# Audit report of USDWSWap_Router

**Prepared By: - Kishan Patel**
Prepared On: - 06/02/2025.

**Prepared for: DWIN INTERTRADE COMPANY LTD.**

# Table of contents

**THIS AUDIT REPORT WILL CONTAIN CONFIDENTIAL INFORMATION ABOUT THE SMART CONTRACT AND INTELLECTUAL PROPERTY OF THE CUSTOMER AS WELL AS INFORMATION ABOUT POTENTIAL VULNERABILITIES OF THEIR EXPLOITATION.**

**THE INFORMATION FROM THIS AUDIT REPORT CAN BE USED INTERNALLY BY THE CUSTOMER OR IT CAN BE DISCLOSED PUBLICLY AFTER ALL VULNERABILITIES ARE FIXED - UPON THE DECISION OF THE CUSTOMER.**

# 1. Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions). Because the total numbers of test cases are unlimited, the audit makes no statements or warranties on the security of the code.

It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

# 2. Introduction

Kishan Patel (Consultant) was contacted by DWIN INTERTRADE COMPANY LIMITED. (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contracts and its code review conducted between 06/02/2025 – 07/02/2025.

The project has 1 files. It contains approx 730 lines of Solidity code. All the functions and state variables are well commented on using the natspec documentation, but that does not create any vulnerability.

# 3. Project information

| | |
|---|---|
| **Token Name** | USDWSwap_Router |
| **Token Symbol** | USDWSwap_Router |
| **Platform** | Binance Smart Chain |
| **Order Started Date** | 06/02/2025 |
| **Order Completed Date** | 07/02/2025 |

# 4. List of attacks checked

- Over and under flows
- Short address attack
- Visibility & Delegate call
- Reentrancy / TheDAO hack
- Forcing BNB to a contract
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference

# 5. Severity Definitions

| Risk | Level Description |
|---|---|
| **Critical** | Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss etc. |
| **Medium** | Medium-level vulnerabilities are important to fix; however, they can’t lead to tokens lose |
| **Low** | Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can’t have significant impact on execution |

# 6. Good things in code

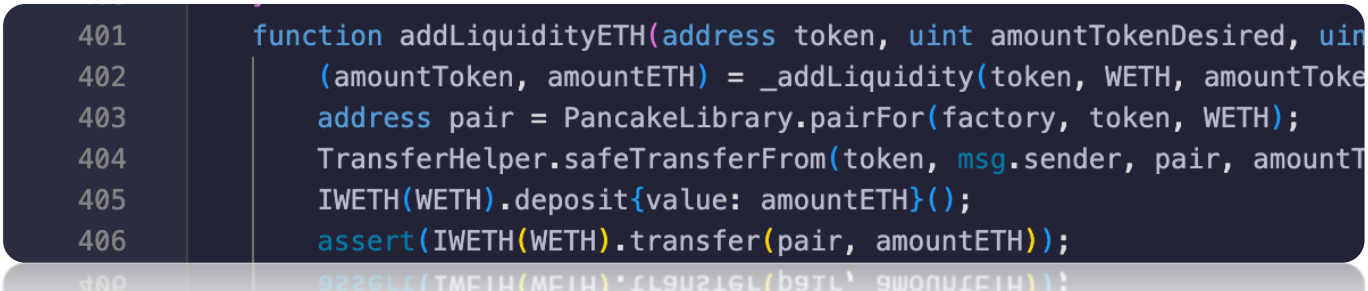
- **Good required condition in functions:-**
  - Here smart contract is checking that msg.sender for direct transfer will be always WETH contract.

```
368     receive() external payable {
369         assert(msg.sender == WETH); // only accept ETH via fallback from
370     }
```

  - Here smart contract is checking that sufficient amounts are available in PancakeRouter.

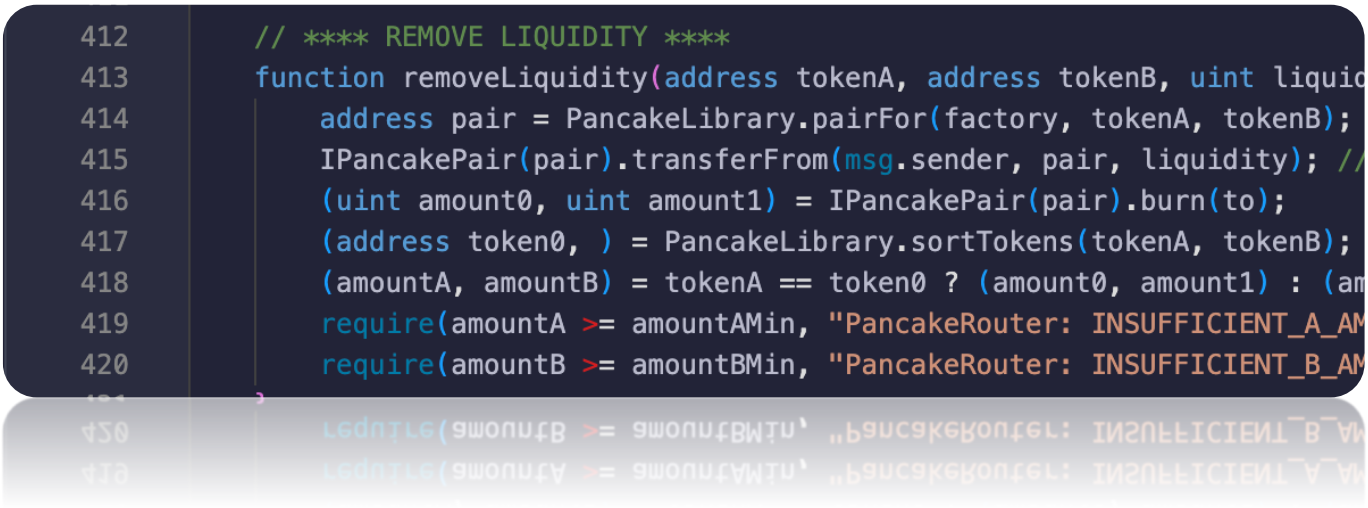
```
372     // **** ADD LIQUIDITY ****
373     function _addLiquidity(address tokenA, address tokenB, uint amountADes
374         // create the pair if it doesn't exist yet
375         if (IPancakeFactory(factory).getPair(tokenA, tokenB) == address(0
376             IPancakeFactory(factory).createPair(tokenA, tokenB);
377         }
378         (uint reserveA, uint reserveB) = PancakeLibrary.getReserves(facto
379         if (reserveA == 0 && reserveB == 0) {
380             (amountA, amountB) = (amountADesired, amountBDesired);
381         } else {
382             uint amountBOptimal = PancakeLibrary.quote(amountADesired, re
383             if (amountBOptimal <= amountBDesired) {
384                 require(amountBOptimal >= amountBMin, "PancakeRouter: INSU
385                 (amountA, amountB) = (amountADesired, amountBOptimal);
386             } else {
387                 uint amountAOptimal = PancakeLibrary.quote(amountBDesired
388                 assert(amountAOptimal <= amountADesired);
389                 require(amountAOptimal >= amountAMin, "PancakeRouter: INSU
390                 (amountA, amountB) = (amountAOptimal, amountBDesired);
391             }
```

- Here smart contract is checking that transfer method of WETH is successful.



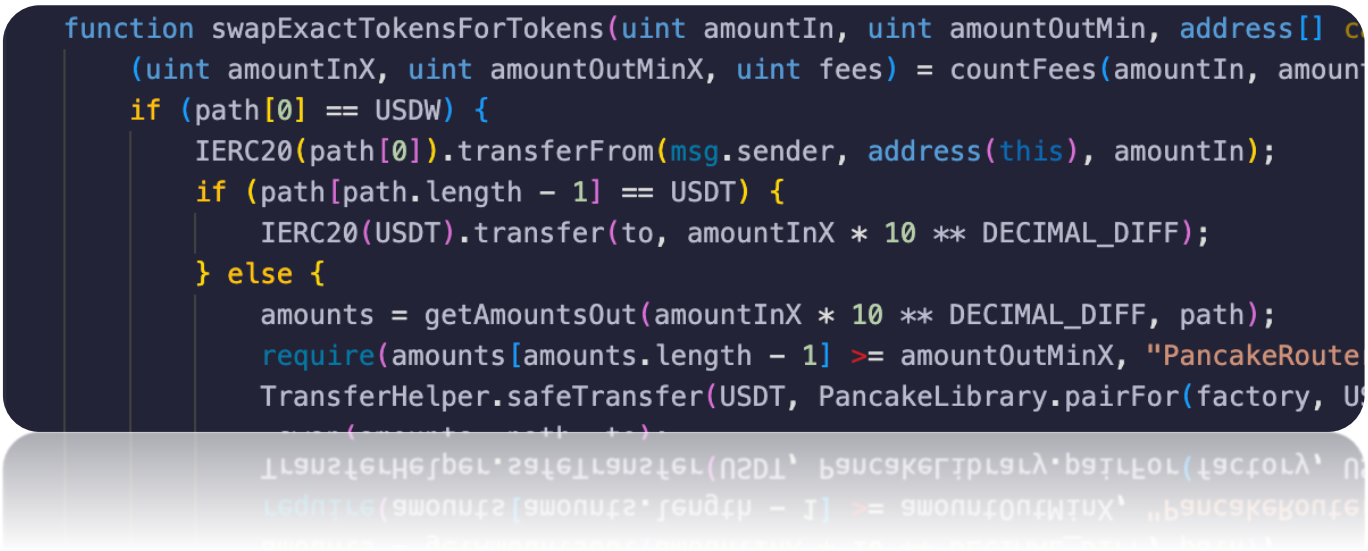
```
401     function addLiquidityETH(address token, uint amountTokenDesired, uin
402         (amountToken, amountETH) = _addLiquidity(token, WETH, amountToke
403         address pair = PancakeLibrary.pairFor(factory, token, WETH);
404         TransferHelper.safeTransferFrom(token, msg.sender, pair, amountT
405         IWETH(WETH).deposit{value: amountETH}();
406         assert(IWETH(WETH).transfer(pair, amountETH));
```

- Here smart contract is checking that sufficient balance for amount a and amount b available in PancakeRouter.



```
412     // **** REMOVE LIQUIDITY ****
413     function removeLiquidity(address tokenA, address tokenB, uint liquic
414         address pair = PancakeLibrary.pairFor(factory, tokenA, tokenB);
415         IPancakePair(pair).transferFrom(msg.sender, pair, liquidity); //
416         (uint amount0, uint amount1) = IPancakePair(pair).burn(to);
417         (address token0, ) = PancakeLibrary.sortTokens(tokenA, tokenB);
418         (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (am
419         require(amountA >= amountAMin, "PancakeRouter: INSUFFICIENT_A_AM
420         require(amountB >= amountBMin, "PancakeRouter: INSUFFICIENT_B_AM
```

- Here smart contract is checking that sufficient output amount available in PancakeRounter.



```
function swapExactTokensForTokens(uint amountIn, uint amountOutMin, address[] c
    (uint amountInX, uint amountOutMinX, uint fees) = countFees(amountIn, amount
    if (path[0] == USDW) {
        IERC20(path[0]).transferFrom(msg.sender, address(this), amountIn);
        if (path[path.length - 1] == USDT) {
            IERC20(USDT).transfer(to, amountInX * 10 ** DECIMAL_DIFF);
        } else {
            amounts = getAmountsOut(amountInX * 10 ** DECIMAL_DIFF, path);
            require(amounts[amounts.length - 1] >= amountOutMinX, "PancakeRoute
            TransferHelper.safeTransfer(USDT, PancakeLibrary.pairFor(factory, U
```

- Here smart contract is checking that sufficient input amount available in PancakeRounter.

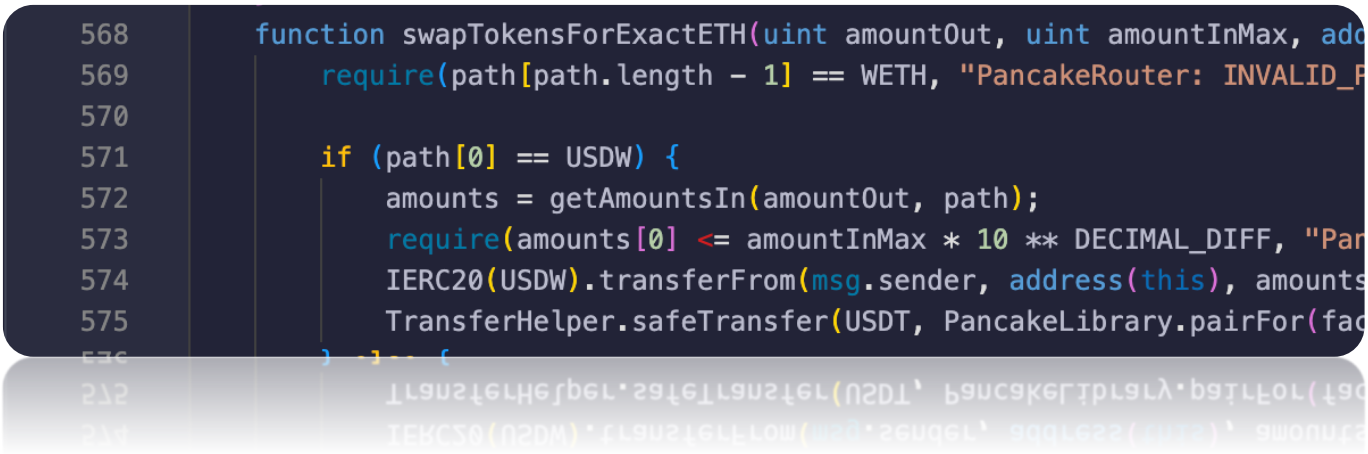
```
function swapTokensForExactTokens(uint amountOut, uint amountInMax, address[] c

    if (path[0] == USDW) {
        amounts = getAmountsIn(amountOut, path);
        IERC20(path[0]).transferFrom(msg.sender, address(this), amounts[0] / (10
        if (path[path.length - 1] == USDT) {
            IERC20(USDT).transfer(to, amounts[0]);
        } else {
            require(amounts[0] <= amountInMax * (10 ** DECIMAL_DIFF), "PancakeRo
            TransferHelper.safeTransfer(USDT, PancakeLibrary.pairFor(factory, US
            _swap(amounts, path, to);
        }
        }
```

- Here smart contract is checking that path is valid, sufficient output amount available in PancakeRounter, and transfer method of WETH contract is successfully called.

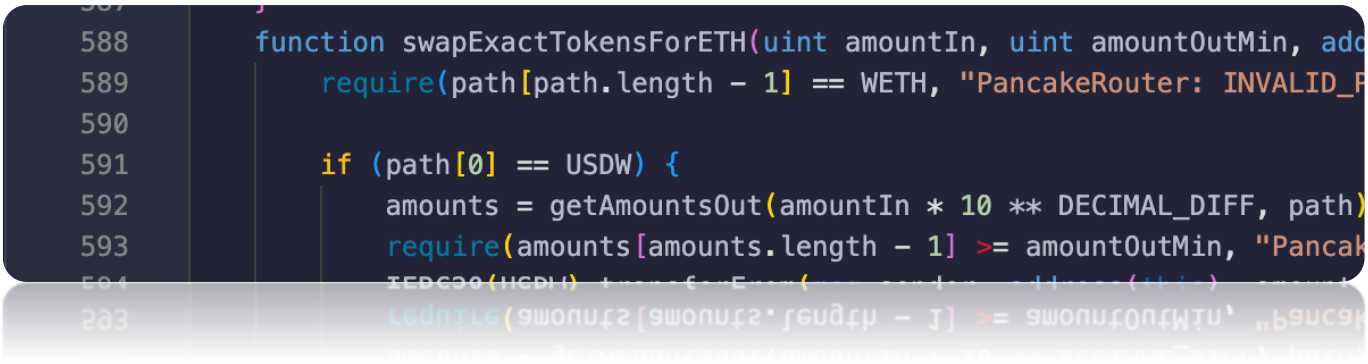
```
function swapExactETHForTokens(uint amountOutMin, address[] calldata path, addr
    require(path[0] == WETH, "PancakeRouter: INVALID_PATH");
    (uint amountInX, uint amountOutMinX, ) = countFees(msg.value, amountOutMin)
    amounts = getAmountsOut(amountInX, path);
    require(amounts[amounts.length - 1] >= amountOutMinX, "PancakeRouter: INSUF
    IWETH(WETH).deposit{value: amounts[0]}();
```

- Here smart contract is checking that path is valid, and sufficient input amount available in PancakeRounter,



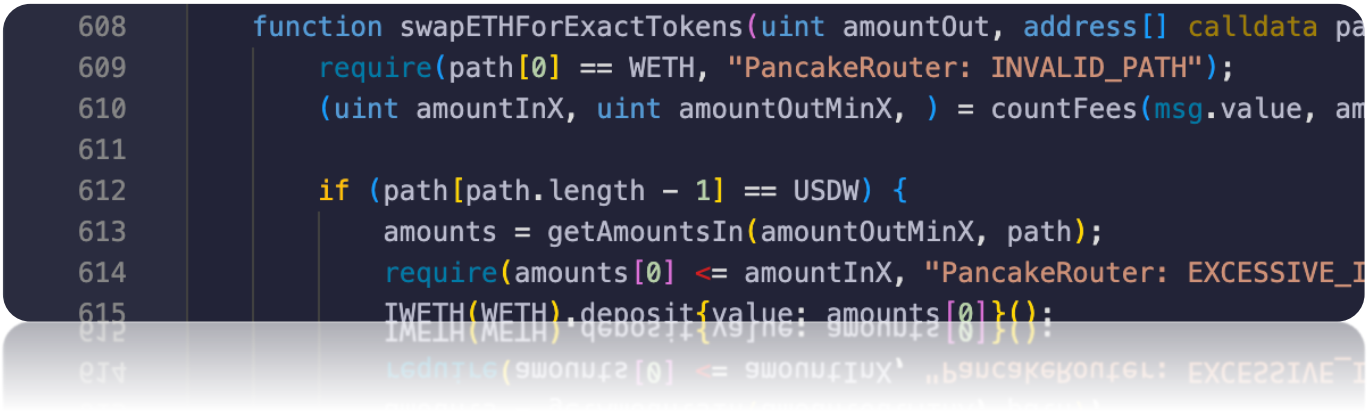
```
568     function swapTokensForExactETH(uint amountOut, uint amountInMax, add
569         require(path[path.length - 1] == WETH, "PancakeRouter: INVALID_P
570
571         if (path[0] == USDW) {
572             amounts = getAmountsIn(amountOut, path);
573             require(amounts[0] <= amountInMax * 10 ** DECIMAL_DIFF, "Par
574             IERC20(USDW).transferFrom(msg.sender, address(this), amounts
575             TransferHelper.safeTransfer(USDT, PancakeLibrary.pairFor(fac
```

- Here smart contract is checking that path is valid, and sufficient output amount available in PancakeRounter,



```
588     function swapExactTokensForETH(uint amountIn, uint amountOutMin, add
589         require(path[path.length - 1] == WETH, "PancakeRouter: INVALID_P
590
591         if (path[0] == USDW) {
592             amounts = getAmountsOut(amountIn * 10 ** DECIMAL_DIFF, path)
593             require(amounts[amounts.length - 1] >= amountOutMin, "Pancak
```

- Here smart contract is checking that path is valid, and sufficient input amount available in PancakeRounter,



```
608     function swapETHForExactTokens(uint amountOut, address[] calldata pa
609         require(path[0] == WETH, "PancakeRouter: INVALID_PATH");
610         (uint amountInX, uint amountOutMinX, ) = countFees(msg.value, am
611
612         if (path[path.length - 1] == USDW) {
613             amounts = getAmountsIn(amountOutMinX, path);
614             require(amounts[0] <= amountInX, "PancakeRouter: EXCESSIVE_I
615             IWETH(WETH).deposit{value: amounts[0]}();
```

- Here smart contract is checking that _tokenAddress is valid and proper.

```
672     function setRewardToken(address _tokenAddress) public adminOnly {
673         require(_tokenAddress != address(0), "invaid address found");
674         REWARDTOKEN = IERC20(_tokenAddress);
675     }
```

- Here smart contract is checking that _usdt is valid and proper.

```
712     function setUsdtAddress(address _usdt) public adminOnly{
713         require(_usdt != address(0) , "not valid");
714         USDT = _usdt;
715     }
```

- Here smart contract is checking that _usdw is valid and proper.

```
717     function setUsdwAddress(address _usdw) public adminOnly{
718         require(_usdw != address(0) , "not valid");
719         USDW = _usdw;
720     }
```

# 7. Critical vulnerabilities in code

- **No Critical vulnerabilities found**

# 8. Medium vulnerabilities in code

- **No Medium vulnerabilities found**

# 9. Low vulnerabilities in code

## 9.1. Suggestions to add code validations:-

=> You have implemented required validation in contract.
=> There are some places where you can improve validation and security of your code.
=> These are all just suggestion it is not bug.

- **Function: - addLiquidityETH**

```
401     function addLiquidityETH(address token, uint amountTokenDesired, uin
402         (amountToken, amountETH) = _addLiquidity(token, WETH, amountToke
403         address pair = PancakeLibrary.pairFor(factory, token, WETH);
404         TransferHelper.safeTransferFrom(token, msg.sender, pair, amountT
405         IWETH(WETH).deposit{value: amountETH}();
```

  - Here in addLiquidityETH function smart contract can check that deposit method of WETH contract is successfully called.

- **Function: - removeLiquidityETH**

```
422     function removeLiquidityETH(address token, uint liquidity, uint amou
423         (amountToken, amountETH) = removeLiquidity(token, WETH, liquidit
424         TransferHelper.safeTransfer(token, to, amountToken);
425         IWETH(WETH).withdraw(amountETH);
```

  - Here in removeLiquidityETH function smart contract can check that withdraw method of WETH contract is successfully called.

- **Function: - removeLiquidityETHSupportingFeeOnTransferTokens**

```
442     function removeLiquidityETHSupportingFeeOnTransferTokens(address tok
443         uint amountToken;
444         (amountToken, amountETH) = removeLiquidity(token, WETH, liquidit
445         TransferHelper.safeTransfer(token, to, amountToken);
446         IWETH(WETH).withdraw(amountETH);
```

  - Here in removeLiquidityETHSupportingFeeOnTransferTokens function smart contract can check that withdraw method of WETH contract is successfully called.

- **Function: - swapExactTokensForTokens**

```
485     function swapExactTokensForTokens(uint amountIn, uint amountOutMin,
486         (uint amountInX, uint amountOutMinX, uint fees) = countoFees(amo
487         if (path[0] == USDW) {
488             IERC20(path[0]).transferFrom(msg.sender, address(this), amou
489             if (path[path.length - 1] == USDT) {
490                 IERC20(USDT).transfer(to, amountInX * 10 ** DECIMAL_DIFF
```

  - Here in swapExactTokensForTokens function smart contract can check that transfer method of USDT contract is successfully called.

- **Function: - swapTokensForExactTokens**

```
517     function swapTokensForExactTokens(uint amountOut, uint amountInMax,
518
519
520         if (path[0] == USDW) {
521             amounts = getAmountsIn(amountOut, path);
522             IERC20(path[0]).transferFrom(msg.sender, address(this), amou
523             if (path[path.length - 1] == USDT) {
524                 IERC20(USDT).transfer(to, amounts[0]);
```

  - Here in swapTokensForExactTokens function smart contract can check that transfer method of USDT, USDW contracts are successfully called.

- **Function: - swapExactETHForTokens**

```
551     function swapExactETHForTokens(uint amountOutMin, address[] calldata
552         require(path[0] == WETH, "PancakeRouter: INVALID_PATH");
553         (uint amountInX, uint amountOutMinX, ) = countFees(msg.value, am
554         amounts = getAmountsOut(amountInX, path);
555         require(amounts[amounts.length - 1] >= amountOutMinX, "PancakeRo
556         IWETH(WETH).deposit{value: amounts[0]}();
```

  - Here in swapExactETHForTokens function smart contract can check that transfer method of WETH contract is successfully called.

- **Function: - setadminswallet**

```
650     function setadminsWallet(address _adminWallet) external adminOnly {
651         adminwallet = payable(_adminWallet);
652     }
```

  - Here in setadminswallet function smart contract can have check that _adminWallet address is valid and proper.

- **Function: - withdrawBNB**

```
654     function withdrawBNB() external adminOnly {
655         adminwallet.transfer(address(this).balance);
656     }
```

  - Here in withdrawBNB function smart contract can have check that transfer to adminwallet is successfully done.

- **Function: - transferAnyERC20Token**

```
658     function transferAnyERC20Token(address tokenAddress , uint256 balanc
659         IERC20(tokenAddress).transfer(adminwallet, balance);
660     }
```

  - Here in transferAnyERC20Token function smart contract can have check that transfer method of tokenAddress contract is successfully done.

# 10. Summary

- **Number of problems in the smart contract as per severity level**

| Critical | Medium | Low |
|---|---|---|
| 0 | 0 | 9 |

According to the assessment, the smart contract code is well secured. The code is written with all validation and all security is implemented. Code is performing well and there is no way to steal funds from this contract.

- **Good Point:** Code performance and quality are good. All kind of necessary validation added into smart contract and all validations are working as excepted.

- **Suggestions:** Please try to implement suggested code validations.