Armors Labs

BOT (BOT) Token

Smart Contract Audit

BOT (BOT) Token Audit Summary

Project name : BOT (BOT) Token Contract

Project address: None

Code URL : https://bscscan.com/address/0x6fa690040946f49b9257b2f04f0611de55a5af9f#code

Commit : None

Project target : BOT (BOT) Token Contract Audit

Blockchain : Binance Smart Chain（BSC）

Test result : PASSED

Audit Info

Audit NO : 0X202112080006

Audit Team : Armors Labs

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

BOT (BOT) Token Audit

The BOT (BOT) Token team asked us to review and audit their BOT (BOT) 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
BOT (BOT) Token Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-12-08

Audit results

Warning:

1. The BOT (BOT) Token contract is a mixture of RFI(Reflect Finance) tokenomics with the added function of anauto-liquidity generating protocol.

2. The owner of contract has the permission to:

    i. exclude/include addresses from rewards/fees,

    ii. set taxFee, liquidityFee, _burnFee, _buybackFee, _walletFee and _maxTxAmount,

    iii. enable swapAndLiquifyEnabled.

iv. lock owner for sometime;

3. The contract is serving as the underlying entity to interact with third party

0x10ED43C718714eb63d5aA57B78B54704E256024E protocols.

Recommendation:

1. Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;

2. Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to theprivate key;

3. Introduction of a DAO / governance / voting module to increase transparency and user involvement.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the BOT (BOT) 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
bot.sol	435bbd926430ce455691dcb1a8b9736a

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0

vulnerability level	number
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
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 2021-11-17
*/

/**
 *Submitted for verification at BscScan.com on 2021-08-12
*/

// SPDX-License-Identifier: MIT

pragma solidity 0.8.6;

interface IERC20 {
```

```
    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);
}
```

```solidity
/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */

library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    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 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;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be 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;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
```

```solidity
         *
         * Requirements:
         *
         * - The divisor cannot be zero.
         */
        function mod(uint256 a, uint256 b) internal pure returns (uint256) {
            return mod(a, b, "SafeMath: modulo by zero");
        }

        /**
         * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
         * Reverts with custom message when dividing by zero.
         *
         * Counterpart to Solidity's `%` operator. This function uses a `revert`
         * opcode (which leaves remaining gas untouched) while Solidity uses an
         * invalid opcode to revert (consuming all remaining gas).
         *
         * Requirements:
         *
         * - The divisor cannot be zero.
         */
        function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
            require(b != 0, errorMessage);
            return a % b;
        }
}

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

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


/**
     * @dev Collection of functions related to the address type
     */
library Address {
        /**
         * @dev Returns true if `account` is a contract.
         *
         * [IMPORTANT]
         * ====
         * It is unsafe to assume that an address for which this function returns
         * false is an externally-owned account (EOA) and not a contract.
         *
         * Among others, `isContract` will return false for the following
         * types of addresses:
         *
         *  - an externally-owned account
         *  - a contract in construction
         *  - an address where a contract will be created
         *  - an address where a contract lived, but was destroyed
         * ====
         */
        function isContract(address account) internal view returns (bool) {
            // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
            // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
            // for accounts without code, i.e. `keccak256('')`
            bytes32 codehash;
            bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
```

```
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effe
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
        return functionCall(target, data, "Address: low-level call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
     * `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data, string memory errorMessage) internal ret
        return _functionCallWithValue(target, data, 0, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
```

```
        *
        * - the calling contract must have an ETH balance of at least `value`.
        * - the called Solidity function must be `payable`.
        *
        * _Available since v3.1._
        */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
    }

    /**
        * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithV
        * with `errorMessage` as a fallback revert reason when `target` reverts.
        *
        * _Available since v3.1._
        */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory er
        require(address(this).balance >= value, "Address: insufficient balance for call");
        return _functionCallWithValue(target, data, value, errorMessage);
    }

    function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memor
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}
/**
    * @title SafeERC20
    * @dev Wrappers around ERC20 operations that throw on failure (when the token
    * contract returns false). Tokens that return no value (and instead revert or
    * throw on failure) are also supported, non-reverting calls are assumed to be
    * successful.
    * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
    * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
    */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, valu
    }

    /**
```

```
        * @dev Deprecated. This function has issues similar to the ones found in
        * {IERC20-approve}, and its usage is discouraged.
        *
        * Whenever possible, use {safeIncreaseAllowance} and
        * {safeDecreaseAllowance} instead.
        */
    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreas
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowan
    }

    /**
        * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), rela
        * on the return value: the return value is optional (but if data is returned, it must not be
        * @param token The token targeted by the call.
        * @param data The call data (encoded using abi.encode or one of its variants).
        */
    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking m
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which
        // the target address contains contract code and also asserts for success in the low-level ca

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed
        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

/**
    * @dev Contract module which provides a basic access control mechanism, where
    * there is an account (an owner) that can be granted exclusive access to
    * specific functions.
    *
    * By default, the owner account will be the one that deploys the contract. This
    * can later be changed with {transferOwnership}.
    *
    * This module is used through inheritance. It will make available the modifier
    * `onlyOwner`, which can be applied to your functions to restrict their use to
    * the owner.
    */
abstract contract Ownable is Context {
    address private _owner;
    address private _previousOwner;
    uint256 private _lockTime;

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

    /**
```

```
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor ()  {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

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

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }

    function geUnlockTime() public view returns (uint256) {
        return _lockTime;
    }

    //Locks the contract for owner for the amount of time provided
    function lock(uint256 time) public virtual onlyOwner {
        _previousOwner = _owner;
        _owner = address(0);
        _lockTime = block.timestamp + time;
        emit OwnershipTransferred(_owner, address(0));
    }

    //Unlocks the contract for owner when _lockTime is exceeds
    function unlock() public virtual {
        require(_previousOwner == msg.sender, "You don't have permission to unlock the token contract
        require(block.timestamp > _lockTime , "Contract is locked until 7 days");
        emit OwnershipTransferred(_owner, _previousOwner);
        _owner = _previousOwner;
    }
}
```

```
// pragma solidity >=0.5.0;

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;
}




// pragma solidity >=0.6.2;

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
    }


    // pragma solidity >=0.6.2;

    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;
}


contract Token is Context, IERC20, Ownable {
    using SafeMath for uint256;
    using Address for address;
    using SafeERC20 for IERC20;

    address dead = 0x000000000000000000000000000000000000dEaD;

    uint8 public maxLiqFee = 10;
    uint8 public maxTaxFee = 10;
    uint8 public maxBurnFee = 10;
    uint8 public maxWalletFee = 10;
    uint8 public maxBuybackFee = 10;
    uint8 public minMxTxPercentage = 1;
    uint8 public minMxWalletPercentage = 1;

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

    mapping (address => bool) private _isExcludedFromFee;

    mapping (address => bool) private _isExcluded;
    address[] private _excluded;

    address public router = 0x10ED43C718714eb63d5aA57B78B54704E256024E;
    //address public router = 0xD99D1c33F9fC3444f8101754aBC46c52416550D1;

    uint256 private constant MAX = ~uint256(0);
    uint256 public _tTotal;
    uint256 private _rTotal;
    uint256 private _tFeeTotal;

    bool public mintedByMudra = true;

    string public _name;
    string public _symbol;
```

```
uint8 private _decimals;

uint8 public _taxFee = 0;
uint8 private _previousTaxFee = _taxFee;

uint8 public _liquidityFee = 0;
uint8 private _previousLiquidityFee = _liquidityFee;

uint8 public _burnFee = 0;
uint8 private _previousBurnFee = _burnFee;

uint8 public _walletFee = 0;
uint8 private _previousWalletFee = _walletFee;

uint8 public _buybackFee = 0;
uint8 private _previousBuybackFee = _buybackFee;

IUniswapV2Router02 public immutable pcsV2Router;
address public immutable pcsV2Pair;
address payable public feeWallet;

bool inSwapAndLiquify;
bool public swapAndLiquifyEnabled = true;

uint256 public _maxTxAmount;
uint256 public _maxWalletAmount;
uint256 public numTokensSellToAddToLiquidity;
uint256 private buyBackUpperLimit = 1 * 10**18;

event SwapAndLiquifyEnabledUpdated(bool enabled);
event SwapAndLiquify(
    uint256 tokensSwapped,
    uint256 ethReceived,
    uint256 tokensIntoLiqudity
);

modifier lockTheSwap {
    inSwapAndLiquify = true;
    _;
    inSwapAndLiquify = false;
}

constructor (address tokenOwner,string memory tokenName,
    string memory tokenSymbol, uint8 decimal, uint256 amountOfTokenWei,
    uint8 setMxTxPer, uint8 setMxWalletPer,
    address payable _feeWallet
    ) {

    _name = tokenName;
    _symbol = tokenSymbol;
    _decimals = decimal;
    _tTotal = amountOfTokenWei;
    _rTotal = (MAX - (MAX % _tTotal));

    _rOwned[tokenOwner] = _rTotal;

    feeWallet = _feeWallet;


    _maxTxAmount = _tTotal.mul(setMxTxPer).div(
        10**2
    );
    _maxWalletAmount = _tTotal.mul(setMxWalletPer).div(
        10**2
    );
```

```solidity
        numTokensSellToAddToLiquidity = amountOfTokenWei.mul(1).div(1000);

        IUniswapV2Router02 _pcsV2Router = IUniswapV2Router02(router);
            // Create a uniswap pair for this new token
        pcsV2Pair = IUniswapV2Factory(_pcsV2Router.factory())
            .createPair(address(this), _pcsV2Router.WETH());

        // set the rest of the contract variables
        pcsV2Router = _pcsV2Router;

        _isExcludedFromFee[tokenOwner] = true;
        _isExcludedFromFee[address(this)] = true;

        emit Transfer(address(0), tokenOwner, _tTotal);
    }

    function name() public view returns (string memory) {
        return _name;
    }

    function symbol() public view returns (string memory) {
        return _symbol;
    }

    function decimals() public view returns (uint8) {
        return _decimals;
    }

    function totalSupply() public view override returns (uint256) {
        return _tTotal;
    }

    function balanceOf(address account) public view override returns (uint256) {
        if (_isExcluded[account]) return _tOwned[account];
        return tokenFromReflection(_rOwned[account]);
    }

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

    function allowance(address owner, address spender) public view override returns (uint256) {
        return _allowances[owner][spender];
    }

    function approve(address spender, uint256 amount) public override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) public override returns
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
        return true;
    }

    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC2
        return true;
    }
```

```solidity
function isExcludedFromReward(address account) public view returns (bool) {
    return _isExcluded[account];
}

function totalFees() public view returns (uint256) {
    return _tFeeTotal;
}

function deliver(uint256 tAmount) public {
    address sender = _msgSender();
    require(!_isExcluded[sender], "Excluded addresses cannot call this function");
    (uint256 rAmount,,,,,) = _getValues(tAmount);
    _rOwned[sender] = _rOwned[sender].sub(rAmount);
    _rTotal = _rTotal.sub(rAmount);
    _tFeeTotal = _tFeeTotal.add(tAmount);
}

function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public view returns(uint256
    require(tAmount <= _tTotal, "Amt must be less than supply");
    if (!deductTransferFee) {
        (uint256 rAmount,,,,,) = _getValues(tAmount);
        return rAmount;
    } else {
        (,uint256 rTransferAmount,,,,) = _getValues(tAmount);
        return rTransferAmount;
    }
}

function tokenFromReflection(uint256 rAmount) public view returns(uint256) {
    require(rAmount <= _rTotal, "Amt must be less than tot refl");
    uint256 currentRate =  _getRate();
    return rAmount.div(currentRate);
}

function excludeFromReward(address account) public onlyOwner() {
    require(!_isExcluded[account], "Account is already excluded from reward");
    if(_rOwned[account] > 0) {
        _tOwned[account] = tokenFromReflection(_rOwned[account]);
    }
    _isExcluded[account] = true;
    _excluded.push(account);
}

function includeInReward(address account) external onlyOwner() {
    require(_isExcluded[account], "Already excluded");
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}


function excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}

function includeInFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = false;
}
```

```
function setAllFeePercent(uint8 taxFee, uint8 liquidityFee, uint8 burnFee, uint8 walletFee, uint8
    require(taxFee >= 0 && taxFee <=maxTaxFee,"TF err");
    require(liquidityFee >= 0 && liquidityFee <=maxLiqFee,"LF err");
    require(burnFee >= 0 && burnFee <=maxBurnFee,"BF err");
    require(walletFee >= 0 && walletFee <=maxWalletFee,"WF err");
    require(buybackFee >= 0 && buybackFee <=maxBuybackFee,"BBF err");
    _taxFee = taxFee;
    _liquidityFee = liquidityFee;
    _burnFee = burnFee;
    _buybackFee = buybackFee;
    _walletFee = walletFee;
}


function buyBackUpperLimitAmount() public view returns (uint256) {
    return buyBackUpperLimit;
}


function setBuybackUpperLimit(uint256 buyBackLimit) external onlyOwner() {
    buyBackUpperLimit = buyBackLimit * 10**18;
}


function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
    require(maxTxPercent >= minMxTxPercentage && maxTxPercent <=100,"err");
    _maxTxAmount = _tTotal.mul(maxTxPercent).div(
        10**2
    );
}


function setMaxWalletPercent(uint256 maxWalletPercent) external onlyOwner() {
    require(maxWalletPercent >= minMxWalletPercentage && maxWalletPercent <=100,"err");
    _maxWalletAmount = _tTotal.mul(maxWalletPercent).div(
        10**2
    );
}


function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {
    swapAndLiquifyEnabled = _enabled;
    emit SwapAndLiquifyEnabledUpdated(_enabled);
}


function setFeeWallet(address payable newFeeWallet) external onlyOwner {
    require(newFeeWallet != address(0), "ZERO ADDRESS");
    feeWallet = newFeeWallet;
}


    //to recieve ETH from pcsV2Router when swaping
receive() external payable {}


function _reflectFee(uint256 rFee, uint256 tFee) private {
    _rTotal = _rTotal.sub(rFee);
    _tFeeTotal = _tFeeTotal.add(tFee);
}


function _getValues(uint256 tAmount) private view returns (uint256, uint256, uint256, uint256, ui
    (uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getTValues(tAmount);
    (uint256 rAmount, uint256 rTransferAmount, uint256 rFee) = _getRValues(tAmount, tFee, tLiquid
    return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee, tLiquidity);
}


function _getTValues(uint256 tAmount) private view returns (uint256, uint256, uint256) {
    uint256 tFee = calculateTaxFee(tAmount);
    uint256 tLiquidity = calculateLiquidityFee(tAmount);
    uint256 tTransferAmount = tAmount.sub(tFee).sub(tLiquidity);
    return (tTransferAmount, tFee, tLiquidity);
}
```

```
function _getRValues(uint256 tAmount, uint256 tFee, uint256 tLiquidity, uint256 currentRate) priv
    uint256 rAmount = tAmount.mul(currentRate);
    uint256 rFee = tFee.mul(currentRate);
    uint256 rLiquidity = tLiquidity.mul(currentRate);
    uint256 rTransferAmount = rAmount.sub(rFee).sub(rLiquidity);
    return (rAmount, rTransferAmount, rFee);
}

function _getRate() private view returns(uint256) {
    (uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
    return rSupply.div(tSupply);
}

function _getCurrentSupply() private view returns(uint256, uint256) {
    uint256 rSupply = _rTotal;
    uint256 tSupply = _tTotal;
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return (_rTotal, _
        rSupply = rSupply.sub(_rOwned[_excluded[i]]);
        tSupply = tSupply.sub(_tOwned[_excluded[i]]);
    }
    if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
    return (rSupply, tSupply);
}

function _takeLiquidity(uint256 tLiquidity) private {
    uint256 currentRate = _getRate();
    uint256 rLiquidity = tLiquidity.mul(currentRate);
    _rOwned[address(this)] = _rOwned[address(this)].add(rLiquidity);
    if(_isExcluded[address(this)])
        _tOwned[address(this)] = _tOwned[address(this)].add(tLiquidity);
}

function calculateTaxFee(uint256 _amount) private view returns (uint256) {
    return _amount.mul(_taxFee).div(
        10**2
    );
}

function calculateLiquidityFee(uint256 _amount) private view returns (uint256) {
    return _amount.mul(_liquidityFee + _burnFee + _walletFee + _buybackFee).div(
        10**2
    );
}

function removeAllFee() private {
    if(_taxFee == 0 && _liquidityFee == 0 && _burnFee == 0 && _walletFee == 0 && _buybackFee == 0

    _previousTaxFee = _taxFee;
    _previousLiquidityFee = _liquidityFee;
    _previousBurnFee = _burnFee;
    _previousWalletFee = _walletFee;
    _previousBuybackFee = _buybackFee;

    _taxFee = 0;
    _liquidityFee = 0;
    _burnFee = 0;
    _walletFee = 0;
    _buybackFee = 0;
}

function restoreAllFee() private {
    _taxFee = _previousTaxFee;
    _liquidityFee = _previousLiquidityFee;
    _burnFee = _previousBurnFee;
```

```
                _walletFee = _previousWalletFee;
                _buybackFee = _previousBuybackFee;
        }

        function isExcludedFromFee(address account) public view returns(bool) {
                return _isExcludedFromFee[account];
        }

        function _approve(address owner, address spender, uint256 amount) private {
                require(owner != address(0), "ERC20: approve from zero address");
                require(spender != address(0), "ERC20: approve to zero address");

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

        function _transfer(
                address from,
                address to,
                uint256 amount
        ) private {
                require(from != address(0), "ERC20: transfer from zero address");
                require(to != address(0), "ERC20: transfer to zero address");
                require(amount > 0, "Transfer amount must be greater than zero");
                if(from != owner() && to != owner())
                        require(amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount.");

                if(from != owner() && to != owner() && to != address(0) && to != dead && to != pcsV2Pair){
                        uint256 contractBalanceRecepient = balanceOf(to);
                        require(contractBalanceRecepient + amount <= _maxWalletAmount, "Exceeds maximum wallet am
                }
                // is the token balance of this contract address over the min number of
                // tokens that we need to initiate a swap + liquidity lock?
                // also, don't get caught in a circular liquidity event.
                // also, don't swap & liquify if sender is uniswap pair.
                uint256 contractTokenBalance = balanceOf(address(this));

                if(contractTokenBalance >= _maxTxAmount)
                {
                        contractTokenBalance = _maxTxAmount;
                }

                bool overMinTokenBalance = contractTokenBalance >= numTokensSellToAddToLiquidity;
                if (
                        !inSwapAndLiquify &&
                        to == pcsV2Pair &&
                        swapAndLiquifyEnabled
                ) {
                        if(overMinTokenBalance){
                                contractTokenBalance = numTokensSellToAddToLiquidity;
                                //add liquidity
                                swapAndLiquify(contractTokenBalance);
                        }
                        if(_buybackFee !=0){
                                uint256 balance = address(this).balance;
                                if (balance > uint256(1 * 10**18)) {

                                        if (balance > buyBackUpperLimit)
                                                balance = buyBackUpperLimit;

                                        buyBackTokens(balance.div(100));
                                }
                        }

                }
```

```
        //indicates if fee should be deducted from transfer
        bool takeFee = true;

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

        //transfer amount, it will take tax, burn, liquidity fee
        _tokenTransfer(from,to,amount,takeFee);
    }

    function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
        //This needs to be distributed among burn, wallet and liquidity
        //burn
        uint8 totFee  = _burnFee + _walletFee + _liquidityFee + _buybackFee;
        uint256 spentAmount = 0;
        uint256 totSpentAmount = 0;
        if(_burnFee != 0){
            spentAmount  = contractTokenBalance.div(totFee).mul(_burnFee);
            _tokenTransferNoFee(address(this), dead, spentAmount);
            totSpentAmount = spentAmount;
        }

        if(_walletFee != 0){
            spentAmount = contractTokenBalance.div(totFee).mul(_walletFee);
            _tokenTransferNoFee(address(this), feeWallet, spentAmount);
            totSpentAmount = totSpentAmount + spentAmount;
        }

        if(_buybackFee != 0){
            spentAmount = contractTokenBalance.div(totFee).mul(_buybackFee);
            swapTokensForBNB(spentAmount);
            totSpentAmount = totSpentAmount + spentAmount;
        }

        if(_liquidityFee != 0){
            contractTokenBalance = contractTokenBalance.sub(totSpentAmount);

            // split the contract balance into halves
            uint256 half = contractTokenBalance.div(2);
            uint256 otherHalf = contractTokenBalance.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
            swapTokensForBNB(half); // <- this breaks the ETH -> HATE swap when swap+liquify is trigg

            // 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 buyBackTokens(uint256 amount) private lockTheSwap {
        if (amount > 0) {
```

```
                swapBNBForTokens(amount);
            }
        }

        function swapTokensForBNB(uint256 tokenAmount) private {
            // generate the uniswap pair path of token -> weth
            address[] memory path = new address[](2);
            path[0] = address(this);
            path[1] = pcsV2Router.WETH();

            _approve(address(this), address(pcsV2Router), tokenAmount);

            // make the swap
            pcsV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
                tokenAmount,
                0, // accept any amount of ETH
                path,
                address(this),
                block.timestamp
            );
        }

        function swapBNBForTokens(uint256 amount) private {
            // generate the uniswap pair path of token -> weth
            address[] memory path = new address[](2);
            path[0] = pcsV2Router.WETH();
            path[1] = address(this);

          // make the swap
            pcsV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(
                0, // accept any amount of Tokens
                path,
                dead, // Burn address
                block.timestamp.add(300)
            );
        }

        function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
            // approve token transfer to cover all possible scenarios
            _approve(address(this), address(pcsV2Router), tokenAmount);

            // add the liquidity
            pcsV2Router.addLiquidityETH{value: ethAmount}(
                address(this),
                tokenAmount,
                0, // slippage is unavoidable
                0, // slippage is unavoidable
                dead,
                block.timestamp
            );
        }

    //this method is responsible for taking all fee, if takeFee is true
    function _tokenTransfer(address sender, address recipient, uint256 amount,bool takeFee) private {
        if(!takeFee)
            removeAllFee();

        if (_isExcluded[sender] && !_isExcluded[recipient]) {
            _transferFromExcluded(sender, recipient, amount);
        } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
            _transferToExcluded(sender, recipient, amount);
        } else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
            _transferStandard(sender, recipient, amount);
        } else if (_isExcluded[sender] && _isExcluded[recipient]) {
            _transferBothExcluded(sender, recipient, amount);
        } else {
```

```
            _transferStandard(sender, recipient, amount);
        }

        if(!takeFee)
            restoreAllFee();
    }

    function _transferStandard(address sender, address recipient, uint256 tAmount) private {
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFe
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    function _transferToExcluded(address sender, address recipient, uint256 tAmount) private {
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFe
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    function _transferFromExcluded(address sender, address recipient, uint256 tAmount) private {
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFe
        _tOwned[sender] = _tOwned[sender].sub(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    function _transferBothExcluded(address sender, address recipient, uint256 tAmount) private {
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFe
        _tOwned[sender] = _tOwned[sender].sub(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    function _tokenTransferNoFee(address sender, address recipient, uint256 amount) private {
        uint256 currentRate = _getRate();
        uint256 rAmount = amount.mul(currentRate);

        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rAmount);

        if (_isExcluded[sender]) {
            _tOwned[sender] = _tOwned[sender].sub(amount);
        }
        if (_isExcluded[recipient]) {
            _tOwned[recipient] = _tOwned[recipient].add(amount);
        }
        emit Transfer(sender, recipient, amount);
    }

    function recoverBEP20(address tokenAddress, uint256 tokenAmount) public onlyOwner {
        // do not allow recovering self token
        require(tokenAddress != address(this), "Self withdraw");
```

```
            IERC20(tokenAddress).transfer(owner(), tokenAmount);
    }
}
```

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.

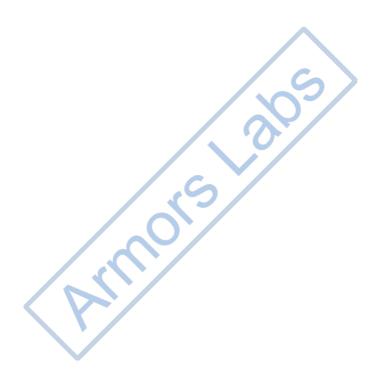## Permission restrictions

Armors Labs

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

  PASSED !   30 / 30

- **Security suggestion:**
  no.

armors.io

contact@armors.io