Audit report of PAYC

Prepared By:- Kishan Patel
Prepared On:- 25/05/2022

Prepared for: PAYC Finance

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 PAYC Finance(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 25/05/2022 – 29/05/2022.

The project has 1 file. It contains approx 1120 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	PAYC FINANCE
Token Symbol	PAYC
Platform	Binance Smart Chain
Order Started Date	25/05/2022
Order Completed Date	29/05/2022

4. List of attacks checked

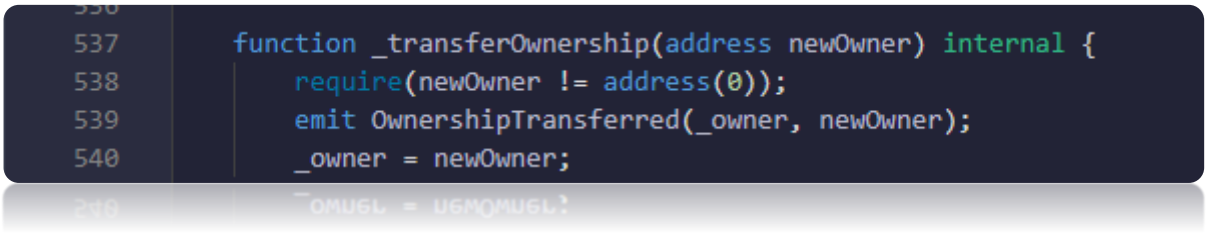
- Over and under flows
- Short address attack
- Visibility & Delegate call
- Reentrancy / TheDAO hack
- Forcing BSC to a contract
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style 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

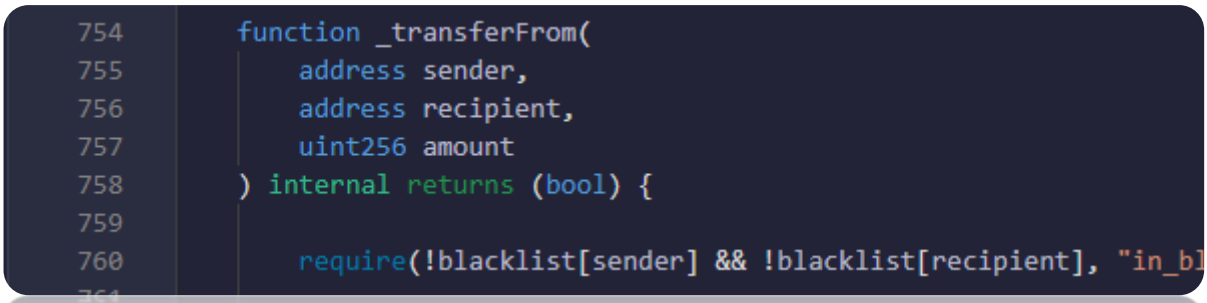
6. Good things in code

- **Good required condition in functions:-**
  - Here you are checking that newOwner address is valid and proper.



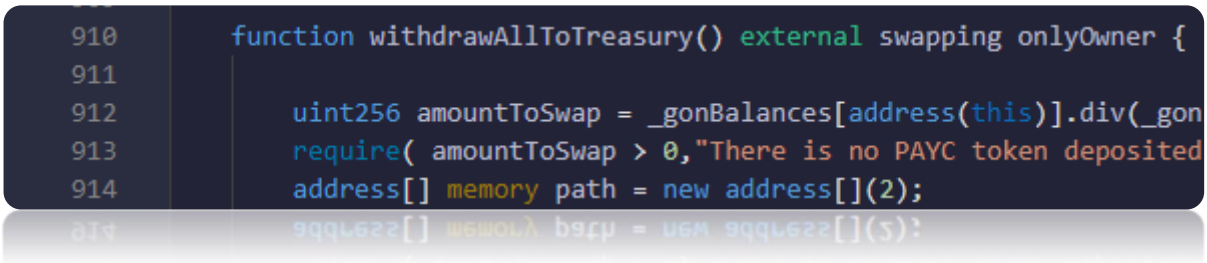
```
537    function _transferOwnership(address newOwner) internal {
538        require(newOwner != address(0));
539        emit OwnershipTransferred(_owner, newOwner);
540        _owner = newOwner;
```

  - Here you are checking that sender or recipient is not blacklisted.



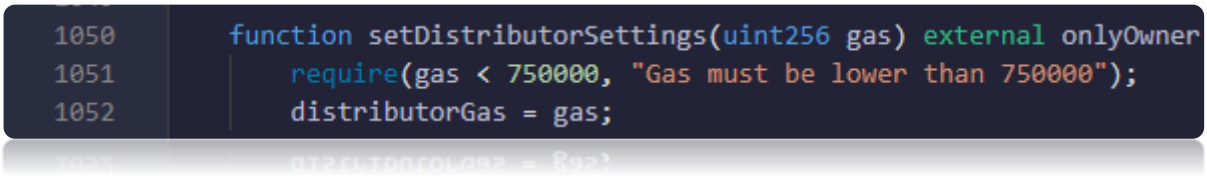
```
754    function _transferFrom(
755        address sender,
756        address recipient,
757        uint256 amount
758    ) internal returns (bool) {
759
760        require(!blacklist[sender] && !blacklist[recipient], "in_bl
```

  - Here you are checking that amountToSwap is bigger than 0.



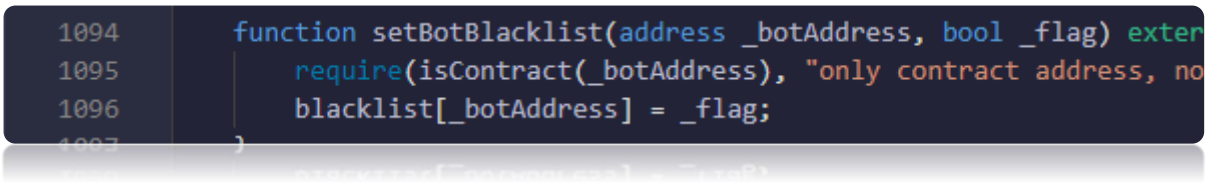
```
910    function withdrawAllToTreasury() external swapping onlyOwner {
911
912        uint256 amountToSwap = _gonBalances[address(this)].div(_gon
913        require( amountToSwap > 0,"There is no PAYC token deposited
914        address[] memory path = new address[](2);
```

  - Here you are checking that gas price is lower than 750000.



```
1050   function setDistributorSettings(uint256 gas) external onlyOwner
1051       require(gas < 750000, "Gas must be lower than 750000");
1052       distributorGas = gas;
```

- Here you are checking that _botAddress is contract address or not.



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. Compiler version is not fixed:-

=> In this file you have put “pragma solidity ^0.7.4;” which is not a good way to define compiler version.

=> Solidity source files indicate the versions of the compiler they can be compiled with. Pragma solidity ^0.7.4; // bad: compiles 0.7.4 and above pragma solidity 0.7.4; //good: compiles 0 .7.4 only

=> If you put(>=) symbol then you are able to get compiler version 0.7.4 and above. But if you don’t use(^/>=) symbol then you are able to use only 0.7.4 version. And if there are some changes come in the compiler and you use the old version then some issues may come at deploy time.

9.2. Suggestions to add code validations:-

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

- **Hardcoded owner address**

```
511     constructor() {
512         _owner = 0x0Be0cAA432570Ce1378b31b3bB00C7f18972DbE4;
513     }
```

  - Here in Ownable contract you have defined _owner address as hardcoded please try to make it msg.sender (deployer of contract).

- **Function: - swapBack**

```
898         (bool success, ) = payable(treasuryReceiver).call{
899             value: amountETHToTreasuryAndPAYC.mul(treasuryFee).div(
900                 treasuryFee.add(PAYCDividendFee)
901             ),
902             gas: 30000
903         }("");
```

  - Here in swapBack function you have set gas value hardcoded. I suggest to not use hard coded value because in blockchain if we have too many transaction then your transaction will be get delayed or cancel.

  - Please check that success value is correct or not.

Function: - approve

```
1021    function approve(address spender, uint256 value)
1022        external
1023        override
1024        returns (bool)
1025    {
1026        _allowedFragments[msg.sender][spender] = value;
1027        emit Approval(msg.sender, spender, value);
1028        return true;
1029    }
```

- Here in approve function you can check that balance of msg.sender is bigger than value or not.

Function: - setWhitelist

```
1090    function setWhitelist(address _addr) external onlyOwner {
1091        _isFeeExempt[_addr] = true;
1092    }
```

- Here in setWhitelist function you can check that true status is not already set for _addr.

Function: - setBotBlacklist

```
1094    function setBotBlacklist(address _botAddress, bool _flag) exter
1095        require(isContract(_botAddress), "only contract address, no
1096        blacklist[_botAddress] = _flag;
```

- Here in setBotBlacklist function you can check that flag value is not already set for_botAddress.

10. Summary

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

Critical	Medium	Low
0	**0**	**6**

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. Address validation and value validation is done properly

- **Suggestions:** Please try to use static solidity version, and try to implement suggested code validations.