Armors Labs

Hound Token

Smart Contract Audit

Hound Token Audit Summary

Project name : Hound Token Contract

Project address: None

Code URL : https://bscscan.com/address/0x1e4402fa427a7a835fc64ea6d051404ce767a569#code

Commit : None

Project target : Hound Token Contract Audit

Blockchain : Binance Smart Chain（BSC）

Test result : PASSED

Audit Info

Audit NO : 0X202203150026

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

Hound Token Audit

The Hound Token team asked us to review and audit their Hound Token contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
Hound Token Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2022-03-15

Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Hound Token contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 3 months from the date of output.

Disclaimer

Armors Labs Reports is not and should not be regarded as an "approval" or "disapproval" of any particular project or team. These reports are not and should not be regarded as indicators of the economy or value of any "product" or "asset" created by any team. Armors do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc’…)

Armors Labs Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Armors does not guarantee the safety or functionality of the technology agreed to be analyzed.

Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused. Armors Labs Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

Audited target file

file	md5
Hound Token.sol	5dacde6bc9148c0ad28ede60dba1a410

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe

Vulnerability	status
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe
Permission restrictions	safe

Contract file

```
/**
 *Submitted for verification at BscScan.com on 2022-03-09
*/

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

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);
}

abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes calldata) {
        this; // silence state mutability warning without generating bytecode - see https://github.co
        return msg.data;
    }
}

interface IUniswapV2Router01 {
    function factory() external pure returns (address);
    function WETH() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint amountETH);
    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountToken, uint amountETH);
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadl
    external
    payable
    returns (uint[] memory amounts);
    function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address
    external
    returns (uint[] memory amounts);
    function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address
    external
    returns (uint[] memory amounts);
    function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline
    external
    payable
    returns (uint[] memory amounts);

    function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);
    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external pure returns (uint
    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external pure returns (uint
    function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memo
    function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memo
}

interface IUniswapV2Router02 is IUniswapV2Router01 {
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountETH);
    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountETH);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external payable;
    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
}

interface IUniswapV2Factory {
    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    function feeTo() external view returns (address);
    function feeToSetter() external view returns (address);

    function getPair(address tokenA, address tokenB) external view returns (address pair);
    function allPairs(uint) external view returns (address pair);
    function allPairsLength() external view returns (uint);

    function createPair(address tokenA, address tokenB) external returns (address pair);

    function setFeeTo(address) external;
    function setFeeToSetter(address) external;
}
```

```
interface IUniswapV2Pair {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, by

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint);
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTim
    function price0CumulativeLast() external view returns (uint);
    function price1CumulativeLast() external view returns (uint);
    function kLast() external view returns (uint);


    function burn(address to) external returns (uint amount0, uint amount1);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;

    function initialize(address, address) external;
}

interface IERC20Metadata is IERC20 {
    /**
     * @dev Returns the name of the token.
     */
    function name() external view returns (string memory);

    /**
     * @dev Returns the symbol of the token.
     */
    function symbol() external view returns (string memory);

    /**
     * @dev Returns the decimals places of the token.
     */
    function decimals() external view returns (uint8);
```

```
}

contract Ownable is Context {
    address private _owner;

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

    constructor () {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    function owner() public view returns (address) {
        return _owner;
    }

    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

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) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

contract ERC20 is Context, IERC20, IERC20Metadata {
    using SafeMath for uint256;

    mapping(address => uint256) private _balances;

    mapping(address => mapping(address => uint256)) private _allowances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    /**
     * @dev Sets the values for {name} and {symbol}.
     *
     * The default value of {decimals} is 18. To select a different value for
     * {decimals} you should overload it.
     *
     * All two of these values are immutable: they can only be set once during
     * construction.
     */
    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() public view virtual override returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view virtual override returns (string memory) {
        return _symbol;
    }

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei. This is the value {ERC20} uses, unless this function is
     * overridden;
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view virtual override returns (uint8) {
        return 18;
    }

    /**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() public view virtual override returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view virtual override returns (uint256) {
        return _balances[account];
    }

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override returns (uint256)
        return _allowances[owner][spender];
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20}.
     *
     * Requirements:
     *
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC2
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(
        address sender,
        address recipient,
        uint256 amount
    ) internal virtual {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(sender, recipient, amount);

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     */
    function _cast(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");

        _beforeTokenTransfer(address(0), account, amount);

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
     *
     * This internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
```

```
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

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

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks]
     */
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
}

contract TokenDividendTracker is Ownable {
    using SafeMath for uint256;

    address[] public shareholders;
    uint256 public currentIndex;
    mapping(address => bool) private _updated;
    mapping (address => uint256) public shareholderIndexes;

    address public  uniswapV2Pair;
    address public lpRewardToken;
    // 上次分红时间
    uint256 public LPRewardLastSendTime;

    constructor(address uniswapV2Pair_, address lpRewardToken_){
        uniswapV2Pair = uniswapV2Pair_;
        lpRewardToken = lpRewardToken_;
    }

    function resetLPRewardLastSendTime() public onlyOwner {
        LPRewardLastSendTime = 0;
    }

    // LP分红发放
    function process(uint256 gas) external onlyOwner {
        uint256 shareholderCount = shareholders.length;

        if(shareholderCount == 0) return;
        uint256 nowbanance = IERC20(lpRewardToken).balanceOf(address(this));
        uint256 gasUsed = 0;
        uint256 gasLeft = gasleft();

        uint256 iterations = 0;

        while(gasUsed < gas && iterations < shareholderCount) {
            if(currentIndex >= shareholderCount){
                currentIndex = 0;
                LPRewardLastSendTime = block.timestamp;
                return;
            }

            uint256 amount = nowbanance.mul(IERC20(uniswapV2Pair).balanceOf(shareholders[currentIndex
            if( amount == 0) {
                currentIndex++;
                iterations++;
                return;
            }
            if(IERC20(lpRewardToken).balanceOf(address(this))  < amount ) return;
            IERC20(lpRewardToken).transfer(shareholders[currentIndex], amount);
            gasUsed = gasUsed.add(gasLeft.sub(gasleft()));
            gasLeft = gasleft();
            currentIndex++;
            iterations++;
        }
    }
    // 根据条件自动将交易账户加入、退出流动性分红
    function setShare(address shareholder) external onlyOwner {
        if(_updated[shareholder] ){
            if(IERC20(uniswapV2Pair).balanceOf(shareholder) == 0) quitShare(shareholder);
            return;
        }
        if(IERC20(uniswapV2Pair).balanceOf(shareholder) == 0) return;
        addShareholder(shareholder);
        _updated[shareholder] = true;

    }
    function quitShare(address shareholder) internal {
        removeShareholder(shareholder);
        _updated[shareholder] = false;
    }

    function addShareholder(address shareholder) internal {
        shareholderIndexes[shareholder] = shareholders.length;
        shareholders.push(shareholder);
    }

    function removeShareholder(address shareholder) internal {
        shareholders[shareholderIndexes[shareholder]] = shareholders[shareholders.length-1];
        shareholderIndexes[shareholders[shareholders.length-1]] = shareholderIndexes[shareholder];
        shareholders.pop();
    }

}


contract Hound is ERC20, Ownable {
    using SafeMath for uint256;

    IUniswapV2Router02 public uniswapV2Router;
    address public  uniswapV2Pair;
    bool private swapping;

    uint256 public swapTokensAtAmount;

    uint256 public deadFee = 3;
    uint256 public liquidityFee = 4;
    uint256 public marketingFee = 1;
    uint256 public foundationFee = 1;
    uint256 public lpRewardFee = 5;
    address public lpRewardToken = 0x55d398326f99059fF775485246999027B3197955;

    uint256 public AmountLiquidityFee;
    uint256 public AmountLpRewardFee;

    address public marketingWalletAddress;
    address public foundationWalletAddress;
    address public liquidityReceiveAddress;
    address public deadWallet = 0x000000000000000000000000000000000000dEaD;


    mapping (address => bool) private _isExcludedFromFees;

    TokenDividendTracker public dividendTracker;

    address private fromAddress;
    address private toAddress;
    mapping (address => bool) isDividendExempt;


    uint256 public minPeriod = 86400;

    uint256 distributorGas = 200000;

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

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


    constructor(
        string memory name_,
        string memory symbol_,
        uint256 totalSupply_,
        address marketingWalletAddr_,
        address foundationWalletAddress_,
        address liquidityReceiveAddress_
    ) payable ERC20(name_, symbol_)  {
        uint256 totalSupply = totalSupply_ * (10**18);
        swapTokensAtAmount = totalSupply.mul(2).div(10**6); // 0.002%;

        IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E2
        address _uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
        .createPair(address(this), _uniswapV2Router.WETH());

        uniswapV2Router = _uniswapV2Router;
        uniswapV2Pair = _uniswapV2Pair;

        marketingWalletAddress = marketingWalletAddr_;
        foundationWalletAddress = foundationWalletAddress_;
        liquidityReceiveAddress = liquidityReceiveAddress_;
        dividendTracker = new TokenDividendTracker(uniswapV2Pair, lpRewardToken);


        excludeFromFees(owner(), true);
        excludeFromFees(marketingWalletAddress, true);
        excludeFromFees(foundationWalletAddress, true);
        excludeFromFees(address(this), true);
        excludeFromFees(address(dividendTracker), true);

        isDividendExempt[address(this)] = true;
        isDividendExempt[address(0)] = true;
        isDividendExempt[address(dividendTracker)] = true;

        _cast(owner(), totalSupply);
    }

    receive() external payable {}


    function excludeFromFees(address account, bool excluded) public onlyOwner {
        if(_isExcludedFromFees[account] != excluded){
            _isExcludedFromFees[account] = excluded;
            emit ExcludeFromFees(account, excluded);
        }
    }


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


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


    function setFoundationWallet(address addr) public onlyOwner {
        foundationWalletAddress = addr;
    }


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


    function setSwapTokensAtAmount(uint256 amount) public onlyOwner {
        swapTokensAtAmount = amount;
    }


    function setLiquidityFee(uint256 val) public onlyOwner {
        liquidityFee = val;
    }


    function setMarketingFee(uint256 val) public onlyOwner {
        marketingFee = val;
    }


    function setFoundationFee(uint256 val) public onlyOwner {
        foundationFee = val;
    }


    function setDeadFee(uint256 val) public onlyOwner {
        deadFee = val;
    }
```

```
function setLpRewardFee(uint256 val) public onlyOwner {
    lpRewardFee = val;
}


function setMinPeriod(uint256 number) public onlyOwner {
    minPeriod = number;
}


function setLiquidityReceiveAddress(address val) public onlyOwner {
    liquidityReceiveAddress = val;
}


function resetLPRewardLastSendTime() public onlyOwner {
    dividendTracker.resetLPRewardLastSendTime();
}


function updateDistributorGas(uint256 newValue) public onlyOwner {
    require(newValue >= 100000 && newValue <= 500000, "distributorGas must be between 200,000 and
    require(newValue != distributorGas, "Cannot update distributorGas to same value");
    distributorGas = newValue;
}

function _transfer(
    address from,
    address to,
    uint256 amount
) internal override {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");
    if(amount == 0) { super._transfer(from, to, 0); return;}

    uint256 contractTokenBalance = balanceOf(address(this));
    bool canSwap = contractTokenBalance >= swapTokensAtAmount;
    if( canSwap &&
        !swapping &&
        from != uniswapV2Pair &&
        from != owner() &&
        to != owner()
    ) {
        swapping = true;
        if(AmountLiquidityFee > 0){
            swapAndLiquify(AmountLiquidityFee);
            AmountLiquidityFee = 0;
        }
        if(AmountLpRewardFee > 0){
            swapLPRewardToken(AmountLpRewardFee);
            AmountLpRewardFee = 0;
        }
        swapping = false;
    }

    bool takeFee = !swapping;
    if(_isExcludedFromFees[from] || _isExcludedFromFees[to]) {
        takeFee = false;
    }

    if(takeFee) {
        if(from != uniswapV2Pair){
            uint256 minHolderAmount = balanceOf(from).mul(90).div(100);
            if(amount > minHolderAmount){
                amount = minHolderAmount;
            }
        }
        amount =  takeAllFee(from, amount);

    }
    super._transfer(from, to, amount);

    if(fromAddress == address(0) )fromAddress = from;
    if(toAddress == address(0) )toAddress = to;
    if(!isDividendExempt[fromAddress] && fromAddress != uniswapV2Pair )   try dividendTracker.set
    if(!isDividendExempt[toAddress] && toAddress != uniswapV2Pair ) try dividendTracker.setShare(
    fromAddress = from;
    toAddress = to;

    if(  !swapping &&
    from != owner() &&
    to != owner() &&
    from !=address(this) &&
    dividendTracker.LPRewardLastSendTime().add(minPeriod) <= block.timestamp
    ) {
        try dividendTracker.process(distributorGas) {} catch {}
    }
}


function takeAllFee(address from,uint256 amount) private returns(uint256 amountAfter) {
    amountAfter = amount;

    uint256 DFee = amount.mul(deadFee).div(100);
    if(DFee > 0) super._transfer(from, deadWallet, DFee);
    amountAfter = amountAfter.sub(DFee);

    uint256 MFee = amount.mul(marketingFee).div(100);
    if(MFee > 0) super._transfer(from, marketingWalletAddress, MFee);
    amountAfter = amountAfter.sub(MFee);

    uint256 FFee = amount.mul(foundationFee).div(100);
    if(FFee > 0) super._transfer(from, foundationWalletAddress, FFee);
    amountAfter = amountAfter.sub(FFee);

    uint256 LFee = amount.mul(liquidityFee).div(100);
    AmountLiquidityFee += LFee;
    amountAfter = amountAfter.sub(LFee);

    uint256 LPFee = amount.mul(lpRewardFee).div(100);
    AmountLpRewardFee += LPFee;
    amountAfter = amountAfter.sub(LPFee);

    super._transfer(from, address(this), LFee.add(LPFee));
}




function swapAndLiquify(uint256 tokens) private {
    // split the contract balance into halves
    uint256 half = tokens.div(2);
    uint256 otherHalf = tokens.sub(half);

    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] = uniswapV2Router.WETH();

        _approve(address(this), address(uniswapV2Router), tokenAmount);

        // make the swap
        uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0, // accept any amount of ETH
            path,
            address(this),
            block.timestamp
        );

    }

    function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
        // approve token transfer to cover all possible scenarios
        _approve(address(this), address(uniswapV2Router), tokenAmount);
        // add the liquidity
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            liquidityReceiveAddress,
            block.timestamp
        );

    }

    function swapLPRewardToken(uint256 tokenAmount) private {
        address[] memory path = new address[](3);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();
        path[2] = lpRewardToken;
        _approve(address(this), address(uniswapV2Router), tokenAmount);
        uniswapV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
            tokenAmount,
            0,
            path,
            address(dividendTracker),
            block.timestamp
        );

    }

}
```

Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unexpected Blockchain Currency

- **Description:**
  Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that

msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

Entropy Illusion

- **Description:**
All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

External Contract Referencing

- **Description:**
One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Description:**
  Check for Unsolved TODO comments
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds

of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

Denial Of Service (DOS)

- **Description:**
This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

Block Timestamp Manipulation

- **Description:**
Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

Constructors with Care

- **Description:**
Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Permission restrictions

- **Description:**
  Contract managers who can control liquidity or pledge pools, etc., or impose unreasonable restrictions on other users.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

armors.io

contact@armors.io