SMART CONTRACT AUDIT REPORT

For

OpenTheta.io

Prepared By: Kishan Patel

Prepared For: OpenTheta.io

Prepared on: 11/11/2021

Table of Content

• Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

• Overview of the audit

The project has 2 files. It contains approx 500 lines of Solidity code. All the functions and state variables are well commented using the natspec documentation, but that does not create any vulnerability.

• Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

• Over and under flows

An overflow happens when the limit of the type variable uint256, 2 ** 256, is exceeded. What happens is that the value resets to zero instead of incrementing more. On the other hand, an underflow happens when you try to subtract 0 minus a number bigger than 0. For example, if you subtract 0 - 1 the result will be = 2 ** 256 instead of -1. This is quite dangerous.

This contract **does** check for overflows and underflows by using OpenZeppelin's SafeMath to mitigate this attack, but all the functions have strong validations, which prevented this attack.

• Short address attack

If the token contract has enough amount of tokens and the buy function doesn't check the length of the address of the sender, the ethereum's virtual machine will just add zeros to the transaction until the address is complete.

Although this contract **is not vulnerable** to this attack, but there are some point where users can mess themselves due to this (Please see below). It is highly recommended to call functions after checking validity of the address.

• Visibility & Delegate call

It is also known as, The Parity Hack, which occurs while misuse of Delegate call.

No such issues found in this smart contract and visibility also properly addressed. There are some places where there is no visibility defined. Smart Contract will assume "Public" visibility if there is no visibility defined. It is good practice to explicitly define the visibility, but again, the contract is not prone to any vulnerability due to this in this case.

• Reentrancy / TheDAO hack

Reentrancy occurs in this case: any interaction from a contract (A) with another contract (B) and any transfer of ethereum hands over control to that contract (B).

This makes it possible for B to call back into A before this interaction is completed.

Use of "require" function in this smart contract mitigated this vulnerability.

- **Forcing Ethereum to a contract**

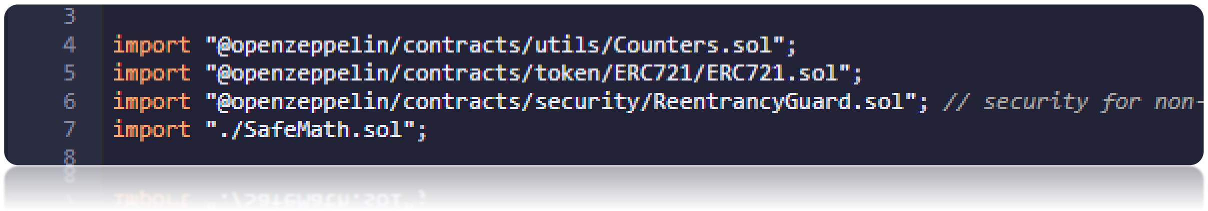
While implementing "selfdestruct" in smart contract, it sends all the ethereum to the target address. Now, if the target address is a contract address, then the fallback function of target contract does not get called. And thus Hacker can bypass the "Required" conditions. Here, the Smart Contract's balance has never been used as guard, which mitigated this vulnerability.

• Good things in smart contract

• SafeMath library:-

Filename: - NFTMarket.sol

- You are using SafeMath library it is a good thing. This protects you from underflow and overflow attacks.

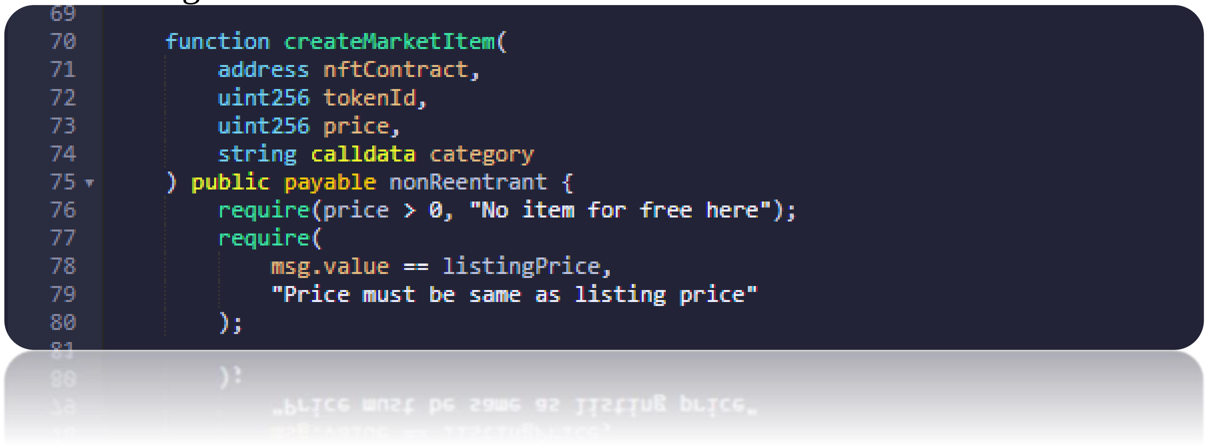


```
import "@openzeppelin/contracts/utils/Counters.sol";
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol"; // security for non-
import "./SafeMath.sol";
```

• Good required condition in functions:-

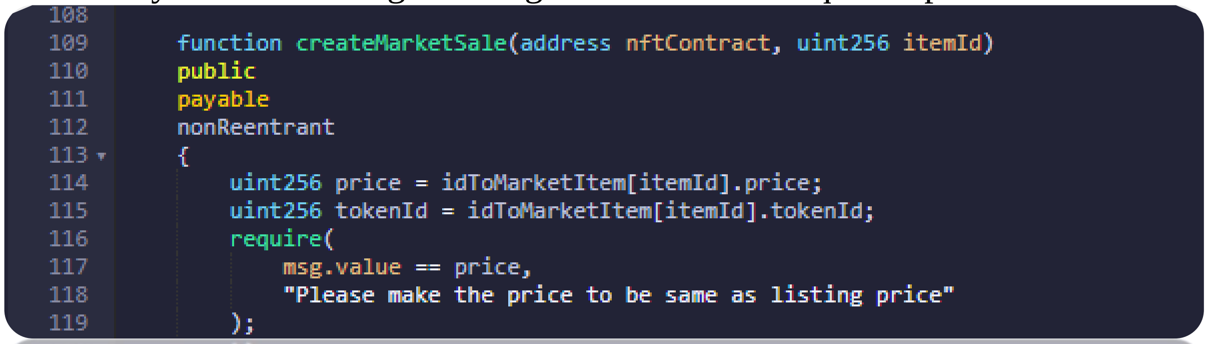
Filename: - NFTMarket.sol

- Here you are checking that price is bigger than 0 and msg.value should be equal to listingPrice.



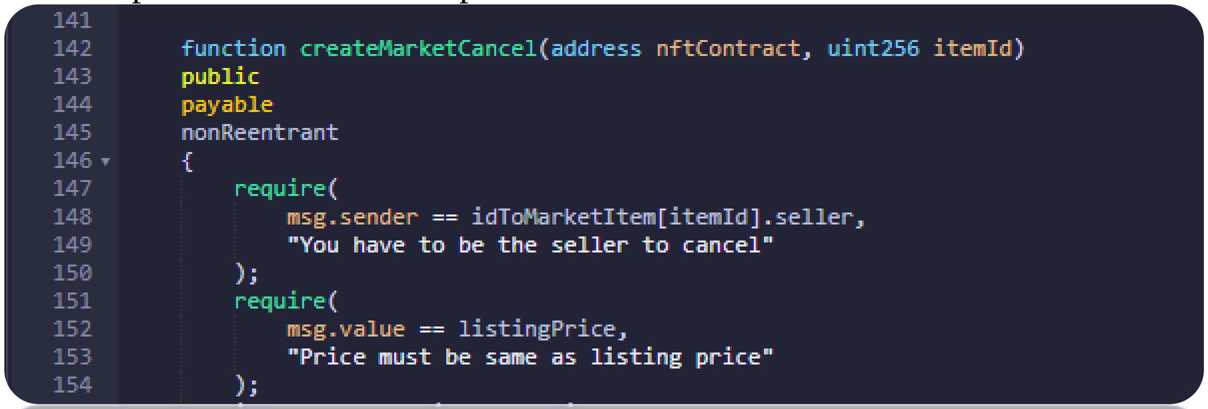
```
    function createMarketItem(
        address nftContract,
        uint256 tokenId,
        uint256 price,
        string calldata category
    ) public payable nonReentrant {
        require(price > 0, "No item for free here");
        require(
            msg.value == listingPrice,
            "Price must be same as listing price"
        );
```

- Here you are checking that msg.value should be equal to price.



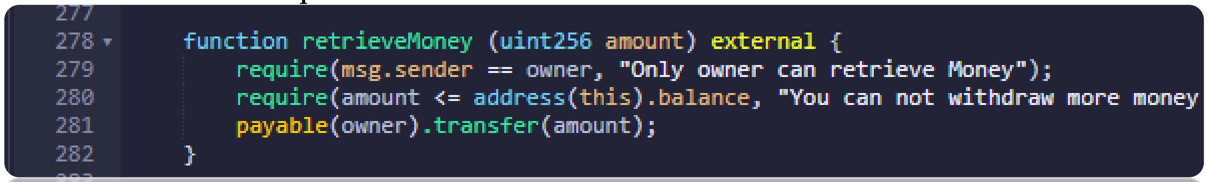
```
function createMarketSale(address nftContract, uint256 itemId)
public
payable
nonReentrant
{
    uint256 price = idToMarketItem[itemId].price;
    uint256 tokenId = idToMarketItem[itemId].tokenId;
    require(
        msg.value == price,
        "Please make the price to be same as listing price"
    );
```

- Here you are checking that msg.sender is should be seller for cancelling market and price should be listed price.



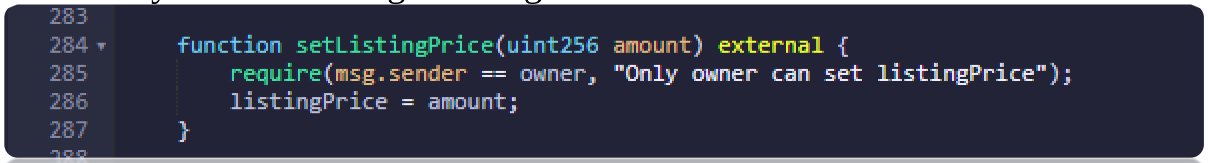
```
function createMarketCancel(address nftContract, uint256 itemId)
public
payable
nonReentrant
{
    require(
        msg.sender == idToMarketItem[itemId].seller,
        "You have to be the seller to cancel"
    );
    require(
        msg.value == listingPrice,
        "Price must be same as listing price"
    );
```

- Here you are checking that msg.sender is owner of contract and amount should be smaller or equal to balance of this contract.



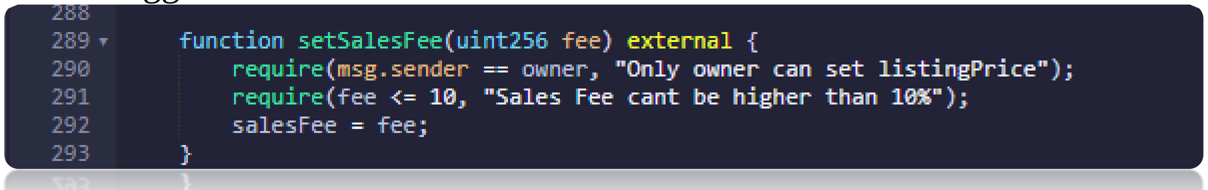
```
function retrieveMoney (uint256 amount) external {
    require(msg.sender == owner, "Only owner can retrieve Money");
    require(amount <= address(this).balance, "You can not withdraw more money
    payable(owner).transfer(amount);
}
```

- Here you are checking that msg.sender is owner of contract.



```
function setListingPrice(uint256 amount) external {
    require(msg.sender == owner, "Only owner can set listingPrice");
    listingPrice = amount;
}
```

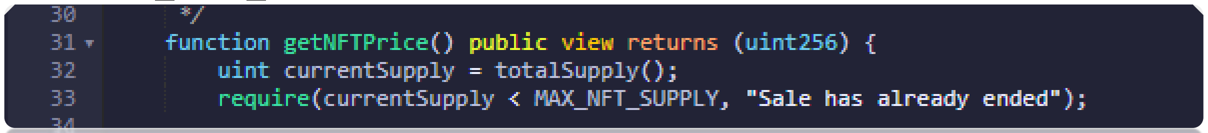
- Here you are checking that msg.sender is owner of contract and fee should not be bigger than 10.



```
    function setSalesFee(uint256 fee) external {
        require(msg.sender == owner, "Only owner can set listingPrice");
        require(fee <= 10, "Sales Fee cant be higher than 10%");
        salesFee = fee;
    }
```

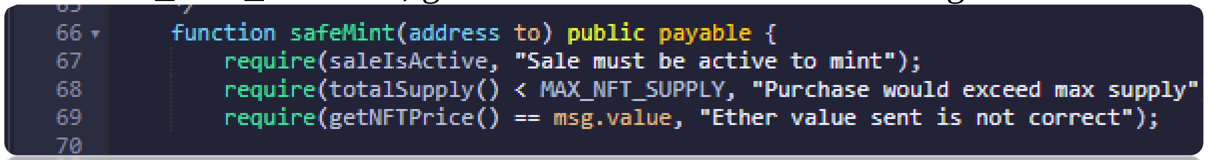
Filename: - NFTtoken.sol

- Here you are checking that currentSupply should not be bigger than MAX_NET_SUPPLY.



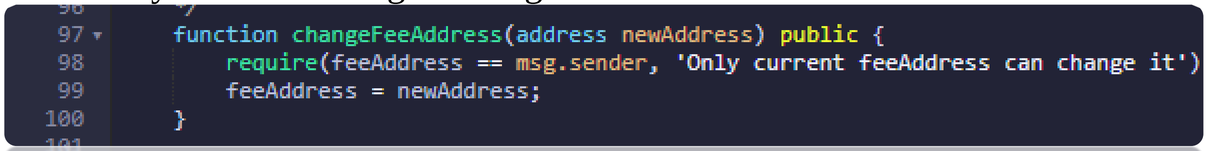
```
    */
    function getNFTPrice() public view returns (uint256) {
        uint currentSupply = totalSupply();
        require(currentSupply < MAX_NFT_SUPPLY, "Sale has already ended");
```

- Here you are checking that saleIsActive, totalSupply should not be bigger than MAX_NFT_SUPPLY, getNFTPrice should be same as msg.value.



```
    function safeMint(address to) public payable {
        require(saleIsActive, "Sale must be active to mint");
        require(totalSupply() < MAX_NFT_SUPPLY, "Purchase would exceed max supply"
        require(getNFTPrice() == msg.value, "Ether value sent is not correct");
```

- Here you are checking that msg.sender should be same as feeAddress.



```
    function changeFeeAddress(address newAddress) public {
        require(feeAddress == msg.sender, 'Only current feeAddress can change it')
        feeAddress = newAddress;
    }
```

- ## Critical vulnerabilities found in the contract

=> No Critial vulnerabilities found

- ## Medium vulnerabilities found in the contract

=> No Medium vulnerabilities found

- **Low severity vulnerabilities found**

  - **7.1: Compiler version is not fixed:-**

Filename: - All files

=> In this file you have put “pragma solidity ^0.8.4;” 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 ^0.8.4; // bad: compiles 0.8.4 and above pragma solidity 0.8.4; //good: compiles 0.8.4 only

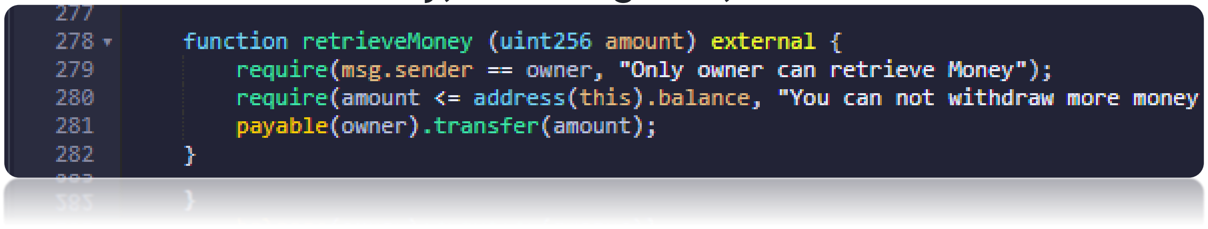
=> If you put(>=) symbol then you are able to get compiler version 0.8.4 and above. But if you don’t use(^/>=) symbol then you are able to use only 0.8.4 version. And if there are some changes come in the compiler and you use the old version then some issues may come at deploy time.

7.2: Suggestions to add validations:-

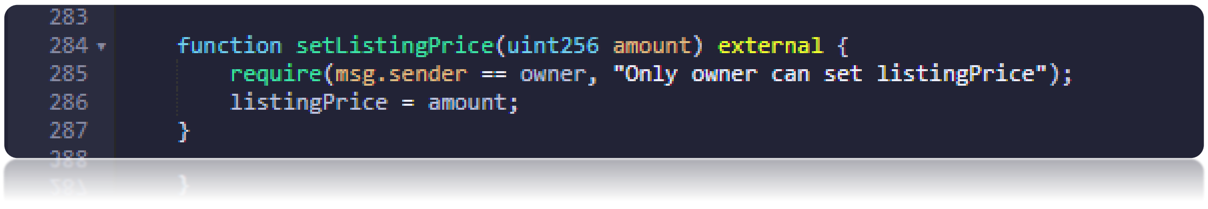
=> You have implemented required validation in contract.
=> There are some place where you can improve validation and security of your code.
=> These are all just suggestion it is not bug.

Filename: - NFTMarket.sol

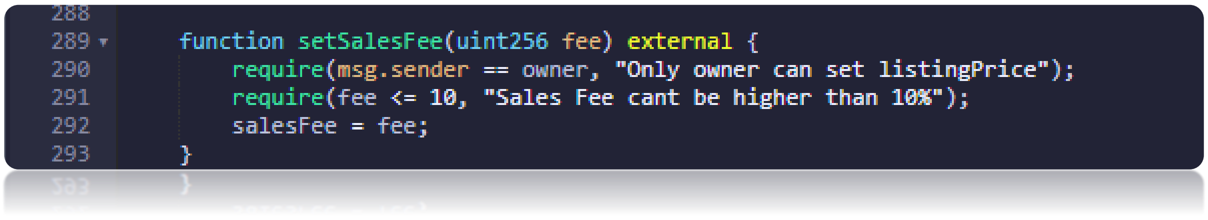
- **Function: - retrieveMoney, setListingPrice, setSalesFee**



```
    function retrieveMoney (uint256 amount) external {
        require(msg.sender == owner, "Only owner can retrieve Money");
        require(amount <= address(this).balance, "You can not withdraw more money
        payable(owner).transfer(amount);
    }
```



```
    function setListingPrice(uint256 amount) external {
        require(msg.sender == owner, "Only owner can set listingPrice");
        listingPrice = amount;
    }
```

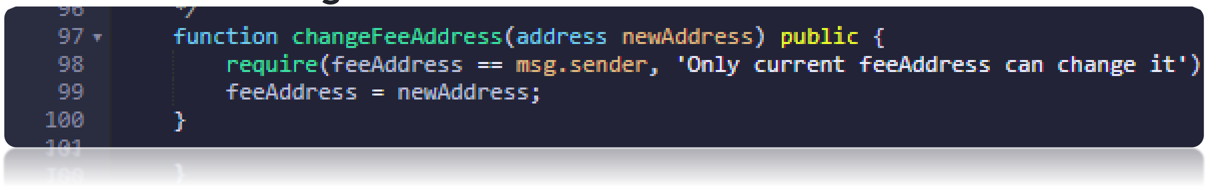


```
    function setSalesFee(uint256 fee) external {
        require(msg.sender == owner, "Only owner can set listingPrice");
        require(fee <= 10, "Sales Fee cant be higher than 10%");
        salesFee = fee;
    }
```

- Here you are defining owner value at contract generation time but it would be helpful to use Ownable library because with this you can change owner of contract in future. Otherwise currently you don’t have way to change owner of contract.

Filename: - NFTtoken.sol

- **Function: - changeFeeAddress**



```
    function changeFeeAddress(address newAddress) public {
        require(feeAddress == msg.sender, 'Only current feeAddress can change it')
        feeAddress = newAddress;
    }
```

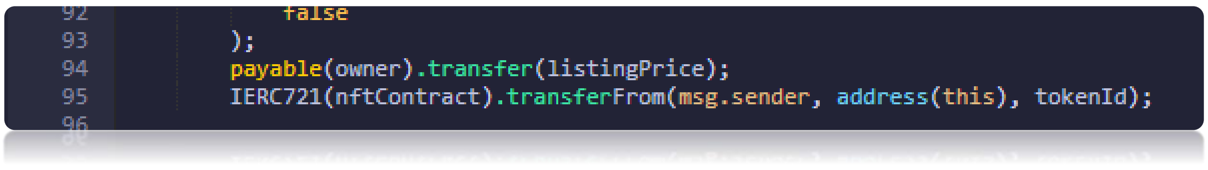
  - You have feeAddress value and it is assigned while deploying contract. Here it would be great if only owner of the contract can change this value because feeAddress should be any address and you cannot give other address permission to change data in your contract.

7.3: Uncheck return response of transfer and transferFrom method:-

=> I have found that you are transferring fund to address using a transfer and transferFrom methods.
=> It is always good to check the return value or response from a function call.
=> Here are some functions where you forgot to check a response.
=> I suggest, if there is a possibility then please check the response.

Filename: - NFTMarket.sol

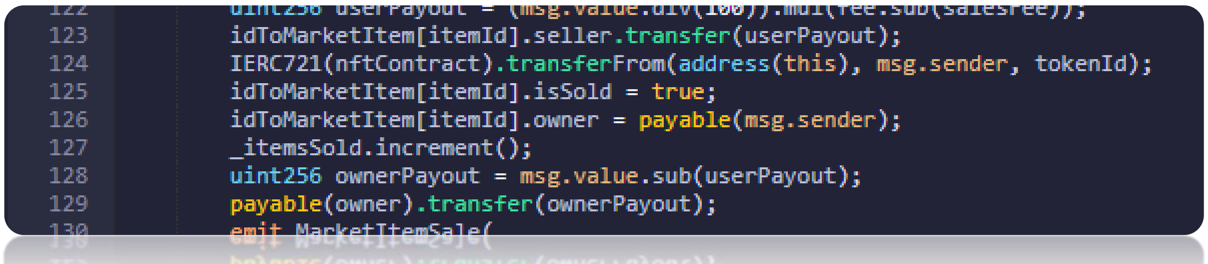
- **Function: - createMarketItem**



```
                false
            );
            payable(owner).transfer(listingPrice);
            IERC721(nftContract).transferFrom(msg.sender, address(this), tokenId);
```

  - Here you are calling transfer and trnasferFrom method 1 time. It is good to check that the transfer is successfully done or not.

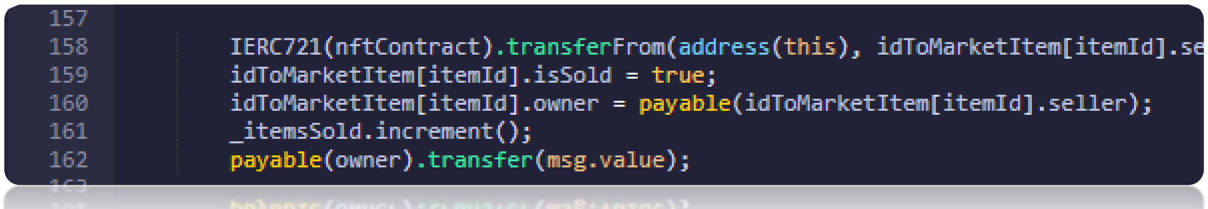
- **Function: - createMarketSale**



```
        uint256 userPayout = (msg.value.div(100)).mul(fee.sub(salesFee));
        idToMarketItem[itemId].seller.transfer(userPayout);
        IERC721(nftContract).transferFrom(address(this), msg.sender, tokenId);
        idToMarketItem[itemId].isSold = true;
        idToMarketItem[itemId].owner = payable(msg.sender);
        _itemsSold.increment();
        uint256 ownerPayout = msg.value.sub(userPayout);
        payable(owner).transfer(ownerPayout);
        emit MarketItemSale(
```

  - Here you are calling transfer and trnasferFrom method 3 times. It is good to check that the transfer is successfully done or not.

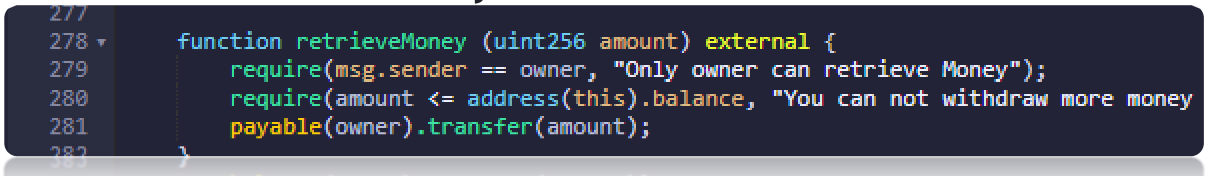
- **Function: - createMarketCancel**



```
        IERC721(nftContract).transferFrom(address(this), idToMarketItem[itemId].se
        idToMarketItem[itemId].isSold = true;
        idToMarketItem[itemId].owner = payable(idToMarketItem[itemId].seller);
        _itemsSold.increment();
        payable(owner).transfer(msg.value);
```

  - Here you are calling transfer and trnasferFrom method 1 times. It is good to check that the transfer is successfully done or not.

- **Function: - retrieveMoney**

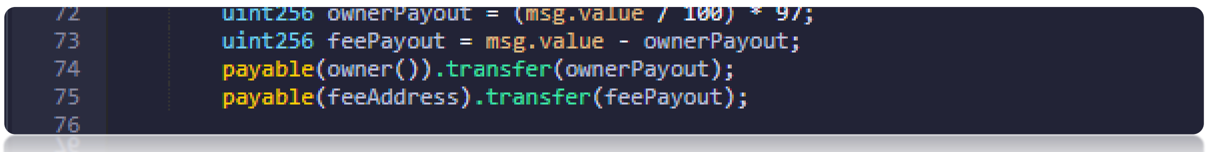


```
    function retrieveMoney (uint256 amount) external {
        require(msg.sender == owner, "Only owner can retrieve Money");
        require(amount <= address(this).balance, "You can not withdraw more money
        payable(owner).transfer(amount);
    }
```

  - Here you are calling transfer method 1 time. It is good to check that the transfer is successfully done or not.

Filename: - NFTtoken.sol

- **Function: - safeMint**



```
        uint256 ownerPayout = (msg.value / 100) * 97;
        uint256 feePayout = msg.value - ownerPayout;
        payable(owner()).transfer(ownerPayout);
        payable(feeAddress).transfer(feePayout);
```

  - Here you are calling transfer method 2 times. It is good to check that the transfer is successfully done or not.

• Summary of the Audit

Overall, the code is written with all validation and all security

is implemented. Code is performs well and there is no way to

steal fund from this contract.

Please try to check the address and value of token externally before sending to the solidity code.

Our final recommendation would be to pay more attention to the visibility of the functions , hardcoded address and mapping since it’s quite important to define who’s supposed to executed the functions and to follow best practices regarding the use of assert, require etc. (which you are doing ;)).

- **Good Point:** Code performance and quality is good. Address validation and value validation is done properly.
- **Suggestions:** Please use the static version of solidity, add suggested code validation and try to check return response of transfer and transferFrom method.