HALBORN CTF - SOLDITY AUDIT REPORT

By: Pankaj Jagtap

Table of Contents

Summary

Report for Halborn CTF Solidity Smart contracts.

Contracts in Scope:

1. HalbornToken.sol
2. HalbornNFT.sol
3. HalbornLoans.sol

Project Summary:

Project Name	_ Smart Contract
Description	The Audit report of Halborn CTF smart contracts with Static and Manual analysis.
Platform	EVM Compatible Chains
Smart Contract Language	Solidity
Codebase	
Contract Address	

Audit Summary

Delivery Date	26 - 03 - 2024
Audit Methodology	Static analysis, Manual Review

Vulnerability Summary

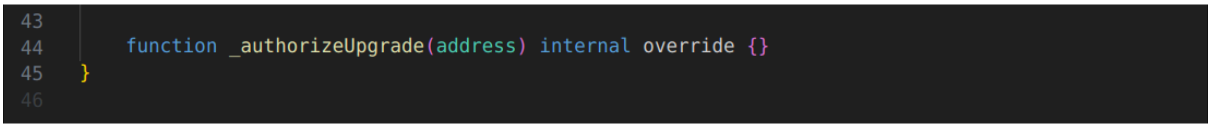
Total Issues	11
High	11
Medium	0
Low	0

Findings

1. HalbornToken.sol

1) Insufficient Access Control

Issue



```
    function _authorizeUpgrade(address) internal override {}
}
```

The function authorize upgrade has insufficient access control.

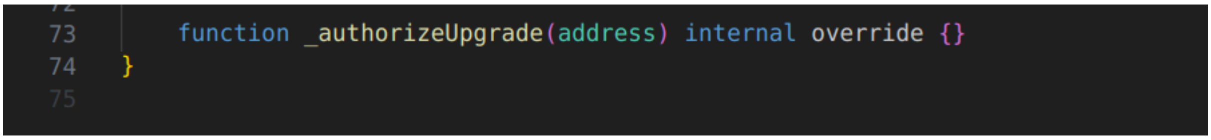
Recommendation

Use onlyOwner modifier for the function

2. HalbornNFT.sol

1) Insufficient Access Control

Issue



```
    function _authorizeUpgrade(address) internal override {}
}
```

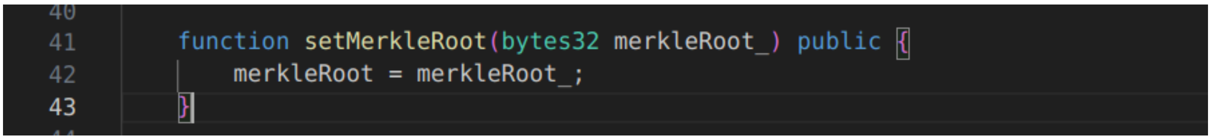
The function authorize upgrade has insufficient access control.

Recommendation

Use onlyOwner modifier for the function

2) Insufficient Access Control

Issue



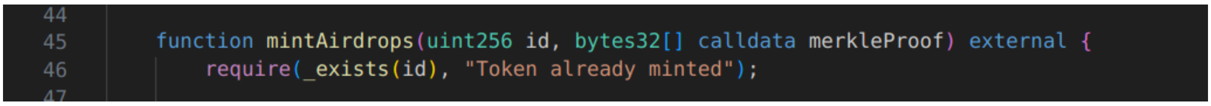
```
    function setMerkleRoot(bytes32 merkleRoot_) public {
        merkleRoot = merkleRoot_;
    }
```

The function setMerkleRoot has insufficient access control.

Recommendation

Use onlyOwner modifier for the function

3) **Inaccurate _exists check**



```
    function mintAirdrops(uint256 id, bytes32[] calldata merkleProof) external {
        require(_exists(id), "Token already minted");
```

Issue

The require statement wants to prevent minted tokens from being airdropped but the check is incorrect.

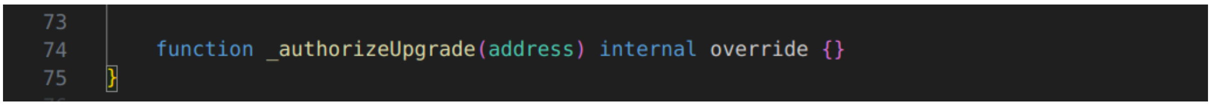
Recommendation

Apply *require(!_exists(id), "Token already minted");*

3. HalbornLoans.sol

1) Insufficient Access Control

Issue



```
    function _authorizeUpgrade(address) internal override {}
}
```

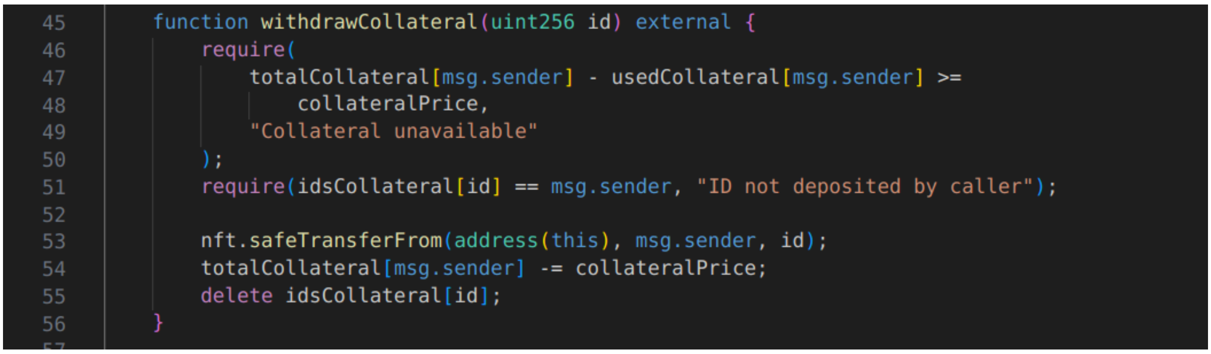
The function authorize upgrade has insufficient access control.

Recommendation

Use onlyOwner modifier for the function

2) Reentrancy in withdrawCollateral function

Issue



```
    function withdrawCollateral(uint256 id) external {
        require(
            totalCollateral[msg.sender] - usedCollateral[msg.sender] >=
                collateralPrice,
            "Collateral unavailable"
        );
        require(idsCollateral[id] == msg.sender, "ID not deposited by caller");

        nft.safeTransferFrom(address(this), msg.sender, id);
        totalCollateral[msg.sender] -= collateralPrice;
        delete idsCollateral[id];
    }
```

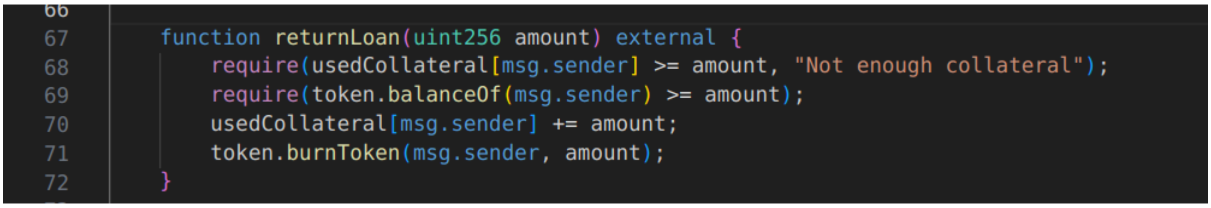
Reentrancy issue in withdrawCollateral function.

Recommendation

Apply nonReentrant modifier from Openzeppelin.

3) Incorrect usedCollateral calculation in returnLoan function

Issue



```
    function returnLoan(uint256 amount) external {
        require(usedCollateral[msg.sender] >= amount, "Not enough collateral");
        require(token.balanceOf(msg.sender) >= amount);
        usedCollateral[msg.sender] += amount;
        token.burnToken(msg.sender, amount);
    }
```

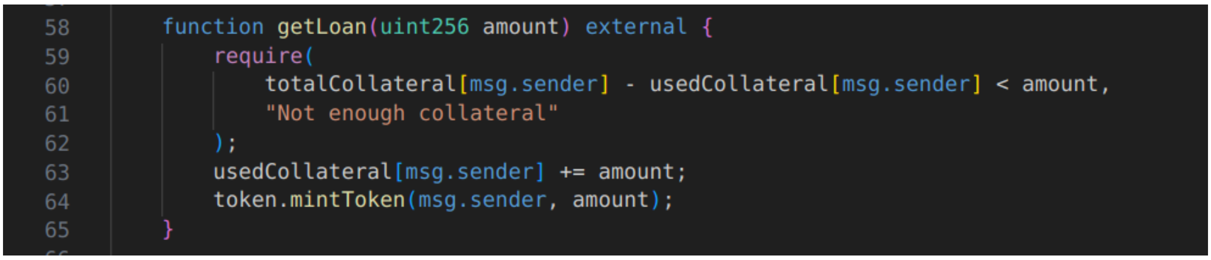
usedCollateral mapping is incremented for any user even when the user returns the loan using returnLoan function

Recommendation

Decrement the usedCollateral for that user.

4) Incorrect check for collateral availability in getLoan function

Issue



```
    function getLoan(uint256 amount) external {
        require(
            totalCollateral[msg.sender] - usedCollateral[msg.sender] < amount,
            "Not enough collateral"
        );
        usedCollateral[msg.sender] += amount;
        token.mintToken(msg.sender, amount);
    }
```

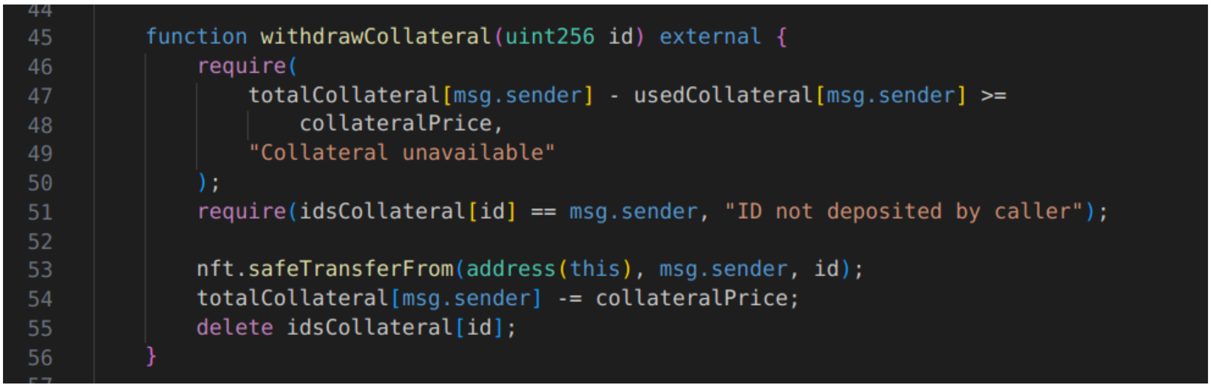
The require statement will always be true for any amount bigger than totalCollateral - usedCollateral

Recommendation

Change the < sign to >= sign.

5) usedCollateral is not decremented after withdrawing collateral

Issues



```
    function withdrawCollateral(uint256 id) external {
        require(
            totalCollateral[msg.sender] - usedCollateral[msg.sender] >=
                collateralPrice,
            "Collateral unavailable"
        );
        require(idsCollateral[id] == msg.sender, "ID not deposited by caller");

        nft.safeTransferFrom(address(this), msg.sender, id);
        totalCollateral[msg.sender] -= collateralPrice;
        delete idsCollateral[id];
    }
```

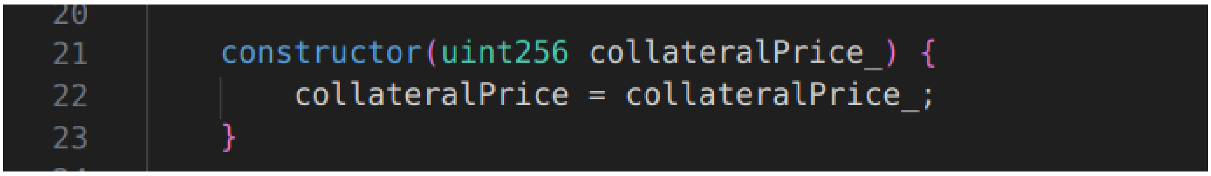
usedCollateral is not decremented after withdrawing collateral

Recommendation

Decrement the usedCollateral variable when user withdraws the collateral

6) Constructor for a UUPS Upgradeable Contract without Disable Initializer Check

Issue



```
    constructor(uint256 collateralPrice_) {
        collateralPrice = collateralPrice_;
    }
```

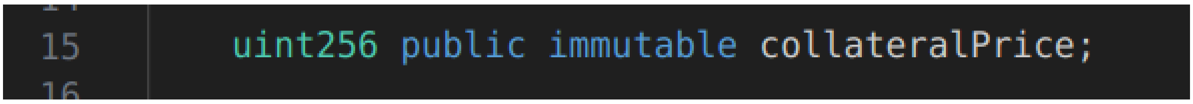
Constructor is being used in an upgradeable contract

Recommendation

Do not use the constructor. Use initialize functions to initialize the variables and add *_disableInitializers* function from Openzeppelin so that the new implementation is never initialized through the constructor.

7) Immutable variable in Upgradeable contract

Issue



```
    uint256 public immutable collateralPrice;
```

Immutable variable is used in an upgradeable contract

Recommendation

Do not use an immutable variable. Instead set up the value of the variable using the initialize function and never change it again.