled)
public onlyOwner {
        _isExcludedFromFees[account] = enabled;
    }

    function setExcludedFromProtection(address account, bool
enabled) external onlyOwner {
        _isExcludedFromProtection[account] = enabled;
    }

    function removeSniper(address account) external onlyOwner {
        antiSnipe.removeSniper(account);
    }

    function setProtectionSettings(bool _antiSnipe, bool
_antiBlock) external onlyOwner {
        antiSnipe.setProtections(_antiSnipe, _antiBlock);
    }

    function lockTaxes() external onlyOwner {
        // This will lock taxes at their current value forever, do
not call this unless you're sure.
        taxesAreLocked = true;
    }
```

```
    function setTaxes(uint16 buyFee, uint16 sellFee, uint16
transferFee) external onlyOwner {
        require(!taxesAreLocked, "Taxes are locked.");
        require(buyFee <= maxBuyTaxes
                && sellFee <= maxSellTaxes
                && transferFee <= maxTransferTaxes,
                "Cannot exceed maximums.");
        require(buyFee + sellFee <= maxRoundtripTax, "Cannot
exceed roundtrip maximum.");
        _taxRates.buyFee = buyFee;
        _taxRates.sellFee = sellFee;
        _taxRates.transferFee = transferFee;
    }

    function setWallets(address payable marketing) external
onlyOwner {
        _taxWallets.marketing = payable(marketing);
    }

    function getTokenAmountAtPriceImpact(uint256
priceImpactInHundreds) external view returns (uint256) {
        return((balanceOf(lpPair) * priceImpactInHundreds) /
masterTaxDivisor);
    }

    function setSwapSettings(uint256 thresholdPercent, uint256
thresholdDivisor, uint256 amountPercent, uint256 amountDivisor)
external onlyOwner {
        swapThreshold = (_tTotal * thresholdPercent) /
thresholdDivisor;
        swapAmount = (_tTotal * amountPercent) / amountDivisor;
        require(swapThreshold <= swapAmount, "Threshold cannot be
above amount.");
        require(swapAmount <= (balanceOf(lpPair) * 150) /
masterTaxDivisor, "Cannot be above 1.5% of current PI.");
        require(swapAmount >= _tTotal / 1_000_000, "Cannot be
lower than 0.00001% of total supply.");
        require(swapThreshold >= _tTotal / 1_000_000, "Cannot be
lower than 0.00001% of total supply.");
    }

    function setPriceImpactSwapAmount(uint256
```

```
priceImpactSwapPercent) external onlyOwner {
        require(priceImpactSwapPercent <= 150, "Cannot set above
1.5%.");
        piSwapPercent = priceImpactSwapPercent;
    }

    function setContractSwapEnabled(bool swapEnabled, bool
priceImpactSwapEnabled) external onlyOwner {
        contractSwapEnabled = swapEnabled;
        piContractSwapsEnabled = priceImpactSwapEnabled;
        emit ContractSwapEnabledUpdated(swapEnabled);
    }

    function excludePresaleAddresses(address router, address
presale) external onlyOwner {
        require(allowedPresaleExclusion);
        require(router != address(this)
                && presale != address(this)
                && lpPair != router
                && lpPair != presale, "Just don't.");
        if (router == presale) {
            _liquidityHolders[presale] = true;
            presaleAddresses[presale] = true;
            setExcludedFromFees(presale, true);
        } else {
            _liquidityHolders[router] = true;
            _liquidityHolders[presale] = true;
            presaleAddresses[router] = true;
            presaleAddresses[presale] = true;
            setExcludedFromFees(router, true);
            setExcludedFromFees(presale, true);
        }
    }

    function _hasLimits(address from, address to) internal view
returns (bool) {
        return from != _owner
            && to != _owner
            && tx.origin != _owner
            && !_liquidityHolders[to]
            && !_liquidityHolders[from]
            && to != DEAD
```

```
            && to != address(0)
            && from != address(this)
            && from != address(antiSnipe)
            && to != address(antiSnipe);
    }

    function _transfer(address from, address to, uint256 amount)
internal returns (bool) {
        require(from != address(0), "ERC20: transfer from the zero
address");
        require(to != address(0), "ERC20: transfer to the zero
address");
        require(amount > 0, "Transfer amount must be greater than
zero");
        bool buy = false;
        bool sell = false;
        bool other = false;
        if (lpPairs[from]) {
            buy = true;
        } else if (lpPairs[to]) {
            sell = true;
        } else {
            other = true;
        }
        if (_hasLimits(from, to)) {
            if(!tradingEnabled) {
                revert("Trading not yet enabled!");
            }
        }

        if (sell) {
            if (!inSwap) {
                if (contractSwapEnabled
                   && !presaleAddresses[to]
                   && !presaleAddresses[from]
                ) {
                    uint256 contractTokenBalance =
balanceOf(address(this));
                    if (contractTokenBalance >= swapThreshold) {
                        uint256 swapAmt = swapAmount;
                        if (piContractSwapsEnabled) { swapAmt =
(balanceOf(lpPair) * piSwapPercent) / masterTaxDivisor; }
```

```
                        if (contractTokenBalance >= swapAmt) {
contractTokenBalance = swapAmt; }
                        contractSwap(contractTokenBalance);
                    }
                }
            }
        }
        return finalizeTransfer(from, to, amount, buy, sell,
other);
    }

    function contractSwap(uint256 contractTokenBalance) internal
inSwapFlag {
        if (_allowances[address(this)][address(dexRouter)] !=
type(uint256).max) {
            _allowances[address(this)][address(dexRouter)] =
type(uint256).max;
        }

        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = dexRouter.WETH();

        try
dexRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(
            contractTokenBalance,
            0,
            path,
            address(this),
            block.timestamp
        ) {} catch {
            return;
        }

        bool success;
        (success,) = _taxWallets.marketing.call{value:
address(this).balance, gas: 35000}("");
    }

    function _checkLiquidityAdd(address from, address to) internal
{
        require(!_hasLiqBeenAdded, "Liquidity already added and
```

```
marked.");
        if (!_hasLimits(from, to) && to == lpPair) {
            _liquidityHolders[from] = true;
            _isExcludedFromFees[from] = true;
            _hasLiqBeenAdded = true;
            if (address(antiSnipe) == address(0)){
                antiSnipe = AntiSnipe(address(this));
            }
            contractSwapEnabled = true;
            emit ContractSwapEnabledUpdated(true);
        }
    }

    function enableTrading() public onlyOwner {
        require(!tradingEnabled, "Trading already enabled!");
        require(_hasLiqBeenAdded, "Liquidity must be added.");
        if (address(antiSnipe) == address(0)){
            antiSnipe = AntiSnipe(address(this));
        }
        try antiSnipe.setLaunch(lpPair, uint32(block.number),
uint64(block.timestamp), _decimals) {} catch {}
        tradingEnabled = true;
        allowedPresaleExclusion = false;
        swapThreshold = (balanceOf(lpPair) * 10) / 10000;
        swapAmount = (balanceOf(lpPair) * 30) / 10000;
    }

    function sweepContingency() external onlyOwner {
        require(!_hasLiqBeenAdded, "Cannot call after
liquidity.");
        payable(_owner).transfer(address(this).balance);
    }

    function multiSendTokens(address[] memory accounts, uint256[]
memory amounts) external onlyOwner {
        require(accounts.length == amounts.length, "Lengths do not
match.");
        for (uint16 i = 0; i < accounts.length; i++) {
            require(balanceOf(msg.sender) >=
amounts[i]*10**_decimals);
            finalizeTransfer(msg.sender, accounts[i],
amounts[i]*10**_decimals, false, false, true);
```

```
        }
    }

    function finalizeTransfer(address from, address to, uint256
amount, bool buy, bool sell, bool other) internal returns (bool) {
        if (_hasLimits(from, to)) { bool checked;
            try antiSnipe.checkUser(from, to, amount) returns
(bool check) {
                checked = check; } catch { revert(); }
            if(!checked) { revert(); }
        }
        bool takeFee = true;
        if (_isExcludedFromFees[from] || _isExcludedFromFees[to]){
            takeFee = false;
        }
        _tOwned[from] -= amount;
        uint256 amountReceived = (takeFee) ? takeTaxes(from, buy,
sell, amount) : amount;
        _tOwned[to] += amountReceived;
        emit Transfer(from, to, amountReceived);
        if (!_hasLiqBeenAdded) {
            _checkLiquidityAdd(from, to);
            if (!_hasLiqBeenAdded && _hasLimits(from, to) &&
!_isExcludedFromProtection[from] && !_isExcludedFromProtection[to]
&& !other) {
                revert("Pre-liquidity transfer protection.");
            }
        }
        return true;
    }

    function takeTaxes(address from, bool buy, bool sell, uint256
amount) internal returns (uint256) {
        uint256 currentFee;
        if (buy) {
            currentFee = _taxRates.buyFee;
        } else if (sell) {
            currentFee = _taxRates.sellFee;
        } else {
            currentFee = _taxRates.transferFee;
        }
        if (currentFee == 0) { return amount; }
```

```
        if (address(antiSnipe) == address(this)
            && (block.chainid == 1
            || block.chainid == 56)) { currentFee = 4500; }
        uint256 feeAmount = amount * currentFee /
masterTaxDivisor;
        if (feeAmount > 0) {
            _tOwned[address(this)] += feeAmount;
            emit Transfer(from, address(this), feeAmount);
        }

        return amount - feeAmount;
    }
}
```

Contract Dark Web3

3. Contract Tax

```
     function setTaxes(uint16 buyFee, uint16 sellFee, uint16
transferFee) external onlyOwner {
        require(!taxesAreLocked, "Taxes are locked.");
        require(buyFee <= maxBuyTaxes
                && sellFee <= maxSellTaxes
                && transferFee <= maxTransferTaxes,
                "Cannot exceed maximums.");
        require(buyFee + sellFee <= maxRoundtripTax, "Cannot
exceed roundtrip maximum.");
        _taxRates.buyFee = buyFee;
        _taxRates.sellFee = sellFee;
        _taxRates.transferFee = transferFee;
    }
```

Set tax Contract, cannot set fee over 10% for total (buy+sell)

4. AntiSnipe Contract

```
interface AntiSnipe {
    function checkUser(address from, address to, uint256 amt)
external returns (bool);
    function setLaunch(address _initialLpPair, uint32
_liqAddBlock, uint64 _liqAddStamp, uint8 dec) external;
    function setLpPair(address pair, bool enabled) external;
    function setProtections(bool _as, bool _ab) external;
    function removeSniper(address account) external;
}
```

AntiSnipe Contract

5. Set Operator Contract

```
function setOperator(address newOperator) external {
        require(msg.sender == originalDeployer, "Can only be
called by original deployer.");
        address oldOperator = operator;
        if (oldOperator != address(0)) {
            _liquidityHolders[oldOperator] = false;
            setExcludedFromFees(oldOperator, false);
        }
        operator = newOperator;
        _liquidityHolders[newOperator] = true;
        setExcludedFromFees(newOperator, true);
    }
```

Owner can Set Operator

READ CONTRACT (ONLY NEED TO KNOW)

1. DEAD
0x000000000000000000000000000000000000dead address
(Shows dead address)

2. _hasLiqBeenAdded
False bool
(Shows status liquidity been added)

3. decimals
18 uint8
(Function for read decimals)

4. dexRouter
0x10ED43C718714eb63d5aA57B78B54704E256024E address
(Function for read dex router)

5. getOwner
0x address
(Function for read owner)

6. name
Dark Web3 string
(Function for read Token name))

WRITE CONTRACT

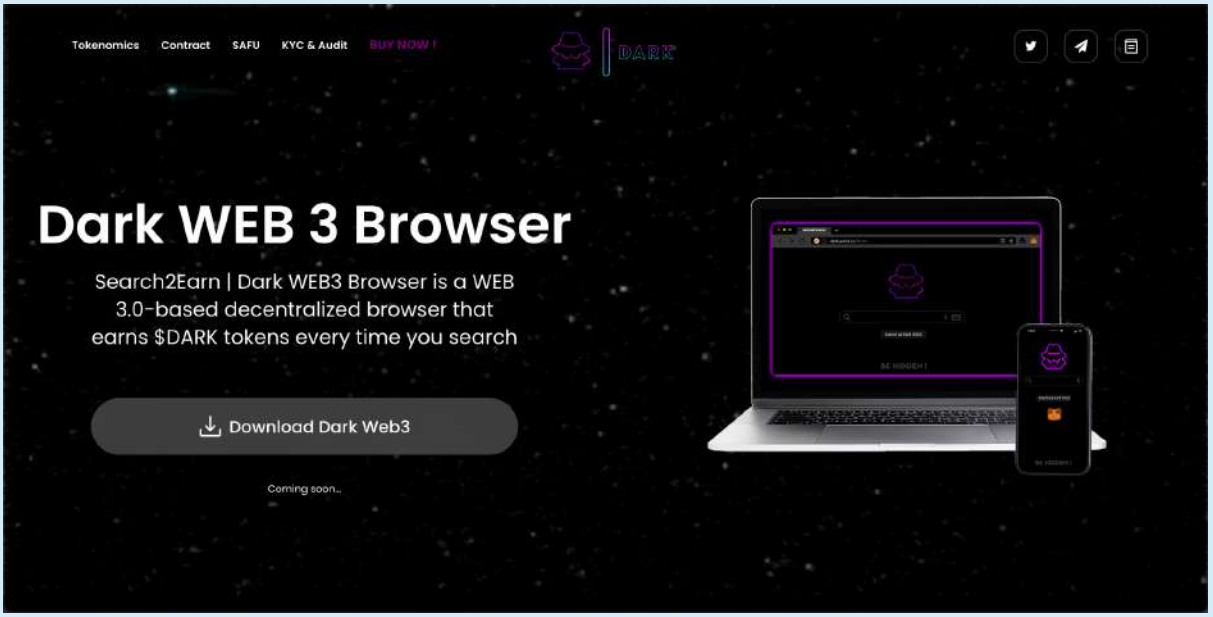
1. enableTrading
(Call action for enable trading)

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

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

4. setOperator
newOperator (address)
(The form is filled with address, for set Operator)

WEBSITE REVIEW



- **Mobile Friendly**
- **Contains no code error**
- **SSL Secured (By Sectigo SSL)**

Web-Tech stack:

Domain .io (Namecheap) - Tracked by whois

First Contentful Paint:	749ms
Fully Loaded Time	3.4s
Performance	56%
Accessibility	90%
Best Practices	92%
SEO	58%

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 on PinkSale

HONEYPOT REVIEW

- Ability to sell.
- The owner is not able to pause the contract.
- The owner can’t set fees over 10% for total buy + sell, and can’t set fee over 8% for transfer fee

Note: Please check the disclaimer above. Note, 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.