Deliverable: Smart Contract Audit Report

The Hodlers Are Millionaires

Smart Contract Review

Security Report

June 2021

Disclaimer

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the Company. The content, conclusions and recommendations set out inthis publication are elaborated in the specific for only project.

The eNebula Solutions does not guarantee the accuracy of the data included in this study. All representations, warranties, undertakings and guarantees relating to the report are excluded, particularly concerning – but not limited to – the qualities of the assessed projects and products. Neither the Company nor any personacting on the Company's behalf may be held responsible for the use that may be made of the information contained herein.

eNebula Solutions retains the right to display audit reports and other content elements as examples of their work in their portfolio and as content features in other projects with protecting all security purpose of customer. The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities fixed - upon a decision of the Customer.

Report Summary

Title	**The Hodlers Are Millionaires Smart Contract Audit**		
Project Owner	The Hodlers Are Millionaires		
Type	Public		
Reviewed by	Vatsal Raychura	**Revision date**	02/06/2021
Approved by	eNebula Solutions Private Limited	**Approval date**	02/06/2021
		Nº Pages	**43**

Overview

Background

The Hodlers Are Millionaires requested that eNebula Solutions perform an Extensive Smart Contract audit of their 'MoonSwap' Smart Contract.

Project Dates

The following is the project schedule for this review and report:

- **June 2**: Smart Contract Review Completed *(Completed)*
- **June 2**: Delivery of Smart Contract Audit Report *(Completed)*

Review Team

The following eNebula Solutions team member participated in this review:

- Sejal Barad, Security Researcher and Engineer
- Vatsal Raychura, Security Researcher and Engineer

Coverage

Target Specification and Revision

For this audit, we performed research, investigation, and review of the smart contract of The Hodlers Are Millionaires.

The following documentation repositories were considered in-scope for the review:

- The Hodlers Are Millionaires Project:

https://testnet.bscscan.com/address/0xC288ec99a7fBC7bD7799D5d54f8F10F9bd4d2931#code

Introduction

Given the opportunity to review the Hodlers Are Millionaires Contracts related smart contract source code, we in the report outline our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts is ready to launch after resolving the **1 high, 8 medium and 17 low severity issues**, there are no critical issues found related to business logic, security or performance.

About The Hodlers Are Millionaires: -

Item	Description
Issuer	The Hodlers Are Millionaires
Website	thehodlersaremillionaires.com
Type	ERC20
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	June 2, 2021

The Full List of Check Items:

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization

Advanced DeFi Scrutiny	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

Common Weakness Enumeration (CWE) Classifications Used in This Audit:

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not securitysoftware.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values,Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code,or if the application does not handle all possible return/statuscodes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.

Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. Theymay not directly introduce a vulnerability, but indicate theproduct has not been carefully developed or maintained.

Findings

Summary

Here is a summary of our findings after analyzing the MoonSwap Smart Contract Review. During the first phase of our audit, we studied the smart contract source code and ran our in-house static code analyzer through the Specific tool. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by tool. We further manually review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	No. of Issues
Critical	0
High	1
Medium	8
Low	17
Total	26

We have so far identified that there are potential issues with severity of **0 Critical, 1 High, 8 Medium, and 17 Low**. Overall, these smart contracts are well-designed and engineered, though the implementation can be improved and bug free by common recommendations given under POCs.

Detailed Results

Basic Coding Bugs

1. **Reentrancy vulnerabilities**

   - Severity: High
   - Result: Found
   - Confidence: Medium
   - Description: The return value of an external file transfer/transferFrom call is not checked.
   - POC:

     External calls:
     - swapAndLiquify(contractTokenBalance) (#1108)
       - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (#1173-1180)
       - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (#1159-1165)

     External calls sending eth:
     - swapAndLiquify(contractTokenBalance) (#1108)
       - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (#1173-1180)

     State variables written after the call(s):
     - _tokenTransfer(from,to,amount,takeFee) (#1124)
       - _rOwned[address(this)] = _rOwned[address(this)].add(rLiquidity) (#1033)
       - _rOwned[sender] = _rOwned[sender].sub(rAmount) (#1215)
       - _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (#1216)
       - _rOwned[sender] = _rOwned[sender].sub(rAmount) (#1206)
       - _rOwned[sender] = _rOwned[sender].sub(rAmount) (#1223)
       - _rOwned[sender] = _rOwned[sender].sub(rAmount) (#913)
       - _rOwned[sender] = _rOwned[sender].sub(rAmount) (#1234)
       - _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (#1207)
       - _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (#1225)
       - _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (#1235)
       - _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (#915)

     - _tokenTransfer(from,to,amount,takeFee) (#1124)
       - _rTotal = _rTotal.sub(rFee) (#986)

     - _tokenTransfer(from,to,amount,takeFee) (#1124)
       - _tOwned[address(this)] = _tOwned[address(this)].add(tLiquidity) (#1035)
       - _tOwned[sender] = _tOwned[sender].sub(tAmount) (#1233)
       - _tOwned[sender] = _tOwned[sender].sub(tAmount) (#912)
       - _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount) (#1224)
       - _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount) (#914)

```
1077    function _transfer(
1078        address from,
1079        address to,
1080        uint256 amount
1081    ) private {
1082        require(from != address(0), "ERC20: transfer from the zero address");
1083        require(to != address(0), "ERC20: transfer to the zero address");
1084        require(amount > 0, "Transfer amount must be greater than zero");
1085        if(from != owner() && to != owner())
1086            require(amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount.");
1087
1088        // is the token balance of this contract address over the min number of
1089        // tokens that we need to initiate a swap + liquidity lock?
1090        // also, don't get caught in a circular liquidity event.
1091        // also, don't swap & liquify if sender is uniswap pair.
1092        uint256 contractTokenBalance = balanceOf(address(this));
1093
1094        if(contractTokenBalance >= _maxTxAmount)
1095        {
1096            contractTokenBalance = _maxTxAmount;
1097        }
1098
1099        bool overMinTokenBalance = contractTokenBalance >= numTokensSellToAddToLiquidity;
1100        if (
1101            overMinTokenBalance &&
1102            !inSwapAndLiquify &&
1103            from != uniswapV2Pair &&
1104            swapAndLiquifyEnabled
1105        ) {
1106            contractTokenBalance = numTokensSellToAddToLiquidity;
1107            //add liquidity
1108            swapAndLiquify(contractTokenBalance);
1109        }
1110
1111        //indicates if fee should be deducted from transfer
1112        bool takeFee = true;
1113
1114        //if any account belongs to _isExcludedFromFee account then remove the fee
1115        if(_isExcludedFromFee[from] || _isExcludedFromFee[to]){
1116            takeFee = false;
1117        }
1118
1119
1120        updateFeestructure();
1121
1122
1123        //transfer amount, it will take tax, burn, liquidity fee
1124        _tokenTransfer(from,to,amount,takeFee);
1125    }
```

- Recommendation: Avoid use of call.value. Update all bookkeeping state variables before transferring execution to an external contract.

2. Divide before Multiply

- Severity: Medium
- Result: Found
- Confidence: Medium
- Description: Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.
- POC:

```
966     function _reflectFee(uint256 rFee, uint256 tFee) private {
967
968         if(rFee > 0 && tFee > 0){
969         uint256 OneFifth= tFee.div(_taxFee); // 1 part
970         uint256 rOneFifth= rFee.div(_taxFee); // 1 part
971         uint256 balance=_taxFee.sub(_charity.add(_burn).add(_redistributeTax));
972         uint256 charityFund = OneFifth.mul(_charity);
973         uint256 burnFund = OneFifth.mul(_burn);
974         if(balance > 0){
975         balance=OneFifth.mul(balance);
976         charityFund=charityFund.add(balance.div(2));
977         burnFund=burnFund.add(balance.div(2));
978         }
979         _transferInternal(msg.sender,CHARITY,charityFund);
980         _transferInternal(msg.sender,BURN,burnFund);
981         uint256 rfunds= rOneFifth.mul(_charity);
982         rfunds =rfunds.add(rOneFifth.mul(_burn));
983         rFee= rFee.sub(rfunds);
984         tFee=tFee.sub(charityFund);
985         tFee=tFee.sub(burnFund);
986          _rTotal = _rTotal.sub(rFee);
987         _tFeeTotal = _tFeeTotal.add(tFee);
988         }
989
990     }
```

- Recommendation: Consider ordering multiplication before division.

3. Divide before Multiply

- Severity: Medium
- Result: Found
- Confidence: Medium
- Description: Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.
- POC:

```
966     function _reflectFee(uint256 rFee, uint256 tFee) private {
967
968         if(rFee > 0 && tFee > 0){
969         uint256 OneFifth= tFee.div( taxFee); // 1 part
970         uint256 rOneFifth= rFee.div(_taxFee); // 1 part
971         uint256 balance=_taxFee.sub(_charity.add(_burn).add(_redistributeTax));
972         uint256 charityFund = OneFifth.mul(_charity);
973         uint256 burnFund = OneFifth.mul(_burn);
974         if(balance > 0){
975         balance=OneFifth.mul(balance);
976         charityFund=charityFund.add(balance.div(2));
977         burnFund=burnFund.add(balance.div(2));
978         }
979         _transferInternal(msg.sender,CHARITY,charityFund);
980         _transferInternal(msg.sender,BURN,burnFund);
981         uint256 rfunds= rOneFifth.mul(_charity);
982         rfunds =rfunds.add(rOneFifth.mul(_burn));
983         rFee= rFee.sub(rfunds);
984         tFee=tFee.sub(charityFund);
985         tFee=tFee.sub(burnFund);
986          _rTotal = _rTotal.sub(rFee);
987         _tFeeTotal = _tFeeTotal.add(tFee);
988         }
989
990     }
```

- Recommendation: Consider ordering multiplication before division.

4. **Divide before Multiply**

   - Severity: Medium
   - Result: Found
   - Confidence: Medium
   - Description: Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.
   - POC:

```
966     function _reflectFee(uint256 rFee, uint256 tFee) private {
967
968         if(rFee > 0 && tFee > 0){
969         uint256 OneFifth= tFee.div(_taxFee); // 1 part
970         uint256 rOneFifth= rFee.div(_taxFee); // 1 part
971         uint256 balance=_taxFee.sub(_charity.add(_burn).add(_redistributeTax));
972         uint256 charityFund = OneFifth.mul(_charity);
973         uint256 burnFund = OneFifth.mul(_burn);
974         if(balance > 0){
975         balance=OneFifth.mul(balance);
976         charityFund=charityFund.add(balance.div(2));
977         burnFund=burnFund.add(balance.div(2));
978         }
979         _transferInternal(msg.sender,CHARITY,charityFund);
980         _transferInternal(msg.sender,BURN,burnFund);
981         uint256 rfunds= rOneFifth.mul(_charity);
982         rfunds =rfunds.add(rOneFifth.mul(_burn));
983         rFee= rFee.sub(rfunds);
984         tFee=tFee.sub(charityFund);
985         tFee=tFee.sub(burnFund);
986          _rTotal = _rTotal.sub(rFee);
987         _tFeeTotal = _tFeeTotal.add(tFee);
988         }
989
990     }
```

   - Recommendation: Consider ordering multiplication before division.

5. Divide before Multiply

- Severity: Medium
- Result: Found
- Confidence: Medium
- Description: Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.
- POC:

```
966     function _reflectFee(uint256 rFee, uint256 tFee) private {
967
968         if(rFee > 0 && tFee > 0){
969         uint256 OneFifth= tFee.div( taxFee); // 1 part
970         uint256 rOneFifth= rFee.div(_taxFee); // 1 part
971         uint256 balance=_taxFee.sub(_charity.add(_burn).add(_redistributeTax));
972         uint256 charityFund = OneFifth.mul(_charity);
973         uint256 burnFund = OneFifth.mul(_burn);
974         if(balance > 0){
975         balance=OneFifth.mul(balance);
976         charityFund=charityFund.add(balance.div(2));
977         burnFund=burnFund.add(balance.div(2));
978         }
979         _transferInternal(msg.sender,CHARITY,charityFund);
980         _transferInternal(msg.sender,BURN,burnFund);
981         uint256 rfunds= rOneFifth.mul(_charity);
982         rfunds =rfunds.add(rOneFifth.mul(_burn));
983         rFee= rFee.sub(rfunds);
984         tFee=tFee.sub(charityFund);
985         tFee=tFee.sub(burnFund);
986          _rTotal = _rTotal.sub(rFee);
987         _tFeeTotal = _tFeeTotal.add(tFee);
988         }
989
990     }
```

- Recommendation: Consider ordering multiplication before division.

6. Divide before Multiply

- Severity: Medium
- Result: Found
- Confidence: Medium
- Description: Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.
- POC:

```
966     function _reflectFee(uint256 rFee, uint256 tFee) private {
967
968         if(rFee > 0 && tFee > 0){
969         uint256 OneFifth= tFee.div(_taxFee); // 1 part
970         uint256 rOneFifth= rFee.div(_taxFee); // 1 part
971         uint256 balance=_taxFee.sub(_charity.add(_burn).add(_redistributeTax));
972         uint256 charityFund = OneFifth.mul(_charity);
973         uint256 burnFund = OneFifth.mul(_burn);
974         if(balance > 0){
975         balance=OneFifth.mul(balance);
976         charityFund=charityFund.add(balance.div(2));
977         burnFund=burnFund.add(balance.div(2));
978         }
979         _transferInternal(msg.sender,CHARITY,charityFund);
980         _transferInternal(msg.sender,BURN,burnFund);
981         uint256 rfunds= rOneFifth.mul(_charity);
982         rfunds =rfunds.add(rOneFifth.mul(_burn));
983         rFee= rFee.sub(rfunds);
984         tFee=tFee.sub(charityFund);
985         tFee=tFee.sub(burnFund);
986          _rTotal = _rTotal.sub(rFee);
987         _tFeeTotal = _tFeeTotal.add(tFee);
988         }
989
990     }
```

- Recommendation: Consider ordering multiplication before division.

7. Dangerous strict equalities

- Severity: Medium
- Result: Found
- Confidence: High
- Description: Use the strict equalities that can be easily manipulated by an attacker.
- POC:

```
1050    function removeAllFee() private {
1051        if(_taxFee == 0 && _liquidityFee == 0) return;
1052
1053        _previousTaxFee = _taxFee;
1054        _previousLiquidityFee = _liquidityFee;
1055
1056        _taxFee = 0;
1057        _liquidityFee = 0;
1058    }
```

- Recommendation: Don't use strict equality to determine if an account has enough Ether or tokens.

8. Dangerous strict equalities

- Severity: Medium
- Result: Found
- Confidence: High
- Description: Use the strict equalities that can be easily manipulated by an attacker.
- POC:

```
1241    function updateFeestructure()private{
1242        for(uint256 i=0; i< fees.length ; i++){
1243            if(fees[i].isUnlocked == false && block.timestamp >= fees[i].unlockedTime){
1244                    _taxFee = fees[i].taxFee;
1245                    _liquidityFee = fees[i].liquidityFee;
1246                    _redistributeTax=fees[i].redistributeTax;
1247                    _burn=fees[i].burn;
1248                    _charity=fees[i].charity;
1249                    fees[i].isUnlocked=true;
1250            }
1251        }
1252
1253    }
```

- Recommendation: Don’t use strict equality to determine if an account has enough Ether or tokens.

9. Unused return

- Severity: Medium
- Result: Found
- Confidence: Medium
- Description: The return value of an external call is not stored in a local or state variable.
- POC:

```
1168    function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
1169        // approve token transfer to cover all possible scenarios
1170        _approve(address(this), address(uniswapV2Router), tokenAmount);
1171
1172        // add the liquidity
1173        uniswapV2Router.addLiquidityETH{value: ethAmount}(
1174            address(this),
1175            tokenAmount,
1176            0, // slippage is unavoidable
1177            0, // slippage is unavoidable
1178            owner(),
1179            block.timestamp
1180        );
1181    }
```

- Recommendation: Ensure that all the return values of the function calls are used.

10. Local variable shadowing.

- Severity: Low
- Result: Found
- Confidence: High
- Description: Detection of shadowing using local variables.
- POC:

```
829     function allowance(address owner, address spender) public view override returns (uint256) {
830         return _allowances[owner][spender];
831     }
```

- Recommendation: Rename the local variables that shadow another component.

11. Local variable shadowing.

- Severity: Low
- Result: Found
- Confidence: High
- Description: Detection of shadowing using local variables.
- POC:

```
1069    function _approve(address owner, address spender, uint256 amount) private {
1070        require(owner != address(0), "ERC20: approve from the zero address");
1071        require(spender != address(0), "ERC20: approve to the zero address");
1072
1073        _allowances[owner][spender] = amount;
1074        emit Approval(owner, spender, amount);
1075    }
```

- Recommendation: Rename the local variables that shadow another component.

12. Reentrancy vulnerabilities

- Severity: Low
- Result: Found
- Confidence: Medium
- Description: Detect missing events for critical arithmetic parameters.
- POC:
  External calls:
  - swapAndLiquify(contractTokenBalance) (#1108)
    - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (#1173-1180)
    - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (#1159-1165)

  External calls sending eth:
  - swapAndLiquify(contractTokenBalance) (#1108)
    - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (#1173-1180)

  State variables written after the call(s):
  - updateFeestructure() (#1120)
    - _burn = fees[i].burn (#1247)
  - updateFeestructure() (#1120)
    - _charity = fees[i].charity (#1248)
  - updateFeestructure() (#1120)
    - _liquidityFee = fees[i].liquidityFee (#1245)
  - _tokenTransfer(from,to,amount,takeFee) (#1124)
    - _liquidityFee = _previousLiquidityFee (#1062)
    - _liquidityFee = 0 (#1057)
  - _tokenTransfer(from,to,amount,takeFee) (#1124)
    - _previousLiquidityFee = _liquidityFee (#1054)
  - _tokenTransfer(from,to,amount,takeFee) (#1124)
    - _previousTaxFee = _taxFee (#1053)
  - updateFeestructure() (#1120)
    - _redistributeTax = fees[i].redistributeTax (#1246)
  - _tokenTransfer(from,to,amount,takeFee) (#1124)
    - _tFeeTotal = _tFeeTotal.add(tFee) (#987)
  - updateFeestructure() (#1120)
    - _taxFee = fees[i].taxFee (#1244)
  - _tokenTransfer(from,to,amount,takeFee) (#1124)
    - _taxFee = _previousTaxFee (#1061)
    - _taxFee = 0 (#1056)

- updateFeestructure() (#1120)
    - fees[i].isUnlocked = true (#1249)

```
1077      function _transfer(
1078          address from,
1079          address to,
1080          uint256 amount
1081      ) private {
1082          require(from != address(0), "ERC20: transfer from the zero address");
1083          require(to != address(0), "ERC20: transfer to the zero address");
1084          require(amount > 0, "Transfer amount must be greater than zero");
1085          if(from != owner() && to != owner())
1086              require(amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount.");
1087
1088          // is the token balance of this contract address over the min number of
1089          // tokens that we need to initiate a swap + liquidity lock?
1090          // also, don't get caught in a circular liquidity event.
1091          // also, don't swap & liquify if sender is uniswap pair.
1092          uint256 contractTokenBalance = balanceOf(address(this));
1093
1094          if(contractTokenBalance >= _maxTxAmount)
1095          {
1096              contractTokenBalance = _maxTxAmount;
1097          }
1098
1099          bool overMinTokenBalance = contractTokenBalance >= numTokensSellToAddToLiquidity;
1100          if (
1101              overMinTokenBalance &&
1102              !inSwapAndLiquify &&
1103              from != uniswapV2Pair &&
1104              swapAndLiquifyEnabled
1105          ) {
1106              contractTokenBalance = numTokensSellToAddToLiquidity;
1107              //add liquidity
1108              swapAndLiquify(contractTokenBalance);
1109          }
1110
1111          //indicates if fee should be deducted from transfer
1112          bool takeFee = true;
1113
1114          //if any account belongs to _isExcludedFromFee account then remove the fee
1115          if(_isExcludedFromFee[from] || _isExcludedFromFee[to]){
1116              takeFee = false;
1117          }
1118
1119
1120          updateFeestructure();
1121
1122
1123          //transfer amount, it will take tax, burn, liquidity fee
1124          _tokenTransfer(from,to,amount,takeFee);
1125      }
```

- Recommendation: Emit an event for critical parameter changes.

13. Reentrancy vulnerabilities

- Severity: Low
- Result: Found
- Confidence: Medium
- Description: Detect missing events for critical arithmetic parameters.
- POC:
  External calls:
  - uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (#761-762)

  State variables written after the call(s):
  - _isExcludedFromFee[owner()] = true (#768)
  - _isExcludedFromFee[address(this)] = true (#769)
  - fees.push(feesLocked(false,8,2,2,1,5,block.timestamp + 63072000)) (#777-787)
  - fees.push(feesLocked(false,uint256(7).div(2),uint256(3).div(2),1,uint256(1).div(2),2,block.timestamp + 157680000)) (#790-800)
  - uniswapV2Router = _uniswapV2Router (#765)

```
756     constructor () public {
757         _rOwned[_msgSender()] = _rTotal;
758
759         IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02
    (0x3605E7B305D04E826726DDa1e81cc6CCD0AF8FAC);
760          // Create a uniswap pair for this new token
761         uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
762             .createPair(address(this), _uniswapV2Router.WETH());
763
764         // set the rest of the contract variables
765         uniswapV2Router = _uniswapV2Router;
766
767         //exclude owner and this contract from fee
768         _isExcludedFromFee[owner()] = true;
769         _isExcludedFromFee[address(this)] = true;
770         init();
771
772         emit Transfer(address(0), _msgSender(), _tTotal);
773     }
```

- Recommendation: Emit an event for critical parameter changes.

14. Reentrancy vulnerabilities

- Severity: Low
- Result: Found
- Confidence: Medium
- Description: Detect missing events for critical arithmetic parameters.
- POC:
  External calls:
  - swapTokensForEth(half) (#1139)
    - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (#1159-1165)
  - addLiquidity(otherHalf,newBalance) (#1145)
    - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (#1173-1180)

  External calls sending eth:
  - addLiquidity(otherHalf,newBalance) (#1145)
    - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (#1173-1180)

  State variables written after the call(s):
  - addLiquidity(otherHalf,newBalance) (#1145)
    - _allowances[owner][spender] = amount (#1073)

```
1127    function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
1128        // split the contract balance into halves
1129        uint256 half = contractTokenBalance.div(2);
1130        uint256 otherHalf = contractTokenBalance.sub(half);
1131
1132        // capture the contract's current ETH balance.
1133        // this is so that we can capture exactly the amount of ETH that the
1134        // swap creates, and not make the liquidity event include any ETH that
1135        // has been manually sent to the contract
1136        uint256 initialBalance = address(this).balance;
1137
1138        // swap tokens for ETH
1139        swapTokensForEth(half); // <- this breaks the ETH -> HATE swap when swap+liquify
    is triggered
1140
1141        // how much ETH did we just swap into?
1142        uint256 newBalance = address(this).balance.sub(initialBalance);
1143
1144        // add liquidity to uniswap
1145        addLiquidity(otherHalf, newBalance);
1146
1147        emit SwapAndLiquify(half, newBalance, otherHalf);
1148    }
```

- Recommendation: Emit an event for critical parameter changes.

15. Reentrancy vulnerabilities

- Severity: Low
- Result: Found
- Confidence: Medium
- Description: Detect missing events for critical arithmetic parameters.
- POC:
  External calls:
  - _transfer(sender,recipient,amount) (#839)
  - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (#1173-1180)
  - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (#1159-1165)

  External calls sending eth:
  - _transfer(sender,recipient,amount) (#839)
  - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (#1173-1180)

  State variables written after the call(s):
  - _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (#840)
  - _allowances[owner][spender] = amount (#1073)

```
838     function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) {
839         _transfer(sender, recipient, amount);
840         _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
841         return true;
842     }
```

- Recommendation: Emit an event for critical parameter changes.

16. Reentrancy vulnerabilities

- Severity: Low
- Result: Found
- Confidence: Medium
- Description: A state variable is changed after a contract uses call.value. The attacker uses a fallback function—which is automatically executed after Ether is transferred from the targeted contract—to execute the vulnerable function again, before the state variable is changed.
- POC:
  External calls:
  - swapAndLiquify(contractTokenBalance) (#1108)
    - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (#1173-1180)
    - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (#1159-1165)

  External calls sending eth:
  - swapAndLiquify(contractTokenBalance) (#1108)
    - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (#1173-1180)

  Event emitted after the call(s):
  - Transfer(sender,recipient,tTransferAmount) (#1218)
    - _tokenTransfer(from,to,amount,takeFee) (#1124)
  - Transfer(sender,recipient,tTransferAmount) (#1210)
    - _tokenTransfer(from,to,amount,takeFee) (#1124)
  - Transfer(sender,recipient,tTransferAmount) (#1228)
    - _tokenTransfer(from,to,amount,takeFee) (#1124)
  - Transfer(sender,recipient,tTransferAmount) (#1238)
    - _tokenTransfer(from,to,amount,takeFee) (#1124)
  - Transfer(sender,recipient,tTransferAmount) (#918)
    - _tokenTransfer(from,to,amount,takeFee) (#1124)

```
1077      function _transfer(
1078          address from,
1079          address to,
1080          uint256 amount
1081      ) private {
1082          require(from != address(0), "ERC20: transfer from the zero address");
1083          require(to != address(0), "ERC20: transfer to the zero address");
1084          require(amount > 0, "Transfer amount must be greater than zero");
1085          if(from != owner() && to != owner())
1086              require(amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount.");
1087
1088          // is the token balance of this contract address over the min number of
1089          // tokens that we need to initiate a swap + liquidity lock?
1090          // also, don't get caught in a circular liquidity event.
1091          // also, don't swap & liquify if sender is uniswap pair.
1092          uint256 contractTokenBalance = balanceOf(address(this));
1093
1094          if(contractTokenBalance >= _maxTxAmount)
1095          {
1096              contractTokenBalance = _maxTxAmount;
1097          }
1098
1099          bool overMinTokenBalance = contractTokenBalance >= numTokensSellToAddToLiquidity;
1100          if (
1101              overMinTokenBalance &&
1102              !inSwapAndLiquify &&
1103              from != uniswapV2Pair &&
1104              swapAndLiquifyEnabled
1105          ) {
1106              contractTokenBalance = numTokensSellToAddToLiquidity;
1107              //add liquidity
1108              swapAndLiquify(contractTokenBalance);
1109          }
1110
1111          //indicates if fee should be deducted from transfer
1112          bool takeFee = true;
1113
1114          //if any account belongs to _isExcludedFromFee account then remove the fee
1115          if(_isExcludedFromFee[from] || _isExcludedFromFee[to]){
1116              takeFee = false;
1117          }
1118
1119
1120          updateFeestructure();
1121
1122
1123          //transfer amount, it will take tax, burn, liquidity fee
1124          _tokenTransfer(from,to,amount,takeFee);
1125      }
```

- Recommendation: Avoid use of call.value Update all bookkeeping state variables before transferring execution to an external contract

17. Reentrancy vulnerabilities

- Severity: Low
- Result: Found
- Confidence: Medium
- Description: A state variable is changed after a contract uses call.value. The attacker uses a fallback function—which is automatically executed after Ether is transferred from the targeted contract—to execute the vulnerable function again, before the state variable is changed.
- POC:
  External calls:
  - uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (#761-762)

  Event emitted after the call(s):
  - Transfer(address(0),_msgSender(),_tTotal) (#772)

```
756     constructor () public {
757         _rOwned[_msgSender()] = _rTotal;
758
759         IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02
    (0x3605E7B305D04E826726DDa1e81cc6CCD0AF8FAC);
760         // Create a uniswap pair for this new token
761         uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
762             .createPair(address(this), _uniswapV2Router.WETH());
763
764         // set the rest of the contract variables
765         uniswapV2Router = _uniswapV2Router;
766
767         //exclude owner and this contract from fee
768         _isExcludedFromFee[owner()] = true;
769         _isExcludedFromFee[address(this)] = true;
770         init();
771
772         emit Transfer(address(0), _msgSender(), _tTotal);
773     }
```

- Recommendation: Avoid use of call.value
  Update all bookkeeping state variables before transferring execution to an external contract

18. Reentrancy vulnerabilities

- Severity: Low
- Result: Found
- Confidence: Medium
- Description: A state variable is changed after a contract uses call.value. The attacker uses a fallback function—which is automatically executed after Ether is transferred from the targeted contract—to execute the vulnerable function again, before the state variable is changed.
- POC:
  External calls:
  - swapTokensForEth(half) (#1139)
    - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (#1159-1165)
  - addLiquidity(otherHalf,newBalance) (#1145)
    - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (#1173-1180)

  External calls sending eth:
  - addLiquidity(otherHalf,newBalance) (#1145)
    - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (#1173-1180)

  Event emitted after the call(s):
  - Approval(owner,spender,amount) (#1074)
    - addLiquidity(otherHalf,newBalance) (#1145)
    - SwapAndLiquify(half,newBalance,otherHalf) (#1147)

```
1127    function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
1128        // split the contract balance into halves
1129        uint256 half = contractTokenBalance.div(2);
1130        uint256 otherHalf = contractTokenBalance.sub(half);
1131
1132        // capture the contract's current ETH balance.
1133        // this is so that we can capture exactly the amount of ETH that the
1134        // swap creates, and not make the liquidity event include any ETH that
1135        // has been manually sent to the contract
1136        uint256 initialBalance = address(this).balance;
1137
1138        // swap tokens for ETH
1139        swapTokensForEth(half); // <- this breaks the ETH -> HATE swap when swap+liquify
    is triggered
1140
1141        // how much ETH did we just swap into?
1142        uint256 newBalance = address(this).balance.sub(initialBalance);
1143
1144        // add liquidity to uniswap
1145        addLiquidity(otherHalf, newBalance);
1146
1147        emit SwapAndLiquify(half, newBalance, otherHalf);
1148    }
```

- Recommendation: Avoid use of call.value
  Update all bookkeeping state variables before transferring execution to an external contract

19. Reentrancy vulnerabilities

- Severity: Low
- Result: Found
- Confidence: Medium
- Description: A state variable is changed after a contract uses call.value. The attacker uses a fallback function—which is automatically executed after Ether is transferred from the targeted contract—to execute the vulnerable function again, before the state variable is changed.
- POC:
  External calls:
  - _transfer(sender,recipient,amount) (#839)
    - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (#1173-1180)
    - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (#1159-1165)

  External calls sending eth:
  - _transfer(sender,recipient,amount) (#839)
    - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (#1173-1180)

  Event emitted after the call(s):
  - Approval(owner,spender,amount) (#1074)
    - _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (#840)

```
838     function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) {
839         _transfer(sender, recipient, amount);
840         _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
841         return true;
842     }
```

- Recommendation: Avoid use of call.value
  Update all bookkeeping state variables before transferring execution to an external contract

20. Block timestamp

- Severity: Low
- Result: Found
- Confidence: Medium
- Description: Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.
- POC:

```
463     function unlock() public virtual {
464         require(_previousOwner == msg.sender, "You don't have permission to unlock");
465         require(now > _lockTime , "Contract is locked until 7 days");
466         emit OwnershipTransferred(_owner, _previousOwner);
467         _owner = _previousOwner;
468     }
```

- Recommendation: Avoid relying on block.timestamp.

21. Block timestamp

- Severity: Low
- Result: Found
- Confidence: Medium
- Description: Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.
- POC:

```
882     function tokenFromReflection(uint256 rAmount) public view returns(uint256) {
883         require(rAmount <= _rTotal, "Amount must be less than total reflections");
884         uint256 currentRate =  _getRate();
885         return rAmount.div(currentRate);
886     }
```

- Recommendation: Avoid relying on block.timestamp.

22. Block timestamp

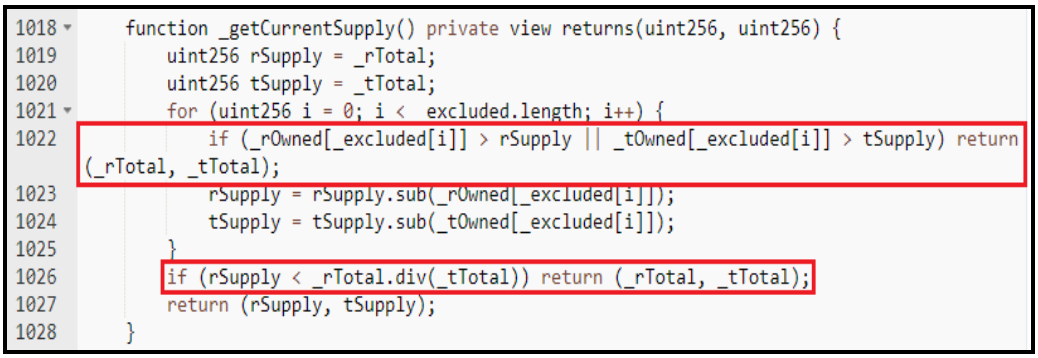
- Severity: Low
- Result: Found
- Confidence: Medium
- Description: Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.
- POC:

```
966     function _reflectFee(uint256 rFee, uint256 tFee) private {
967
968         if(rFee > 0 && tFee > 0){
969         uint256 OneFifth= tFee.div(_taxFee); // 1 part
970         uint256 rOneFifth= rFee.div(_taxFee); // 1 part
971         uint256 balance=_taxFee.sub(_charity.add(_burn).add(_redistributeTax));
972         uint256 charityFund = OneFifth.mul(_charity);
973         uint256 burnFund = OneFifth.mul(_burn);
974         if(balance > 0){
975         balance=OneFifth.mul(balance);
976         charityFund=charityFund.add(balance.div(2));
977         burnFund=burnFund.add(balance.div(2));
978         }
979         _transferInternal(msg.sender,CHARITY,charityFund);
980         _transferInternal(msg.sender,BURN,burnFund);
981         uint256 rfunds= rOneFifth.mul(_charity);
982         rfunds =rfunds.add(rOneFifth.mul(_burn));
983         rFee= rFee.sub(rfunds);
984         tFee=tFee.sub(charityFund);
985         tFee=tFee.sub(burnFund);
986          _rTotal = _rTotal.sub(rFee);
987         _tFeeTotal = _tFeeTotal.add(tFee);
988         }
989
990     }
```

- Recommendation: Avoid relying on block.timestamp.

23. Block timestamp

- Severity: Low
- Result: Found
- Confidence: Medium
- Description: Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.
- POC:



```
1018    function _getCurrentSupply() private view returns(uint256, uint256) {
1019        uint256 rSupply = _rTotal;
1020        uint256 tSupply = _tTotal;
1021        for (uint256 i = 0; i <  excluded.length; i++) {
1022            if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return
    (_rTotal, _tTotal);
1023            rSupply = rSupply.sub(_rOwned[_excluded[i]]);
1024            tSupply = tSupply.sub(_tOwned[_excluded[i]]);
1025        }
1026        if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
1027        return (rSupply, tSupply);
1028    }
```

- Recommendation: Avoid relying on block.timestamp.

24. Block timestamp

- Severity: Low
- Result: Found
- Confidence: Medium
- Description: Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.
- POC:

```
1050    function removeAllFee() private {
1051        if(_taxFee == 0 && _liquidityFee == 0) return;
1052
1053        _previousTaxFee = _taxFee;
1054        _previousLiquidityFee = _liquidityFee;
1055
1056        _taxFee = 0;
1057        _liquidityFee = 0;
1058    }
```

- Recommendation: Avoid relying on block.timestamp.

25. Block timestamp

- Severity: Low
- Result: Found
- Confidence: Medium
- Description: Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.
- POC:

```
1077    function _transfer(
1078        address from,
1079        address to,
1080        uint256 amount
1081    ) private {
1082        require(from != address(0), "ERC20: transfer from the zero address");
1083        require(to != address(0), "ERC20: transfer to the zero address");
1084        require(amount > 0, "Transfer amount must be greater than zero");
1085        if(from != owner() && to != owner())
1086            require(amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount.");
1087
1088        // is the token balance of this contract address over the min number of
1089        // tokens that we need to initiate a swap + liquidity lock?
1090        // also, don't get caught in a circular liquidity event.
1091        // also, don't swap & liquify if sender is uniswap pair.
1092        uint256 contractTokenBalance = balanceOf(address(this));
1093
1094        if(contractTokenBalance >= _maxTxAmount)
1095        {
1096            contractTokenBalance = _maxTxAmount;
1097        }
1098
1099        bool overMinTokenBalance = contractTokenBalance >= numTokensSellToAddToLiquidity;
1100        if (
1101            overMinTokenBalance &&
1102            !inSwapAndLiquify &&
1103            from != uniswapV2Pair &&
1104            swapAndLiquifyEnabled
1105        ) {
1106            contractTokenBalance = numTokensSellToAddToLiquidity;
1107            //add liquidity
1108            swapAndLiquify(contractTokenBalance);
1109        }
1110
1111        //indicates if fee should be deducted from transfer
1112        bool takeFee = true;
1113
1114        //if any account belongs to _isExcludedFromFee account then remove the fee
1115        if(_isExcludedFromFee[from] || _isExcludedFromFee[to]){
1116            takeFee = false;
1117        }
1118
1119
1120        updateFeestructure();
1121
1122
1123        //transfer amount, it will take tax, burn, liquidity fee
1124        _tokenTransfer(from,to,amount,takeFee);
1125    }
```

- Recommendation: Avoid relying on block.timestamp.

26. Block timestamp

- Severity: Low
- Result: Found
- Confidence: Medium
- Description: Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.
- POC:

```
1241    function updateFeestructure()private{
1242        for(uint256 i=0; i< fees.length ; i++){
1243            if(fees[i].isUnlocked == false && block.timestamp >= fees[i].unlockedTime){
1244                    _taxFee = fees[i].taxFee;
1245                    _liquidityFee = fees[i].liquidityFee;
1246                    _redistributeTax=fees[i].redistributeTax;
1247                    _burn=fees[i].burn;
1248                    _charity=fees[i].charity;
1249                    fees[i].isUnlocked=true;
1250                }
1251            }
1252
1253    }
```

- Recommendation: Avoid relying on block.timestamp.

Basic Coding Bugs

27. Constructor Mismatch

- Description: Whether the contract name and its constructor are not identical to each other.
- Result: Not found
- Severity: Critical

28. Ownership Takeover

- Description: Whether the set owner function is not protected.
- Result: Not found
- Severity: Critical

29. Redundant Fallback Function

- Description: Whether the contract has a redundant fallback function.
- Result: Not found
- Severity: Critical

30. Overflows & Underflows

- Description: Whether the contract has general overflow or underflow vulnerabilities
- Result: Not found
- Severity: Critical

31. Reentrancy

- Description: Reentrancy is an issue when code can call back into your contract and change state, such as withdrawing ETHs.
- Result: Not found
- Severity: Critical

32. Money-Giving Bug

- Description: Whether the contract returns funds to an arbitrary address.
- Result: Not found
- Severity: High

33. Blackhole

- Description: Whether the contract locks ETH indefinitely: merely in without out.
- Result: Not found
- Severity: High

34. Unauthorized Self-Destruct

- Description: Whether the contract can be killed by any arbitrary address.
- Result: Not found
- Severity: Medium

35. Revert DoS

- Description: Whether the contract is vulnerable to DoS attack because of unexpected revert.
- Result: Not found
- Severity: Medium

36. Unchecked External Call

- Description: Whether the contract has any external call without checking the return value.
- Result: Not found
- Severity: Medium

37. Gasless Send

- Description: Whether the contract is vulnerable to gasless send.
- Result: Not found
- Severity: Medium

38. Send Instead of Transfer

- Description: Whether the contract uses send instead of transfer.
- Result: Not found
- Severity: Medium

39. Costly Loop

- Description: Whether the contract has any costly loop which may lead to Out-Of-Gas exception.
- Result: Not found
- Severity: Medium

40. (Unsafe) Use of Untrusted Libraries

- Description: Whether the contract use any suspicious libraries.
- Result: Not found
- Severity: Medium

41. (Unsafe) Use of Predictable Variables

- Description: Whether the contract contains any randomness variable, but its value can be predicated.
- Result: Not found
- Severity: Medium

42. Transaction Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Not found
- Severity: Medium

43. Deprecated Uses

- Description: Whether the contract use the deprecated tx.origin to perform the authorization.
- Result: Not found
- Severity: Medium

Semantic Consistency Checks

- Description: Whether the semantic of the white paper is different from the implementation of the contract.
- Result: Not found
- Severity: Critical

Conclusion

In this audit, we thoroughly analyzed the Hodlers Are Millionaires' 'MoonSwap' Smart Contract. The current code base is well organized but there are promptly some High, medium and low-level issues found in this phase of Smart Contract Audit.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

About eNebula Solutions

We believe that people have a fundamental need to security and that the use of secure solutions enables every person to more freely use the Internet and every other connected technology. We aim to provide security consulting service to help others make their solutions more resistant to unauthorized access to data & inadvertent manipulation of the system. We support teams from the design phase through the production to launch and surely after.

The eNebula Solutions team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities & specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code & networks and build custom tools as necessary.

Although we are a small team, we surely believe that we can have a momentous impact on the world by being translucent and open about the work we do.

For more information about our security consulting, please mail us at – contact@enebula.in.