Armors Labs

BIBToken (BIB) Token

Smart Contract Audit

BIBToken (BIB) Token Audit Summary

Project name : BIBToken (BIB) Token Contract

Project address: None

Code URL : https://github.com/bibvip-github/bibcoin

Commit : 28c1a71fe7ae7c863213bdb82712251458715dc3

Project target : BIBToken (BIB) Token Contract Audit

Blockchain : Binance Smart Chain（BSC）

Test result : PASSED

Audit Info

Audit NO : 0X202208250006

Audit Team : Armors Labs

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

BIBToken (BIB) Token Audit

The BIBToken (BIB) Token team asked us to review and audit their BIBToken (BIB) 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
BIBToken (BIB) Token Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.1	2022-08-25

Audit results

Note that:

1. The BIBToken (BIB) Token is not only ERC20 token
    i. a handling fee is charged for each transaction and service fee can be adjusted
    ii. each Transfer /DEX transaction of BIB tokens will trigger a dividend distribution
    iii. frozen amounts that dependent on contracts do not allow users to transfer
    iv. the contract can be upgraded and the owner can change the dependent contract (bonus contract, freeze contract), there are potential risks
2. The owner of contract has the permission to:
    i. set reward fee、blackhole fee、liquidity fee and fee white list
    ii. enable pause, swapEnabled

3. The contract is serving as the underlying entity to interact with third party 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 BIBToken (BIB) 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
./token/BEP20Token.sol	55cc5668f939d8e17ac70561f93e3ec7
./token/BIBToken.sol	e950bd630b3ca7c857f2c69fedd66e1d
./token/FreezeToken.sol	a5f17d2000c24f3ee5d500f3be2195f7
./token/DividendPayingToken.sol	f969666dd37ab850a116d9c6edac868f
./token/TokenDividendTracker.sol	38f122fd73ae37345899f2e2a43e8e5c
./interface/ITokenDividendTracker.sol	c86acece58fc0a2c4af5bed0850012fc
./interface/FreezeTokenInterface.sol	7daece18721bb913e3596500e8367101
./interface/DividendPayingTokenInterface.sol	935b8c4b86b1c0a8b70601e7ffc1ff12
./interface/Context.sol	10a314f62166999aa0f31627bea77c32
./interface/DividendPayingTokenOptionalInterface.sol	8410d92a24b9de36c5ad1a36dc1d6d36
./interface/IERC20.sol	bce92f83632d85f89fc3f20f504f554a

file	md5
./interface/UniswapInterface.sol	92859dfa5f566929f2a72b84c264eb96
./interface/IERC20Metadata.sol	62cbb25d16f4ef208f75224d538bb5c4
./library/IterableMapping.sol	a77c6a9041436c926c0f7394ad9039a6
./library/SafeMath.sol	87dda8ef1fc753b3cc4d7262935463d1
./library/SafeMathUint.sol	e4c9a12b3a0ea918b914ac31acd67c3b
./library/SafeMathInt.sol	3c78bbdced0db0932d861463843e3e3a
./vest/VestingToken.sol	5f08f9460316e29f17742af432eba57e

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
Block Timestamp Manipulation	safe
Constructors with Care	safe

Vulnerability	status
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe
Permission restrictions	safe

Contract file

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "../interface/ITokenDividendTracker.sol";
import "../interface/FreezeTokenInterface.sol";
import "@openzeppelin/contracts/utils/math/SafeMath.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import '@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol';
import '@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol';
import "@uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
import "@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol";
import "@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";

contract BIBToken is Initializable, ERC20Upgradeable, PausableUpgradeable, OwnableUpgradeable {
    using SafeMath for uint256;
    mapping(address => bool) public admin;

    IUniswapV2Router02 public uniswapV2Router;
    FreezeTokenInterface public freezeToken;
    address public uniswapV2Pair;
    address public constant deadAddress = 0x000000000000000000000000000000000000dEaD;
    address public rewardToken; //rewardToken
    address public router;

    uint public rewardFee = 6;
    uint public blackholeFee = 1;
    uint public liquidityFee = 3;

    bool private swapping;

    ITokenDividendTracker public dividendTracker;
    address public w0;
    address public tokenOwner;
    address public w1;
    address public w2;
    address public w3;
    address public w4;
    address public w5;
    address public w6;
    address public w7;
    uint public decimalVal = 1e18;

    uint256 public maxSellTransactionAmount = 10_000_000_000_000 * decimalVal;
    uint256 public swapTokensAtAmount = 1000_000_000 * decimalVal;

    bool public swapEnabled;

    uint256 initialSupply = 100_000_000_000 * decimalVal;
    // use by default 300,000 gas to process auto-claiming dividends
    uint256 public gasForProcessing = 300000;
    bool public allowTransfer;

    mapping(address => bool) public isFromWhiteList;
    mapping(address => bool) public isToWhiteList;
    mapping(address => bool) public noProcessList;
    address private canStopAntibotMeasures;
    uint256 public antibotEndTime;

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

    event UpdateUniswapV2Router(address indexed newAddress, address indexed oldAddress);

    event ExcludeFromFees(address indexed account, bool isExcluded, bool isFrom);

    event NoProcessList(address indexed account, bool noProcess);

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

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

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

    event SendDividends(
        uint256 tokensSwapped,
        uint256 amount
    );

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

    function initialize(
        address _dividendTracker,
        address _router,
        address _rewardToken
        ) initializer public {
        tokenOwner = msg.sender;
        __ERC20_init("BIBToken", "BIB");
        _mint(tokenOwner, initialSupply);
        __Pausable_init();
        __Ownable_init();

        require(address(0) != _rewardToken, "INVLID_REWARD_TOKEN");
        require(address(0) != _dividendTracker, "INVLID_DIVIDENTTRACKER");
        require(address(0) != _router, "INVLID_ROUTER");

        rewardToken = _rewardToken;
        dividendTracker = ITokenDividendTracker(_dividendTracker);
        require(dividendTracker.controller() == address(this), "Token: The new dividend tracker must

        router = _router;
        IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(router);
        address _uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
            .createPair(address(this), _uniswapV2Router.WETH());
        uniswapV2Router = _uniswapV2Router;
        uniswapV2Pair = _uniswapV2Pair;
        _setAutomatedMarketMakerPair(_uniswapV2Pair, true);

        // exclude from receiving dividends
        dividendTracker.excludeFromDividends(address(dividendTracker));
        dividendTracker.excludeFromDividends(address(this));
        dividendTracker.excludeFromDividends(owner());
        dividendTracker.excludeFromDividends(deadAddress);

        // exclude from paying fees or having max transaction amount
        setFeeWhiteList(owner(), true, true);
        setFeeWhiteList(address(this), true, true);
        setFeeWhiteList(_uniswapV2Pair, true, true);
        setFeeWhiteList(deadAddress, true, false);

        swapEnabled = true;
    }

    function pause() public onlyOwner {
        _pause();
    }

    function unpause() public onlyOwner {
        _unpause();
    }

    function initAddress(
        address _w1,
        address _w2,
        address _w3,
        address _w4,
        address _w0,
        address _w5,
        address _w6,
        address _w7) public onlyOwner {
        require(_w1 != address(0), "_w1 is not the zero address");
        require(_w2 != address(0), "_w2 is not the zero address");
        require(_w3 != address(0), "_w2 is not the zero address");
        require(_w4 != address(0), "_w4 is not the zero address");
        require(_w0 != address(0), "_w0 is not the zero address");
        require(_w5 != address(0), "_w5 is not the zero address");
        require(_w6 != address(0), "_w6 is not the zero address");
        require(_w7 != address(0), "_w7 is not the zero address");

        w1 = _w1;
        w2 = _w2;
        w3 = _w3;
        w4 = _w4;
        w0 = _w0;
        w5 = _w5;
        w6 = _w6;
        w7 = _w7;
        dividendTracker.excludeFromDividends(address(w1));
        dividendTracker.excludeFromDividends(address(w2));
        dividendTracker.excludeFromDividends(address(w3));
        dividendTracker.excludeFromDividends(address(w4));
        dividendTracker.excludeFromDividends(address(w0));
        dividendTracker.excludeFromDividends(address(w5));
        dividendTracker.excludeFromDividends(address(w6));
        dividendTracker.excludeFromDividends(address(w7));
    }
    receive() external payable {}

    function release() public onlyOwner {
        transfer(w1,(initialSupply.mul(15).div(100)));
        transfer(w2,(initialSupply.mul(25).div(100)));
        transfer(w3,(initialSupply.mul(15).div(100)));
        transfer(w4,(initialSupply.mul(2).div(100)));
        transfer(w0,(initialSupply.mul(3).div(100)));
        transfer(w5,(initialSupply.mul(9).div(100)));
        transfer(w6,(initialSupply.mul(16).div(100)));
        transfer(w7,(initialSupply.mul(15).div(100)));
    }

    function addExcludeFromDividends(address[] memory addrs) public onlyOwner {
        for (uint256 i = 0; i < addrs.length; i++) {
            dividendTracker.excludeFromDividends(addrs[i]);
        }
    }

    function updateUniswapV2Router(address newAddress) public onlyOwner {
        require(!isContract(newAddress), "newAddress is contract address");
        require(newAddress != address(uniswapV2Router), "Token: The router already has that address")
        emit UpdateUniswapV2Router(newAddress, address(uniswapV2Router));
        uniswapV2Router = IUniswapV2Router02(newAddress);
        uniswapV2Pair = IUniswapV2Factory(uniswapV2Router.factory())
            .createPair(address(this), uniswapV2Router.WETH());
    }

    function isContract(address addr) public view returns (bool) {
        uint size;
        assembly { size := extcodesize(addr) }
        return size > 0;
    }

    function setFeeWhiteList(address account, bool excluded, bool isFrom) public onlyOwner {
        if (isFrom) {
            require(isFromWhiteList[account] != excluded, "Token: Account is already the value of 'ex
            isFromWhiteList[account] = excluded;
        } else {
            require(isToWhiteList[account] != excluded, "Token: Account is already the value of 'excl
            isToWhiteList[account] = excluded;
        }
        emit ExcludeFromFees(account, excluded, isFrom);
    }

    function setMultipleWhiteList(address[] calldata accounts, bool excluded, bool isFrom) public onl
        for(uint256 i = 0; i < accounts.length; i++) {
            setFeeWhiteList(accounts[i], excluded, isFrom);
        }
    }

    function setNoProcessList(address account, bool noProcess) public onlyOwner {
        require(noProcessList[account] != noProcess, "Token: Account is already the value of 'noProce
        noProcessList[account] = noProcess;
        emit NoProcessList(account, noProcess);
    }

    function setMultipleNoProcessList(address[] calldata accounts, bool noProcess) public onlyOwner {
        for(uint256 i = 0; i < accounts.length; i++) {
            setNoProcessList(accounts[i], noProcess);
        }
    }

    function setAutomatedMarketMakerPair(address pair, bool value) public onlyOwner {
        require(pair != uniswapV2Pair, "Token: The PancakeSwap pair cannot be removed from automatedM
        _setAutomatedMarketMakerPair(pair, value);
    }

    function _setAutomatedMarketMakerPair(address pair, bool value) private {
        require(automatedMarketMakerPairs[pair] != value, "Token: Automated market maker pair is alre
        automatedMarketMakerPairs[pair] = value;

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

        emit SetAutomatedMarketMakerPair(pair, value);
    }

    function updateGasForProcessing(uint256 newValue) public onlyOwner {
        require(newValue != gasForProcessing, "Token: Cannot update gasForProcessing to same value");
        emit GasForProcessingUpdated(newValue, gasForProcessing);
        gasForProcessing = newValue;
    }

    function updateSwapTokensAtAmount(uint256 newValue) public onlyOwner {
        require(newValue != swapTokensAtAmount, "Token: Cannot update swapTokensAtAmount to same valu
        emit SwapTokensAtAmountUpdated(newValue, swapTokensAtAmount);
        swapTokensAtAmount = newValue;
    }

    function setSellFee(uint16 _rewardfee, uint16 _blackhole, uint16 _liquidity) external onlyOwner {
        rewardFee = _rewardfee;
        blackholeFee = _blackhole;
        liquidityFee = _liquidity;
        require(rewardFee + blackholeFee + liquidityFee <= 10, "INVALID_FEE_RATIO");
    }

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

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

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

    function processDividendTracker(uint256 gas) external onlyOwner {
    (uint256 iterations, uint256 claims, uint256 lastProcessedIndex) = dividendTracker.process(gas);
        emit ProcessedDividendTracker(iterations, claims, lastProcessedIndex, false, gas, tx.origin);
    }
    function claim() external {
        dividendTracker.processAccount(payable(msg.sender), false);
    }

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

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

    function setW0(address w) external onlyOwner{
        require(w != address(0), "w is not the zero address");
        w0 = payable(w);
    }

    function setW1Address(address w) public onlyOwner {
        require(w != address(0), "w is not the zero address");
        w1 = w;
    }

    function setW2Address(address w) public onlyOwner {
        require(w != address(0), "w is not the zero address");
        w2 = w;
    }

    function setW3Address(address w) public onlyOwner {
        require(w != address(0), "w is not the zero address");
        w3 = w;
    }

    function setW4Address(address w) public onlyOwner {
        require(w != address(0), "w is not the zero address");
        w4 = w;
    }

    function setW5Address(address w) public onlyOwner {
        require(w != address(0), "w is not the zero address");
        w5 = w;
    }

    function setW6Address(address w) public onlyOwner {
        require(w != address(0), "w is not the zero address");
        w6 = w;
    }

    function setW7Address(address w) public onlyOwner {
        require(w != address(0), "w is not the zero address");
        w7 = w;
    }

    function setFreezeTokenAddress(address _freezeToken) public onlyOwner {
        require(_freezeToken != address(0), "_freezeToken is not the zero address");
        freezeToken = FreezeTokenInterface(_freezeToken);
    }

    function setSwapEnabled(bool value) external onlyOwner{
        swapEnabled = value;
    }

    function setAllowTransfer(bool value) external onlyOwner{
        allowTransfer = value;
    }
    function _checkFreezeAmount(address account, uint256 transferAmount) internal view returns(bool)
        if (address(freezeToken) == address(0)) {
            return true;
        }
        uint256 freezeAmount = freezeToken.getFreezeAmount(account);
        return balanceOf(account) - freezeAmount >= transferAmount;
    }

    function _transfer(
    address from,
    address to,
    uint256 amount
    ) internal override {
        require(allowTransfer, "ERC20: unable to transfer");
        require(from != address(0), "ERC20: transfer from the zero address");
        require(to != address(0), "ERC20: transfer to the zero address");
        require (_checkFreezeAmount(from, amount), "Not enough available balance");

        // forbiden bot
        if (from != owner() && to != owner() && (block.timestamp <= antibotEndTime || antibotEndTime
            require (to == canStopAntibotMeasures, "Timerr: Bots can't stop antibot measures");
            if (antibotEndTime == 0)
                antibotEndTime = block.timestamp + 3;
        }

        if(amount == 0) {
            super._transfer(from, to, 0);
            return;
        }

        // Check max wallet
        if (from != owner() && to != uniswapV2Pair){
            require (balanceOf(to) + amount <= maxSellTransactionAmount, " Receiver's wallet balance
        }

        // check if have sufficient balance
        uint256 contractTokenBalance = balanceOf(address(this));
        bool canSwap = contractTokenBalance >= swapTokensAtAmount;
        if(swapEnabled && !swapping && !noProcessList[to] && canSwap) {
            swapping = true;
            contractTokenBalance = swapTokensAtAmount;

            // part goes to liquadation pool
            uint256 swapTokens = contractTokenBalance.mul(
                liquidityFee).div(liquidityFee.add(rewardFee));
            swapAndLiquify(swapTokens);

            // part gose to dividens pool
            swapAndSendDividends(contractTokenBalance.sub(swapTokens));

            swapping = false;
        }

        bool takeFee = !swapping;

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

        if(takeFee) {
            uint256 fees = amount.mul(liquidityFee.add(rewardFee)).div(100);
            uint256 burned = amount.mul(blackholeFee).div(100);
            amount = amount.sub(fees).sub(burned);
            super._transfer(from, deadAddress, burned);
            super._transfer(from, address(this), fees);
        }

        super._transfer(from, to, amount);

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

        if(!swapping && !noProcessList[to]) {
            uint256 gas = gasForProcessing;

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

    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
            deadAddress,
            block.timestamp
        );
    }

    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;//address(this)??

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

        // how much rewardToken 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 swapTokensForBusd(uint256 tokenAmount) private {
        address[] memory path = new address[](3);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();
        path[2] = rewardToken;

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

        // make the swap
        uniswapV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
            tokenAmount,
            0,
            path,
            address(this),
            block.timestamp
        );
    }

    function setAntiBotStopAddress (address account) external onlyOwner {
        require (account != address(0));
        canStopAntibotMeasures = account;
    }

    function _beforeTokenTransfer(address from, address to, uint256 amount) internal whenNotPaused ov
        super._beforeTokenTransfer(from, to, amount);
    }

    function swapAndSendDividends(uint256 tokens) private{
        swapTokensForBusd(tokens);
        uint256 dividends = IERC20(rewardToken).balanceOf(address(this));
        bool success = IERC20(rewardToken).transfer(address(dividendTracker), dividends);

        if (success) {
            dividendTracker.distributeDividends(dividends);
            emit SendDividends(tokens, dividends);
        }
    }
}

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "@openzeppelin/contracts/finance/VestingWallet.sol";

contract VestingToken is VestingWallet{
    constructor(
        address beneficiaryAddress,
        uint64 startTimestamp,
        uint64 durationSeconds
    ) VestingWallet(beneficiaryAddress, startTimestamp, durationSeconds){}

    receive() external payable override {
        revert("UNSUPPORTED_OP");
    }

    function release() public override {
        revert("UNSUPPORTED_OP");
    }
}
```

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