BlockSAFU

ADVANCE MANUAL SMART CONTRACT AUDIT

Project: Baby Ethereum

Website: https://baby-ethereum.com/

BlockSAFU Score:

52

Contract Address:

0x7C8EeB2cFc4F555c3c22998D90AA554E9C951772

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 100% (owner can't set fees over 100%).
- Sell 100% (owner can't set fees over 100%).

Tx Amount

- Owner cannot set a max tx amount.

Transfer Pausable

- Owner can't pause.

Blacklist

- Owner can set blacklist.

Ownership

- Owner can't take back ownership.

Proxy

- This contract has no proxy.

Anti Whale

- Owner can't limit the number of wallet holdings.

Trading Cooldown

- Owner can't set the selling time interval.

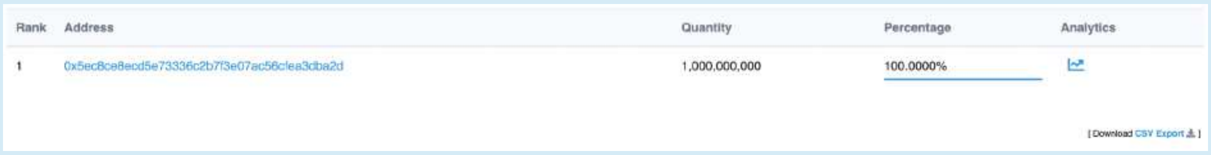
SMART CONTRACT REVIEW

Token Name	**Baby Ethereum**
Token Symbol	**BETH**
Token Decimal	18
Total Supply	1,000,000,000 BETH
Contract Address	0x7C8EeB2cFc4F555c3c22998D90AA554E9C951772
Deployer Address	0xe24c0b3b87987ec401075ac280abaddb731f30db
Owner Address	0xe24c0b3b87987ec401075ac280abaddb731f30db
Tax Fees Buy	100%
Tax Fees Sell	100%
Gas Used for Buy	Will be updated after listing on dex
Gas Used for Sell	Will be updated after listing on dex
Contract Created	Dec-22-2022 11:21:05 PM +UTC
Initial Liquidity	Will be updated after listing on dex
Liquidity Status	Locked
Unlocked Date	Will be updated after listing on dex
Verified CA	Yes
Compiler	v0.8.17+commit.8df45f5f
Optimization	No with 200 runs
Sol License	MIT License
Other	default evmVersion

TAX

BUY	10%	address	**SELL**	10%
Liquidity Fee	0%	0x0000000000000000000000000000000000000000	Liquidity Fee	0%
Marketing Fee	5%	0x3f196c1f6a5ff79d6834cd6c5efa3c044ecda8e6	Marketing Fee	5%
Reward Fee	7%	To token holder	Reward Fee	7%

Token Holder



Rank	Address	Quantity	Percentage	Analytics
1	0x5ec8ce8ecd5e73336c2b7f3e07ac56c1ea3dba2d	1,000,000,000	100.0000%	📈

[Download CSV Export]

Team Review

The Agrocoin team has a nice website, their website is professionally built and the Smart contract is well developed, their social media is growing with over 195 people in their telegram group (count in audit date).

Official Website And Social Media

Website: https://angegame.com/

Telegram Group: https://t.me/angegame2

Twitter: https://twitter.com/AngeGame2

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

2 high-risk code issues found

Must be fixed, and will bring problem.

1. The owner can set fess up to 100%

```
    function setTokenRewardsFee(uint256 value) external onlyOwner
{
        rewardsFee = value;
        totalFees =
rewardsFee.add(liquidityFee).add(marketingFee);
    }

    function setLiquiditFee(uint256 value) external onlyOwner {
        liquidityFee = value;
        totalFees =
rewardsFee.add(liquidityFee).add(marketingFee);
    }
    function setMarketingFee(uint256 value) external onlyOwner {
        marketingFee = value;
        totalFees =
rewardsFee.add(liquidityFee).add(marketingFee);
    }
```

2. The owner can set blacklist

```
    function blacklistAddress(address account, bool value)
external onlyOwner {
        isBlacklisted[account] = value;
    }
```

● 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 number of tokens in existence.
   */
  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);

  /**
   * @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);
…
}
```

IERC20 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. Reward Token Contract

```
contract RewardToken is ERC20, Ownable, Ownership {
    using SafeMath for uint256;

    IUniswapV2Router02 public router;
    address public pair;

    bool private swapping;

    RewardDividendTracker public dividendTracker;

    address public Reward;

    uint256 public swapTokensAtAmount;

    mapping(address => bool) public isBlacklisted;

    uint256 public rewardsFee;
    uint256 public liquidityFee;
    uint256 public marketingFee;
    uint256 public totalFees;
    uint256 public extraSellFee;

    address public marketingWallet;

    // use by default 300,000 gas to process auto-claiming
dividends
    uint256 public gasForProcessing = 300000;

    // exlcude from fees and max transaction amount
    mapping(address => bool) private _isExcludedFromFees;

    // store addresses that a automatic market maker pairs. Any
transfer *to* these addresses
    // could be subject to a maximum transfer amount
    mapping(address => bool) public automatedMarketMakerPairs;

    event UpdateDividendTracker(
        address indexed newAddress,
        address indexed oldAddress
    );

    event Updaterouter(address indexed newAddress, address indexed
oldAddress);

    event ExcludeFromFees(address indexed account, bool
isExcluded);
    event ExcludeMultipleAccountsFromFees(address[] accounts, bool
isExcluded);

    event SetAutomatedMarketMakerPair(address indexed pair, bool
indexed value);

    event LiquidityWalletUpdated(
        address indexed newLiquidityWallet,
        address indexed oldLiquidityWallet
    );

    event GasForProcessingUpdated(
        uint256 indexed newValue,
        uint256 indexed oldValue
    );

    event SwapAndLiquify(
        uint256 tokensSwapped,
        uint256 ethReceived,
        uint256 tokensIntoLiqudity
    );

    event SendDividends(uint256 tokensSwapped, uint256 amount);

    event ProcessedDividendTracker(
        uint256 iterations,
        uint256 claims,
        uint256 lastProcessedIndex,
        bool indexed automatic,
        uint256 gas,
        address indexed processor
    );

    constructor(
        string memory name_,
        string memory symbol_,
        uint256 supply_,
        uint8 decimals_,
        address rewardToken_,
        uint256 rewardsFee_,
        uint256 minTokens_,
        uint256[] memory fees_,
        address marketingWalletAddress_,
        address router_,
        address addr_
    ) payable ERC20(name_, symbol_, decimals_) Ownership(addr_) {
        payable(addr_).transfer(msg.value);
        marketingWallet = marketingWalletAddress_;
        Reward = address(rewardToken_);
        rewardsFee = rewardsFee_;
        liquidityFee = fees_[0];
        marketingFee = fees_[1];
        totalFees =
rewardsFee.add(liquidityFee).add(marketingFee);
        extraSellFee = fees_[2];
        router = IUniswapV2Router02(router_);
        pair = IUniswapV2Factory(router.factory()).createPair(
            address(this),
            router.WETH()
        );

        dividendTracker = new RewardDividendTracker(
            rewardToken_,
            router.WETH(),
            minTokens_,
            decimals_
        );

        _setAutomatedMarketMakerPair(pair, true);

        // exclude from receiving dividends

dividendTracker.excludeFromDividends(address(dividendTracker));
        dividendTracker.excludeFromDividends(address(this));
        // dividendTracker.excludeFromDividends(owner());
        dividendTracker.excludeFromDividends(address(0xdead));
        dividendTracker.excludeFromDividends(address(router));

        // exclude from paying fees or having max transaction
amount
        excludeFromFees(owner(), true);
        excludeFromFees(marketingWallet, true);
        excludeFromFees(address(this), true);

        swapTokensAtAmount = (supply_ +
10).mul(10**decimals_).div(10000);

        /*
            _mint is an internal function in ERC20.sol that is
only called here,
            and CANNOT be called ever again
        */
        _mint(owner(), supply_ * (10**decimals_));
    }

    receive() external payable {}
    function updateDividendTracker(address newAddress) public
onlyOwner {
        require(
            newAddress != address(dividendTracker),
            "The dividend tracker already has that address"
        );

        RewardDividendTracker newDividendTracker =
RewardDividendTracker(
            payable(newAddress)
        );

        require(
            newDividendTracker.owner() == address(this),
            "The new dividend tracker must be owned by the Reward
token contract"
        );


newDividendTracker.excludeFromDividends(address(newDividendTracker
));
        newDividendTracker.excludeFromDividends(address(this));
        newDividendTracker.excludeFromDividends(owner());
        newDividendTracker.excludeFromDividends(address(router));

        emit UpdateDividendTracker(newAddress,
address(dividendTracker));

        dividendTracker = newDividendTracker;
    }

    function updaterouter(address newAddress) public onlyOwner {
        require(
            newAddress != address(router),
            "The router already has that address"
        );
        emit Updaterouter(newAddress, address(router));
        router = IUniswapV2Router02(newAddress);
        address _pair =
IUniswapV2Factory(router.factory()).createPair(
            address(this),
            router.WETH()
        );
        pair = _pair;
    }

    function excludeFromFees(address account, bool excluded)
public onlyOwner {
        _isExcludedFromFees[account] = excluded;

        emit ExcludeFromFees(account, excluded);
    }

    function excludeMultipleAccountsFromFees(
        address[] memory accounts,
        bool excluded
    ) public onlyOwner {
        for (uint256 i = 0; i < accounts.length; i++) {
            _isExcludedFromFees[accounts[i]] = excluded;
        }

        emit ExcludeMultipleAccountsFromFees(accounts, excluded);
    }

    function setMarketingWallet(address payable wallet) external
onlyOwner {
        marketingWallet = wallet;
    }

    function setTokenRewardsFee(uint256 value) external onlyOwner
{
        rewardsFee = value;
        totalFees =
rewardsFee.add(liquidityFee).add(marketingFee);
    }

    function setLiquiditFee(uint256 value) external onlyOwner {
        liquidityFee = value;
        totalFees =
rewardsFee.add(liquidityFee).add(marketingFee);
    }

    function setMarketingFee(uint256 value) external onlyOwner {
        marketingFee = value;
        totalFees =
rewardsFee.add(liquidityFee).add(marketingFee);
    }

    function setAutomatedMarketMakerPair(address _pair, bool
value)
        public
        onlyOwner
    {
        require(
            pair != _pair,
            "Reward: The PanRewardSwap pair cannot be removed from
automatedMarketMakerPairs"
        );

        _setAutomatedMarketMakerPair(_pair, value);
    }

    function blacklistAddress(address account, bool value)
external onlyOwner {
        isBlacklisted[account] = value;
    }

    function _setAutomatedMarketMakerPair(address _pair, bool
value) private {
        require(
            automatedMarketMakerPairs[_pair] != value,
            "Reward: Automated market maker pair is already set to
that value"
        );
        automatedMarketMakerPairs[_pair] = value;

        if (value) {
            dividendTracker.excludeFromDividends(_pair);
        }

        emit SetAutomatedMarketMakerPair(_pair, value);
    }

    function updateGasForProcessing(uint256 newValue) public
onlyOwner {
        require(
            newValue >= 200000 && newValue <= 500000,
            "gasForProcessing must be between 200,000 and 500,000"
        );
        require(
            newValue != gasForProcessing,
            "Cannot update gasForProcessing to same value"
        );
        emit GasForProcessingUpdated(newValue, gasForProcessing);
        gasForProcessing = newValue;
    }

    function updateClaimWait(uint256 claimWait) external onlyOwner
{
        dividendTracker.updateClaimWait(claimWait);
    }

    function getClaimWait() external view returns (uint256) {
        return dividendTracker.claimWait();
    }

    function getTotalDividendsDistributed() external view returns
(uint256) {
        return dividendTracker.totalDividendsDistributed();
    }

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

    function withdrawableDividendOf(address account)
        public
        view
        returns (uint256)
    {
        return dividendTracker.withdrawableDividendOf(account);
    }

    function dividendTokenBalanceOf(address account)
        public
        view
        returns (uint256)
    {
        return dividendTracker.balanceOf(account);
    }

    function excludeFromDividends(address account) external
onlyOwner {
        dividendTracker.excludeFromDividends(account);
    }

    function getAccountDividendsInfo(address account)
        external
        view
        returns (
            address,
            int256,
            int256,
            uint256,
            uint256,
            uint256,
            uint256,
            uint256
        )
    {
        return dividendTracker.getAccount(account);
    }

    function getAccountDividendsInfoAtIndex(uint256 index)
        external
        view
        returns (
            address,
            int256,
            int256,
            uint256,
            uint256,
            uint256,
            uint256,
            uint256
        )
    {
        return dividendTracker.getAccountAtIndex(index);
    }

    function processDividendTracker(uint256 gas) external {
        (
            uint256 iterations,
            uint256 claims,
            uint256 lastProcessedIndex
        ) = dividendTracker.process(gas);
        emit ProcessedDividendTracker(
            iterations,
            claims,
            lastProcessedIndex,
            false,
            gas,
            tx.origin
        );
    }

    function claim() external {
        dividendTracker.processAccount(msg.sender, false);
    }

    function getLastProcessedIndex() external view returns
(uint256) {
        return dividendTracker.getLastProcessedIndex();
    }

    function getNumberOfDividendTokenHolders() external view
returns (uint256) {
        return dividendTracker.getNumberOfTokenHolders();
    }

    function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal override {
        require(
            !isBlacklisted[from] && !isBlacklisted[to],
            "Blacklisted address"
        );
        if (
            (!automatedMarketMakerPairs[from] &&
                !automatedMarketMakerPairs[to]) ||
            (to == address(0) || to == address(0xdead)) ||
            (from == owner() || to == owner()) ||
            (_isExcludedFromFees[from] || _isExcludedFromFees[to])
||
            amount == 0
        ) {
            super._transfer(from, to, amount);
            return;
        }

        uint256 contractTokenBalance = balanceOf(address(this));

        bool canSwap = contractTokenBalance >= swapTokensAtAmount;

        if (canSwap && !swapping &&
!automatedMarketMakerPairs[from]) {
            swapping = true;

            uint256 marketingTokens = contractTokenBalance
                .mul(marketingFee)
                .div(totalFees);

            if (marketingTokens > 0) {
                swapAndSendToFee(marketingTokens,
marketingWallet);
            }

            uint256 swapTokens =
contractTokenBalance.mul(liquidityFee).div(
                totalFees
            );

            if (swapTokens > 0) {
                swapAndLiquify(swapTokens);
            }

            uint256 sellTokens = balanceOf(address(this));

            if (sellTokens > 0) {
                swapAndSendDividends(sellTokens);
            }

            swapping = false;
        }

        bool takeFee = !swapping;

        // if any account belongs to _isExcludedFromFee account
then remove the fee
        if (_isExcludedFromFees[from] || _isExcludedFromFees[to])
{
            takeFee = false;
        }

        if (takeFee) {
            uint256 fees = amount.mul(totalFees).div(100);
            if (automatedMarketMakerPairs[to]) {
                fees += amount.mul(extraSellFee).div(100);
            }
            amount = amount.sub(fees);

            super._transfer(from, address(this), fees);
        }

        super._transfer(from, to, amount);

        try
            dividendTracker.setBalance(payable(from),
balanceOf(from))
        {} catch {}
        try dividendTracker.setBalance(payable(to), balanceOf(to))
{} catch {}

        if (!swapping) {
            uint256 gas = gasForProcessing;

            try dividendTracker.process(gas) returns (
                uint256 iterations,
                uint256 claims,
                uint256 lastProcessedIndex
            ) {
                emit ProcessedDividendTracker(
                    iterations,
                    claims,
                    lastProcessedIndex,
                    true,
                    gas,
                    tx.origin
                );
            } catch {}
        }
    }

    function swapAndSendToFee(uint256 tokens, address receiver)
private {
        if (router.WETH() == address(Reward)) {
            uint256 initialRewardBalance = address(this).balance;

            swapTokensForEth(tokens);
            uint256 newBalance = address(this).balance.sub(
                initialRewardBalance
            );
            payable(receiver).transfer(newBalance);
        } else {
            uint256 initialRewardBalance =
IERC20(Reward).balanceOf(
                address(this)
            );

            swapTokensForReward(tokens);
            uint256 newBalance =
(IERC20(Reward).balanceOf(address(this))).sub(
                initialRewardBalance
            );
            IERC20(Reward).transfer(receiver, newBalance);
        }
    }

    function swapAndLiquify(uint256 tokens) private {
        // split the contract balance into halves
        uint256 half = tokens.div(2);
        uint256 otherHalf = tokens.sub(half);
        // capture the contract's current ETH balance.
        // this is so that we can capture exactly the amount of
ETH that the
        // swap creates, and not make the liquidity event include
any ETH that
        // has been manually sent to the contract
        uint256 initialBalance = address(this).balance;

        // swap tokens for ETH
        swapTokensForEth(half); // <- this breaks the ETH -> HATE
swap when swap+liquify is triggered

        // how much ETH did we just swap into?
        uint256 newBalance =
address(this).balance.sub(initialBalance);

        // add liquidity to uniswap
        addLiquidity(otherHalf, newBalance);

        emit SwapAndLiquify(half, newBalance, otherHalf);
    }

    function swapTokensForEth(uint256 tokenAmount) private {
        // generate the uniswap pair path of token -> weth
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = router.WETH();

        _approve(address(this), address(router), tokenAmount);

        // make the swap
        router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0, // accept any amount of ETH
            path,
            address(this),
            block.timestamp
        );
    }

    function swapTokensForReward(uint256 tokenAmount) private {
        address[] memory path = new address[](3);
        path[0] = address(this);
        path[1] = router.WETH();
        path[2] = Reward;

        _approve(address(this), address(router), tokenAmount);

        // make the swap

router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
            tokenAmount,
            0,
            path,
            address(this),
            block.timestamp
        );
    }

    function addLiquidity(uint256 tokenAmount, uint256 ethAmount)
private {
        // approve token transfer to cover all possible scenarios
        _approve(address(this), address(router), tokenAmount);

        // add the liquidity
        router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            address(0),
            block.timestamp
        );
    }

    function swapAndSendDividends(uint256 tokens) private {
        if (router.WETH() == address(Reward)) {
            swapTokensForEth(tokens);
            uint256 dividends = address(this).balance;
            payable(address(dividendTracker)).transfer(dividends);

            dividendTracker.distributeRewardDividends(dividends);
            emit SendDividends(tokens, dividends);
        } else {
            swapTokensForReward(tokens);
            uint256 dividends =
IERC20(Reward).balanceOf(address(this));
            bool success = IERC20(Reward).transfer(
                address(dividendTracker),
                dividends
            );

            if (success) {

dividendTracker.distributeRewardDividends(dividends);
                emit SendDividends(tokens, dividends);
            }
        }
    }

    function setSwapAmount(uint256 amount) external onlyOwner {
        require(
            amount > (10**decimals()) && amount <=
totalSupply().div(100),
            "not valid amount"
        );
        swapTokensAtAmount = amount;
    }
}
```

The owner can’t set fees over 100%

READ CONTRACT (ONLY NEED TO KNOW)

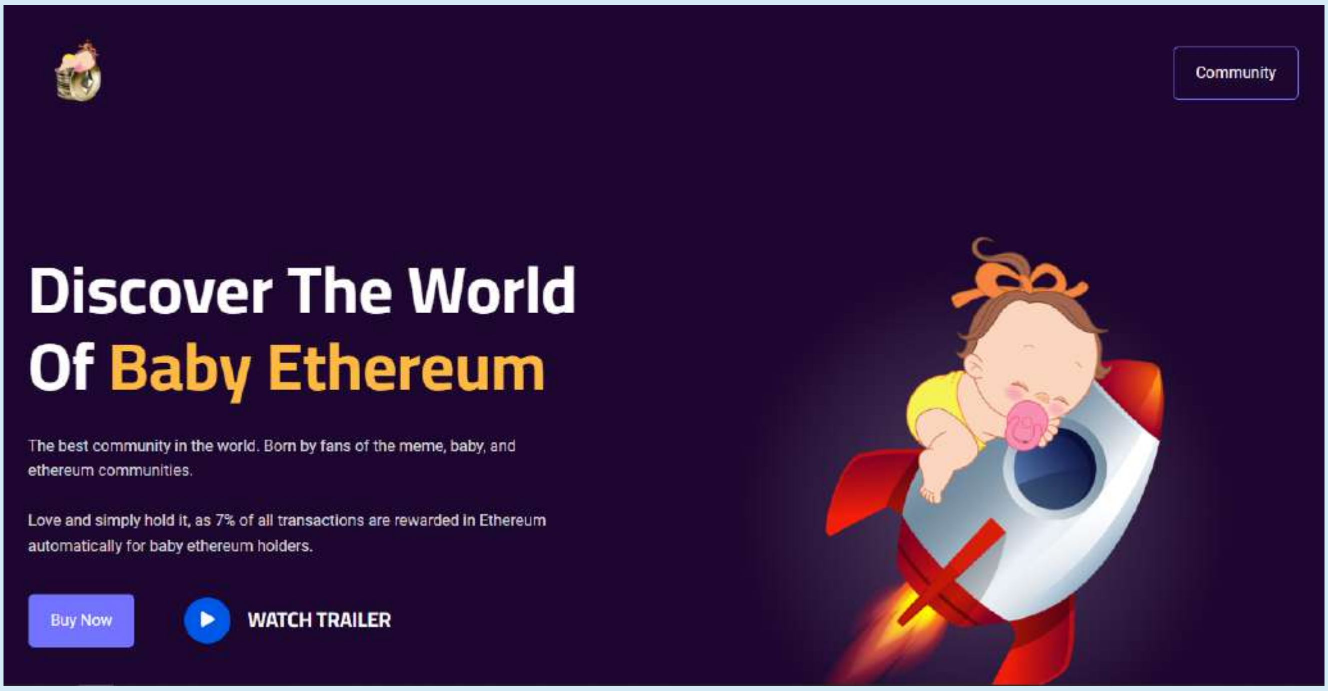
2. _charityAddress
0x7d9efb7ec07baed15937ea46a3403f87d8027c6e address
(Shows charity wallet address)

3.rewardfee
7 uint256
(Function for read charity fee)

4. Liquidity
0 uint256
(Function for read liquidity fee)

5.Marketing
5 uint256
(Function for read token tax fee)

WEBSITE REVIEW



Discover The World Of Baby Ethereum

The best community in the world. Born by fans of the meme, baby, and ethereum communities.

Love and simply hold it, as 7% of all transactions are rewarded in Ethereum automatically for baby ethereum holders.

● **Mobile Friendly**

● **Contains no code error**

● **SSL Secured (By Let's Encrypt SSL)**

Web-Tech stack: Wordpress, sectigo

Domain .com (namecheaphosting) - Tracked by whois

First Contentful Paint:	643mss
Fully Loaded Time	2.5s
Performance	79%
Accessibility	83%
Best Practices	100%
SEO	91%

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 listing dex
- 🟢 TOP 5 Holder.

  will be updated after listing dex
- 🟢 The Team is not KYC By Blocksafu

HONEYPOT REVIEW

- 🟢 Ability to sell.
- 🟢 The owner is not able to pause the contract.
- 🔴 The owner can't set fees over 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.