QuillAudits

Audit Report
June, 2021

HANDOUT

DEFI BSC CHARITY TOKEN

Contents

Introduction

This audit report highlights the overall security of the Handout contract with commit hash 36e47333. There were some fixes suggested which were done in the commit 862a1de5c66be6a9dd140dc49645b07650343f9b. With this report, QuillAudits have tried to ensure the reliability of the smart contract by completing the assessment of their system's architecture and smart contract codebase.

Auditing Approach and Methodologies applied

In this audit, we consider the following crucial features of the code.

- Whether the implementation of token standards.
- Whether the code is secure.
- Whether the code meets the best coding practices.
- Whether the code meets the SWC Registry issue.

The audit has been performed according to the following procedure:

Manual Audit

- Inspecting the code line by line and revert the initial algorithms of the protocol and then compare them with the specification
- Manually analyzing the code for security vulnerabilities.
- Gas Consumption and optimisation
- Assessing the overall project structure, complexity & quality.
- Checking SWC Registry issues in the code.
- Unit testing by writing custom unit testing for each function.
- Checking whether all the libraries used in the code of the latest version.
- Analysis of security on-chain data.
- Analysis of the failure preparations to check how the smart contract performs in case of bugs and vulnerability.

Automated analysis

- Scanning the project's code base with Mythril, Slither, Echidna, Manticore, others.
- Manually verifying (reject or confirm) all the issues found by tools.
- Performing Unit testing.
- Manual Security Testing (SWC-Registry, Overflow)
- Running the tests and checking their coverage.

Audit Details

Project Name: Handout Charity Token
Project Token: HANDO
Contract commit hash: 36e47333f44e4e4646f541b4d170f400c257172c
Updated commit hash: 862a1de5c66be6a9dd140dc49645b07650343f9b
Code-Link: https://gitlab.com/handout-token/hando/-/blob/main/Contract/hando-token-prd.sol
Languages: Solidity
Platforms and Tools: HardHat, Remix, VScode, solhint, and other tools mentioned in the automated analysis section.

Audit Goals

The focus of this audit was to verify whether the smart contract is secure, resilient, and working according to ERC20 specs. The audit activity can be grouped into three categories.

Security
Identifying security related issues within each contract and the system of contract.

Sound Architecture
Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality
A full review of the contract source code. The primary areas of focus include:

- Correctness.
- Section of code with high complexity.
- Readability.
- Quantity and quality of test coverage.

Issue Categories

Every issue in this report was assigned a severity level from the following:

High severity issues

Issues on this level are critical to the smart contract's performance/ functionality and should be fixed before moving to a live environment.

Medium severity issues

Issues on this level could potentially bring problems and should eventually be fixed.

Low severity issues

Issues on this level are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Number of issues per severity

	High	Medium	Low	Informational
Open	0	0	3	2
Acknowledged	0	0	4	2
Closed	0	3	1	2

Manual Audit

SWC Registry test

We have tested SWC registry issues as well. Out of all tests, three issues were found, which are of low priority. We have put the details about it below in the low priority section.

Serial No.	Description	Comments
SWC-132	Unexpected Ether balance	Pass: Avoided strict equality checks for the Ether balance in a contract
SWC-131	Presence of unused variables	Pass: No unused variables
SWC-128	DoS With Block Gas Limit	Pass
SWC-122	Lack of Proper Signature Verification	Pass
SWC-120	Weak Sources of Randomness from Chain Attributes	Pass
SWC-119	Shadowing State Variables	Found (slither-detect)
SWC-118	Incorrect Constructor Name	Pass. No incorrect constructor name used
SWC-116	Timestamp Dependence	Found (low)
SWC-115	Authorization through tx.origin	Pass: No tx.origin found
SWC-114	Transaction Order Dependence	Pass

Serial No.	Description	Comments
SWC-113	DoS with Failed Call	Pass: No failed call
SWC-112	Delegatecall to Untrusted Callee	Pass
SWC-111	Use of Deprecated Solidity Functions	Pass : No deprecated function used
SWC-108	State Variable Default Visibility	Pass: Explicitly defined visibility for all state variables
SWC-107	Reentrancy	Found (low)
SWC-106	Unprotected SELF-DESTRUCT Instruction	Pass: Not found any such vulnerability
SWC-104	Unchecked Call Return Value	Pass: Not found any such vulnerability
SWC-103	Floating Pragma	Found (low)
SWC-102	Outdated Compiler Version	Found (low)
SWC-101	Integer Overflow and Underflow	Pass

High level severity issues

No issues found

Medium level severity issues

Three medium Severity Issue found.

1. **Contract gains to non-withdrawable BNB via the swapAndLiquify function [Line 1031]**

   Description: The **swapAndLiquify** function converts half of the **contractTokenBalance** HANDO tokens to BNB. For every **swapAndLiquify** function call, a small amount of BNB remains in the contract. This amount grows over time with the **swapAndLiquify** function being called throughout the life of the contract. The HANDO contract does not contain a method to withdraw these funds, and the BNB will be locked in the HANDO contract forever.

```
1028            _tokenTransfer(from,to,amount);
1029        }
1030
1031        function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
1032            // split the contract balance into halves
1033            uint256 half = contractTokenBalance.div(2);
1034            uint256 otherHalf = contractTokenBalance.sub(half);
```

   Status: Closed (in the next commit)

   In the next commit, Liquidity Fee was set to “0”, which makes the function SwapAndLiquify not to be called.

2. **Centralized risk in addLiquidity [~Line 1072]**

   This finding focuses on the addLiquidity function calls the uniswapV2Router.addLiquidityETH function with the to address specified as owner() for acquiring the generated LP tokens from the Handout pool. As a result, the _owner address will accumulate a significant portion of LP tokens over time.

```
1071
1072        function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
1073            // approve token transfer to cover all possible scenarios
1074            _approve(address(this), address(uniswapV2Router), tokenAmount);
1075
1076            // add the liquidity
1077            uniswapV2Router.addLiquidityETH{value: ethAmount}(
1078                address(this),
1079                tokenAmount,
1080                0, // slippage is unavoidable
1081                0, // slippage is unavoidable
1082                owner(),
1083                block.timestamp
1084            );
1085        }
```

This is one of the prevalent issues floating around the community and causing a great deal of concern by token holders.

Status: Closed (in the next commit)

Handout considered that there is a need for a function to add liquidity.

Comments from the Handout team:

> “We won’t be the owner of LP tokens, because Unicrypt will initiate the liquidity pool on Pancakeswap. They will lock the LP tokens for 266 years. That means we will never be able to take out the liquidity from the pool. Moreover, the liquidity adding fee function is now disabled in the smart contract."

3. Swap fee non-transferable

```
1159
1160    //Call this function after finalizing the presale on unicrypt
1161    function enableAllFees() external onlyOwner() {
1162        _taxFee = 2;
1163        _previousTaxFee = _taxFee;
1164        _liquidityFee = 1;
1165        _previousLiquidityFee = _liquidityFee;
1166        _burnFee = 1;
1167        _previousBurnFee = _taxFee;
1168        _charityFee = 6;
1169        _previousCharityFee = _charityFee;
1170        inSwapAndLiquify = true;
1171        emit SwapAndLiquifyEnabledUpdated(true);
1172    }
1173
```

inSwapAndLiquify = true; emit SwapAndLiquifyEnabledUpdated(true);
Here it should have been
swapAndLiquifyEnabled = true; emit
SwapAndLiquifyEnabledUpdated(true);

Because of inSwapAndLiquify always remains true, and in _transfer() function !inSwapAndLiquify condition check will never be true and thus swapAndLiquify() will NEVER HAPPEN.

Since, condition is not true, it makes many functions (like swapAndliquify(), swapTokensForEth(), addLiquidity()) unused in the whole lifetime contract. This would violate the business logic of the token (2%of LP token will never happen).

Status: Closed (in the next commit)

In the latest commit fee was made 0, which nullifies this issue.

Low level severity issues

1. **Description: Costly Loop [944, 883]**

   The loop in the contract includes state variables like .length of a non-memory array in the condition of the for loops.
   As a result, these state variables consume a lot more extra gas for every iteration of the for a loop.
   The below functions include such loops at the above-mentioned lines:
   - _getCurrentSupply ()
   - includeInReward

```
880
881 ▾   function includeInReward(address account) external onlyOwner() {
882         require(_isExcluded[account], "Account is already excluded");
883 ▾       for (uint256 i = 0; i < _excluded.length; i++) {
884 ▾           if (_excluded[i] == account) {
885                 _excluded[i] = _excluded[_excluded.length - 1];
886                 _tOwned[account] = 0;
887                 _isExcluded[account] = false;
888                 _excluded.pop();
889                 break;
890             }
891         }
892     }
893
```

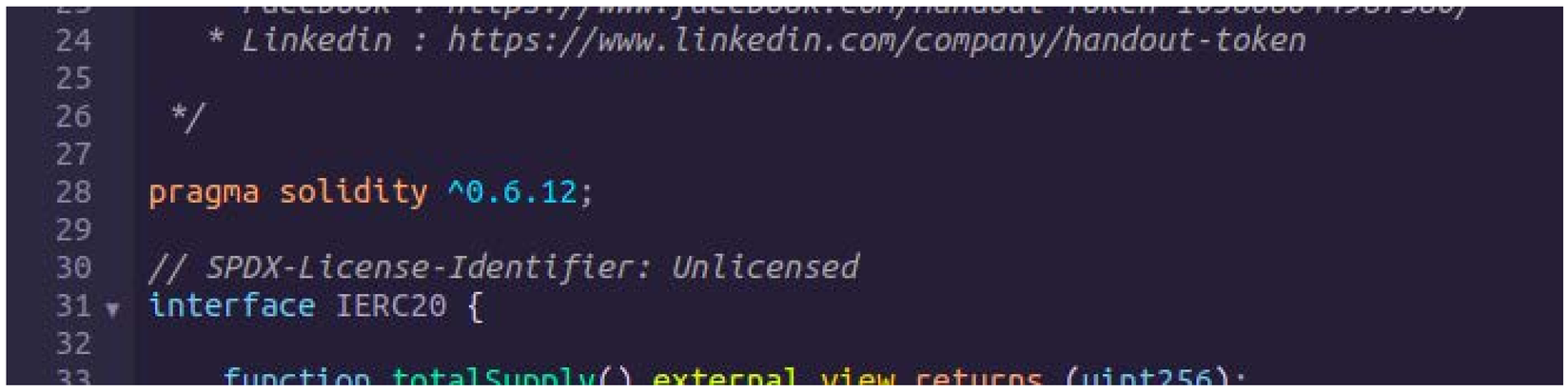
Recommendations: It's quite effective to use a local variable instead of a state variable like .length in a loop. For instance,

```
uint256 local_variable = _groupInfo.addresses.length;
 for (uint256 i = 0; i < local_variable; i++) {
        if (_groupInfo.addresses[i] == msg.sender) {
            _isAddressExistInGroup = true;
            _senderIndex = i;
            break;
        }
    }
```

Reading reference link: https://blog.b9lab.com/getting-loopy-with-solidity-1d51794622ad

Status: Acknowledged by the Auditee.

2. Description → SWC 102: Outdated Compiler Version



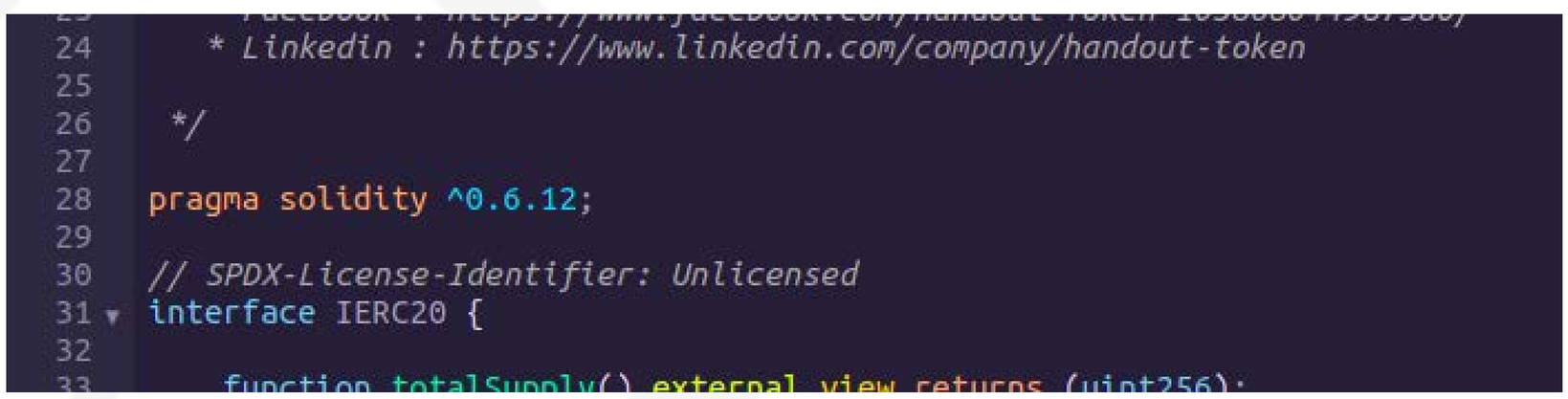
Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.

Remediation
It is recommended to use a recent version of the Solidity compiler, which is Version 0.8.4

Status: Acknowledged by the Auditee.

The handout team acknowledged it citing it wouldn't affect in any way.

3. Description → SWC 103: Floating Pragma



Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation
Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.
Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile it locally.

Status: Open

4. **Description: Potential use of "block.timestamp" as a source of randomness [1068, 1083]**

```
1065                0, // accept any amount of ETH
1066                path,
1067                address(this),
1068                block.timestamp
1069            );
1070        }
1071
1072        function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
1073            // approve token transfer to cover all possible scenarios
1074            _approve(address(this), address(uniswapV2Router), tokenAmount);
1075
1076            // add the liquidity
1077            uniswapV2Router.addLiquidityETH{value: ethAmount}(
1078                address(this),
1079                tokenAmount,
1080                0, // slippage is unavoidable
1081                0, // slippage is unavoidable
1082                owner(),
1083                block.timestamp
1084            );
1085        }
1086
```

Contracts often need access to time values to perform certain types of functionality. Values such as block.timestamp, and block.number can give you a sense of the current time or a time delta; however, they are not safe to use for most purposes.
In the case of block.timestamp, developers often attempt to use it to trigger time-dependent events. As Ethereum is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set a timestamp smaller than the previous one (otherwise, the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

Remediation
Developers should write smart contracts with the notion that block values are not precise, and the use of them can lead to unexpected effects. Alternatively, they may make use of oracles.

References
- Safety: Timestamp dependence
- Ethereum Smart Contract Best Practices - Timestamp Dependence
- How do Ethereum mining nodes maintain a time consistent with network?
- Solidity: Timestamp dependency, is it possible to do safely?

Status: Open

5. **Description: Reentrancy issue (Line: 1001-1029, 1077-1084, 1063-1069, 1024)**

   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.

Attack Scenarios
A contract that holds a map of account balances allows users to call a withdraw function. However, withdraw calls send, which transfers control to the calling contract, but doesn't decrease their balance until after send has finished executing. The attacker can then repeatedly withdraw money that they do not have.

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

Read:
- Solidity guide
- The DAO hack
- The SpankChain hack

Note: This issue was found by slither as well.

Status: Open

6. Using the approve function of the ERC-20 token standard

The ‘approve’ function of the contract is vulnerable. Using a front-running attack one can spend approved tokens before the change of allowance value.

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

To prevent attack vectors described above, clients should make sure to create user interfaces in such a way that they set the allowance first to 0 before setting it to another value for the same spender. However, the contract itself shouldn't enforce it to allow backward compatibility with contracts deployed before.
Detailed reading around it can be found at SWC114 & EIP 20

Status: Acknowledged by the Auditee.

The handout team acknowledged it, citing they use ABI of bscscan, and they aren’t creating their own ABI.

7. Logic incorrectly implemented

Logic to deduct fees is incorrectly implemented.

Scenario: Let's say we are transferring 100 tokens. The fee should be 10 tokens. Now when we transfer 100 tokens, _burnFee and _charityFee are deducted first. (1% and 6%).

Now the amount left will be 93 tokens. Now 93 tokens is sent into the _transferStandard() function. There _taxFee and _liquidityFee will be deducted (1% and 2%). Now the problem is % will be applied on the 93 tokens; it will be 2.79 tokens. Now we will be left with 90.21 tokens (instead of 90 tokens). The total fees was 9.79 tokens (instead of 10 tokens).

Ideally, the fee percentage should be deducted from the principal amount, not the deducted amount.

Status: Closed (in the next commit)

In the next commit, the fee was changed to 0.

8. **Excluded users from fees, can do transactions more than _maxTxAmount.**

   During transfer(), if the sender or recipient is excluded from fees, then they will not be checked for _maxTxAmount. Thus users, that are excluded from fees, can do transactions more than _maxTxAmount.

   Suggestion: Remove Line 1094 out of else { }.

   Status: Acknowledged by the Auditee.

Informational

1. **Function declaration as external**

   A function with a public visibility modifier that is not called internally. Changing the visibility level to external increases code readability. Moreover, in many cases, functions with external visibility modifiers spend less gas compared to functions with public visibility modifiers.

   The function definition in the file are marked as public

   - renounceOwnership()
   - transferOwnership
   - geUnlockTime()
   - lock(uint256)
   - unlock()
   - totalSupply()
   - transfer(address,uint256)
   - Allowance
   - Approve
   - transferFrom
   - increaseAllowance
   - decreaseAllowance
   - isExcludedFromReward
   - totalFees
   - Deliver
   - reflectionFromToken
   - excludeFromReward
   - isExcludedFromFee
   - excludeFromFee
   - includeInFee
   - setSwapAndLiquifyEnabled

   However, it is never directly called by another function in the same contract or in any of its descendants. Consider marking it as"external" instead.

   Recommendations: Use the external visibility modifier for functions never called from the contract via internal call. Reading Link.

Status: Acknowledged by the Auditee.

The handout team acknowledged it, citing that they intended to make it public.

2. **ETH to BNB change**

   The function signature swapTokensForEth(uint256 tokenAmount) does not properly convey the purpose of the function, as the underlying logic actually swaps the Handout token for BNB, and does not swap the token for ETH. It could easily be the case where the team originally set out to launch onto the Ethereum mainnet, but eventually settled on Binance Smart Chain. If this was the case, they could have easily just corrected the contract readability. It could also be the case where they simply copied this logic from another contract.

   The function also mentions the uniswap v2 router in the code instead of pancakeswap, but Handout interacts with pancakeswap, which is a clone of uniswap built for Binance Smart Chain. No logic is affected by these syntactic choices, but it only fuels the accusations of this contract mainly consisting of a simple copy-paste-change effort.

   Status: Open

3. **Not handling variables properly**

   Variables like _tTotal, numTokensSellToAddToLiquidity, _name, _symbol, and _decimals could be declared as constant since these state variables are never changed.

   Status: Open

4. The return value of the function **addLiquidityETH** is not properly handled. Variables can be used to receive the return values of functions mentioned above and handle both success and failure cases if needed by the business logic.

Status: Closed (in the next commit)

In the latest commit fee was made 0, which nullifies this issue.

5. **High fee will impact on the price**

   inSwapAndLiquify = true; emit SwapAndLiquifyEnabledUpdated(true);
   Here it should have been
   swapAndLiquifyEnabled = true; emit
   SwapAndLiquifyEnabledUpdated(true);
   Because of inSwapAndLiquify always remains true, and in _transfer() function !inSwapAndLiquify condition check will never be true and thus swapAndLiquify() will NEVER HAPPEN.

   Status: Closed (in the next commit)

   In the latest commit fee was made 0, which nullifies this issue.

6. **More fee will be calculated before the swap**

   numTokensSellToAddToLiquidity is very high (i.e. 40% of totalSupply, 4* 10^8). That means a lot of fees will have to be calculated before any swap to PCS Router can happen. And when the swap happens, it will impact the price a lot. 20% of HANDO tokens will be converted to BNB which is a lot. Also when adding liquidity to the PCS pool, due to price impact a lot of BNB will be left in the contract. And there is no mechanism to withdraw those tokens.

   Status: Acknowledged by the Auditee.

Functional test

Function test has been done for multiple functions of the files. All functions are called by the current owner. Results are below:

- enableAllFees → activate fees → called by owner only
  -- > PASS

- excludeFromFee → exclude an address bep-20 from fees
  --> PASS

- excludeFromReward → exclude an address bep-20 from reward
  --> PASS

- includeInFee → include an address bep-20 from fees
  --> PASS

- includeInReward → include an address bep-20 from reward
  --> PASS

- renounceOwnership → renounce to the contract
  --> PASS

- transferOwnership → transfer contract to new owner
  --> PASS

- Lock → lock the contract
  --> PASS

Automated Testing

We have used multiple automated testing frameworks. This makes code more secure and common attacks. The results are below.

Slither

Slither is a Solidity static analysis framework that runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses. After running Slither we got the results below.

```
INFO:Detectors:
Reentrancy in hando._transfer(address,address,uint256) (hando.sol#1001-1029):
	External calls:
	- swapAndLiquify(contractTokenBalance) (hando.sol#1024)
		- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (hando.sol#1077-1084)
		- uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (hando.sol#1063-1069)
	External calls sending eth:
	- swapAndLiquify(contractTokenBalance) (hando.sol#1024)
		- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (hando.sol#1077-1084)
	State variables written after the call(s):
	- _tokenTransfer(from,to,amount) (hando.sol#1028)
		- _rOwned[address(this)] = _rOwned[address(this)].add(rLiquidity) (hando.sol#956)
		- _rOwned[sender] = _rOwned[sender].sub(rAmount) (hando.sol#1131)
		- _rOwned[sender] = _rOwned[sender].sub(rAmount) (hando.sol#1140)
		- _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (hando.sol#1132)
		- _rOwned[sender] = _rOwned[sender].sub(rAmount) (hando.sol#897)
		- _rOwned[sender] = _rOwned[sender].sub(rAmount) (hando.sol#1151)
		- _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (hando.sol#1142)
		- _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (hando.sol#1152)
		- _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (hando.sol#899)
	- _tokenTransfer(from,to,amount) (hando.sol#1028)
		- _rTotal = _rTotal.sub(rFee) (hando.sol#911)
	- _tokenTransfer(from,to,amount) (hando.sol#1028)
		- _tOwned[address(this)] = _tOwned[address(this)].add(tLiquidity) (hando.sol#958)
		- _tOwned[sender] = _tOwned[sender].sub(tAmount) (hando.sol#896)
		- _tOwned[sender] = _tOwned[sender].sub(tAmount) (hando.sol#1150)
		- _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount) (hando.sol#1141)
		- _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount) (hando.sol#898)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
hando.addLiquidity(uint256,uint256) (hando.sol#1072-1085) ignores return value by uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (ha
ndo.sol#1077-1084)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
hando.allowance(address,address).owner (hando.sol#812) shadows:
	- Ownable.owner() (hando.sol#440-442) (function)
hando._approve(address,address,uint256).owner (hando.sol#993) shadows:
	- Ownable.owner() (hando.sol#440-442) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
```

```
INFO:Detectors:
Reentrancy in hando._transfer(address,address,uint256) (hando.sol#1001-1029):
        External calls:
        - swapAndLiquify(contractTokenBalance) (hando.sol#1024)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (hando.sol#1077-1084)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (hando.sol#1063-1069)
        External calls sending eth:
        - swapAndLiquify(contractTokenBalance) (hando.sol#1024)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (hando.sol#1077-1084)
        State variables written after the call(s):
        - _tokenTransfer(from,to,amount) (hando.sol#1028)
                - _burnFee = 1 (hando.sol#983)
                - _burnFee = 0 (hando.sol#976)
        - _tokenTransfer(from,to,amount) (hando.sol#1028)
                - _charityFee = 6 (hando.sol#984)
                - _charityFee = 0 (hando.sol#977)
        - _tokenTransfer(from,to,amount) (hando.sol#1028)
                - _liquidityFee = 2 (hando.sol#982)
                - _liquidityFee = 0 (hando.sol#975)
                - _liquidityFee = 0 (hando.sol#1114)
                - _liquidityFee = _previousLiquidityFee (hando.sol#1122)
        - _tokenTransfer(from,to,amount) (hando.sol#1028)
                - _tFeeTotal = _tFeeTotal.add(tFee) (hando.sol#912)
        - _tokenTransfer(from,to,amount) (hando.sol#1028)
                - _taxFee = 1 (hando.sol#981)
                - _taxFee = 0 (hando.sol#974)
                - _taxFee = 0 (hando.sol#1113)
                - _taxFee = _previousTaxFee (hando.sol#1121)
Reentrancy in hando.constructor() (hando.sol#767-784):
        External calls:
        - uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (hando.sol#773-774)
        State variables written after the call(s):
        - _isExcludedFromFee[owner()] = true (hando.sol#780)
        - _isExcludedFromFee[address(this)] = true (hando.sol#781)
        - uniswapV2Router = _uniswapV2Router (hando.sol#777)
Reentrancy in hando.swapAndLiquify(uint256) (hando.sol#1031-1052):
        External calls:
        - swapTokensForEth(half) (hando.sol#1043)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (hando.sol#1063-1069)
        - addLiquidity(otherHalf,newBalance) (hando.sol#1049)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (hando.sol#1077-1084)
        External calls sending eth:
        - addLiquidity(otherHalf,newBalance) (hando.sol#1049)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (hando.sol#1077-1084)
```

```
INFO:Detectors:
Reentrancy in hando._transfer(address,address,uint256) (hando.sol#1001-1029):
        External calls:
        - swapAndLiquify(contractTokenBalance) (hando.sol#1024)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (hando.sol#1077-1084)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (hando.sol#1063-1069)
        External calls sending eth:
        - swapAndLiquify(contractTokenBalance) (hando.sol#1024)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (hando.sol#1077-1084)
        Event emitted after the call(s):
        - Transfer(sender,recipient,tTransferAmount) (hando.sol#1135)
                - _tokenTransfer(from,to,amount) (hando.sol#1028)
        - Transfer(sender,recipient,tTransferAmount) (hando.sol#1145)
                - _tokenTransfer(from,to,amount) (hando.sol#1028)
        - Transfer(sender,recipient,tTransferAmount) (hando.sol#1155)
                - _tokenTransfer(from,to,amount) (hando.sol#1028)
        - Transfer(sender,recipient,tTransferAmount) (hando.sol#902)
                - _tokenTransfer(from,to,amount) (hando.sol#1028)
Reentrancy in hando.constructor() (hando.sol#767-784):
        External calls:
        - uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (hando.sol#773-774)
        Event emitted after the call(s):
        - Transfer(address(0),_msgSender(),_tTotal) (hando.sol#783)
Reentrancy in hando.swapAndLiquify(uint256) (hando.sol#1031-1052):
        External calls:
        - swapTokensForEth(half) (hando.sol#1043)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (hando.sol#1063-1069)
        - addLiquidity(otherHalf,newBalance) (hando.sol#1049)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (hando.sol#1077-1084)
        External calls sending eth:
        - addLiquidity(otherHalf,newBalance) (hando.sol#1049)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (hando.sol#1077-1084)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (hando.sol#998)
                - addLiquidity(otherHalf,newBalance) (hando.sol#1049)
        - SwapAndLiquify(half,newBalance,otherHalf) (hando.sol#1051)
Reentrancy in hando.transferFrom(address,address,uint256) (hando.sol#821-825):
        External calls:
        - _transfer(sender,recipient,amount) (hando.sol#822)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (hando.sol#1077-1084)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (hando.sol#1063-1069)
        External calls sending eth:
        - _transfer(sender,recipient,amount) (hando.sol#822)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (hando.sol#1077-1084)
```

```
INFO:Detectors:
Ownable.unlock() (hando.sol#487-492) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(now < _lockTime,Contract is locked until 7 days) (hando.sol#489)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (hando.sol#292-301) uses assembly
        - INLINE ASM (hando.sol#299)
Address._functionCallWithValue(address,bytes,uint256,string) (hando.sol#385-406) uses assembly
        - INLINE ASM (hando.sol#398-401)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Pragma version^0.6.12 (hando.sol#28) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
solc-0.6.12 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (hando.sol#319-325):
        - (success) = recipient.call{value: amount}() (hando.sol#323)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (hando.sol#385-406):
        - (success,returndata) = target.call{value: weiValue}(data) (hando.sol#389)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (hando.sol#531) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (hando.sol#532) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (hando.sol#549) is not in mixedCase
Function IUniswapV2Router01.WETH() (hando.sol#571) is not in mixedCase
Contract hando (hando.sol#709-1195) is not in CapWords
Parameter hando.calculateTaxFee(uint256)._amount (hando.sol#961) is not in mixedCase
Parameter hando.calculateLiquidityFee(uint256)._amount (hando.sol#967) is not in mixedCase
Parameter hando.setSwapAndLiquifyEnabled(bool)._enabled (hando.sol#1191) is not in mixedCase
Variable hando._taxFee (hando.sol#731) is not in mixedCase
Variable hando._liquidityFee (hando.sol#734) is not in mixedCase
Variable hando._burnFee (hando.sol#737) is not in mixedCase
Variable hando._charityFee (hando.sol#740) is not in mixedCase
Variable hando._maxTxAmount (hando.sol#750) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformity-to-solidity-naming-conventions
INFO:Detectors:
hando.slitherConstructorVariables() (hando.sol#709-1195) uses literals with too many digits:
        - _tTotal = 1000000000 * 10 ** 9 (hando.sol#723)
hando.slitherConstructorVariables() (hando.sol#709-1195) uses literals with too many digits:
        - _maxTxAmount = 1000000000 * 10 ** 9 (hando.sol#750)
hando.slitherConstructorVariables() (hando.sol#709-1195) uses literals with too many digits:
        - numTokensSellToAddToLiquidity = 400000000 * 10 ** 9 (hando.sol#751)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

```
hando.numTokensSellToAddToLiquidity (hando.sol#751) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (hando.sol#459-462)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (hando.sol#468-472)
geUnlockTime() should be declared external:
        - Ownable.geUnlockTime() (hando.sol#474-476)
lock(uint256) should be declared external:
        - Ownable.lock(uint256) (hando.sol#479-484)
unlock() should be declared external:
        - Ownable.unlock() (hando.sol#487-492)
name() should be declared external:
        - hando.name() (hando.sol#786-788)
symbol() should be declared external:
        - hando.symbol() (hando.sol#790-792)
decimals() should be declared external:
        - hando.decimals() (hando.sol#794-796)
totalSupply() should be declared external:
        - hando.totalSupply() (hando.sol#798-800)
transfer(address,uint256) should be declared external:
        - hando.transfer(address,uint256) (hando.sol#807-810)
allowance(address,address) should be declared external:
        - hando.allowance(address,address) (hando.sol#812-814)
approve(address,uint256) should be declared external:
        - hando.approve(address,uint256) (hando.sol#816-819)
transferFrom(address,address,uint256) should be declared external:
        - hando.transferFrom(address,address,uint256) (hando.sol#821-825)
increaseAllowance(address,uint256) should be declared external:
        - hando.increaseAllowance(address,uint256) (hando.sol#827-830)
decreaseAllowance(address,uint256) should be declared external:
        - hando.decreaseAllowance(address,uint256) (hando.sol#832-835)
isExcludedFromReward(address) should be declared external:
        - hando.isExcludedFromReward(address) (hando.sol#837-839)
totalFees() should be declared external:
        - hando.totalFees() (hando.sol#841-843)
deliver(uint256) should be declared external:
        - hando.deliver(uint256) (hando.sol#845-852)
reflectionFromToken(uint256,bool) should be declared external:
        - hando.reflectionFromToken(uint256,bool) (hando.sol#854-863)
excludeFromReward(address) should be declared external:
        - hando.excludeFromReward(address) (hando.sol#871-879)
isExcludedFromFee(address) should be declared external:
        - hando.isExcludedFromFee(address) (hando.sol#989-991)
excludeFromFee(address) should be declared external:
        - hando.excludeFromFee(address) (hando.sol#1158-1160)
includeInFee(address) should be declared external:
        - hando.includeInFee(address) (hando.sol#1162-1164)
setSwapAndLiquifyEnabled(bool) should be declared external:
        - hando.setSwapAndLiquifyEnabled(bool) (hando.sol#1191-1194)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:hando.sol analyzed (10 contracts with 46 detectors), 64 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Manticore

Manticore is a symbolic execution tool for the analysis of smart contracts and binaries. It executes a program with symbolic inputs and explores all the possible states it can reach. It also detects crashes and other failure cases in binaries and smart contracts.
Manticore results throw the same warning which is similar to the Slither warning.

Solhint

Solhint is an open-source project created by https://protofire.io. Its goal is to provide a linting utility for Solidity code. This tool is used as a part of software best practice. Below is the report we got by running solhint.

```
hando.sol
   265:2  error  Line length must be no more than 120 but current length is 132  max-line-length
   317:2  error  Line length must be no more than 120 but current length is 160  max-line-length
   336:2  error  Line length must be no more than 120 but current length is 156  max-line-length
   355:2  error  Line length must be no more than 120 but current length is 122  max-line-length
   380:2  error  Line length must be no more than 120 but current length is 146  max-line-length
   385:2  error  Line length must be no more than 120 but current length is 149  max-line-length
   770:2  error  Line length must be no more than 120 but current length is 136  max-line-length
   771:2  error  Line length must be no more than 120 but current length is 167  max-line-length
   823:2  error  Line length must be no more than 120 but current length is 130  max-line-length
   833:2  error  Line length must be no more than 120 but current length is 138  max-line-length
   895:2  error  Line length must be no more than 120 but current length is 146  max-line-length
   928:2  error  Line length must be no more than 120 but current length is 147  max-line-length
  1130:2  error  Line length must be no more than 120 but current length is 146  max-line-length
  1139:2  error  Line length must be no more than 120 but current length is 146  max-line-length
  1149:2  error  Line length must be no more than 120 but current length is 146  max-line-length

✖ 15 problems (15 errors, 0 warnings)
```

Anchain:

Anchain sandbox audits the security score of any Solidity-based smart contract, having analyzed the source code of every mainnet EVM smart contract plus the 1M + unique, user-uploaded smart contracts. Code has been analyzed there and got the report below.

ETH Smart Contract Audit Report

MD5:5a060617c9b34a35fd200825862a90ce

Runtime:8.0s

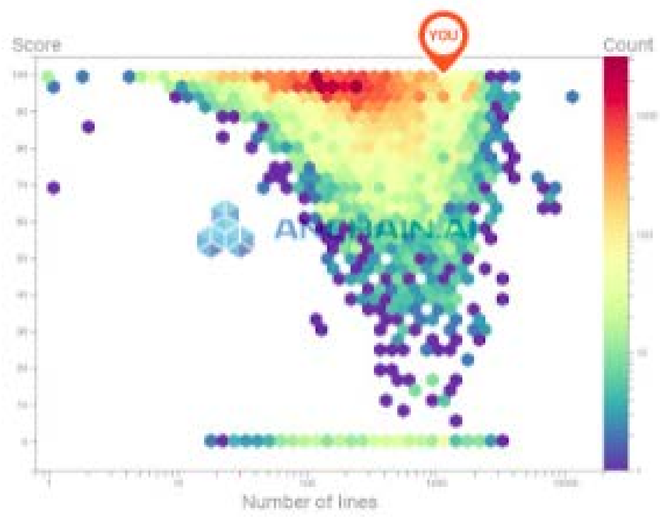
Scored higher than
100% of similar code

amongst 50k smart contracts
audited by Anchain.AI .

Score
100

Threat Level
Low

Number of lines
1195



Overview

Code Class	EVM Coverage
Address	27.0%
Context	0%
IUniswapV2Factory	0%
IUniswapV2Router01	0%
Ownable	0%

0 Vulnerabilities Found

High Risk Medium Risk Low Risk

Vulnerability Checklist

Address

- Integer Underflow
- Integer Overflow
- Parity Multisig Bug
- Callstack Depth Attack
- Transaction-Ordering Dependency
- Timestamp Dependency
- Re-Entrancy

Context

- Integer Underflow
- Integer Overflow
- Parity Multisig Bug
- Callstack Depth Attack
- Transaction-Ordering Dependency
- Timestamp Dependency
- Re-Entrancy

IUniswapV2Factory

- Integer Underflow
- Integer Overflow
- Parity Multisig Bug
- Callstack Depth Attack
- Transaction-Ordering Dependency
- Timestamp Dependency
- Re-Entrancy

IUniswapV2Router01

- Integer Underflow
- Integer Overflow
- Parity Multisig Bug
- Callstack Depth Attack
- Transaction-Ordering Dependency
- Timestamp Dependency
- Re-Entrancy

Ownable

- Integer Underflow
- Integer Overflow
- Parity Multisig Bug
- Callstack Depth Attack
- Transaction-Ordering Dependency
- Timestamp Dependency
- Re-Entrancy

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and
a public bug bounty program to ensure the security of the code. Besides a security audit, please don’t consider this report as investment advice.

Summary

The use of smart contracts is simple, and the code is relatively small. Altogether the code is written and demonstrates effective use of abstraction, separation of concern, and modularity. But there are a few issues/vulnerabilities to be tackled at various security levels; it is recommended to fix them before deploying the contract on the main network. Given the subjective nature of some assessments, it will be up to the Handout team to decide whether any changes should be made.

HANDOUT

DEFI BSC CHARITY TOKEN

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com