MythX

REPORT 63CC15C5AA1300001A78BA3A

Created	Sat Jan 21 2023 16:41:41 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	62b1a8425ec4948f52c83856

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
65331abe-9f2b-4a37-9c8a-85cedd64186c	NFTInfo.sol	6

Analysis 65331abe-9f2b-4a37-9c8a-85cedd64186c

MythX

Started	Sat Jan 21 2023 16:41:44 GMT+0000 (Coordinated Universal Time)
Finished	Sat Jan 21 2023 18:07:18 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Remythx
Main Source File	NFTInfo.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	6

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.8.9"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

NFTInfo.sol

Locations

```
//SPDX-License-Identifier:MIT
pragma solidity ^0.8.9;


```

LOW

SWC-107

Write to persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

NFTInfo.sol

Locations

```
if (_pathno == 0) {
setARTinWrapper(4, dummy, _tokennumber, _holder); //Resets to default URI
artenabled[_tokennumber] = 0;
}
if (_pathno == 1) {
```

LOW

SWC-107

Write to persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`NFTInfo.sol`

Locations

```
if (_pathno == 1) {
setARTinWrapper(4, dummy, _tokennumber, _holder); //Art path 1
artenabled[_tokennumber] = 1;
}
if (_pathno == 2) {
```

LOW

SWC-107

Write to persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`NFTInfo.sol`

Locations

```
if (_pathno == 2) {
setARTinWrapper(4, dummy, _tokennumber, _holder); //Art path 2
artenabled[_tokennumber] = 2;
}
}
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`NFTInfo.sol`

Locations

```
"Not Auth(U)"
);
wrapper(wrapperaddress).setStringArtPaths(
_pathtype,
_path,
_tokenid,
_holder
);
}

```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`NFTInfo.sol`

Locations

```
"Not Auth!"
);
temp = wrapper(wrapperaddress).getArtApproval(_tokennumber, _holder); //requires the users approval to adjust the path!
require(temp == true, "Owner not approved!");
if (_pathno == 0) {
```

Source file

`NFTInfo.sol`

Locations

```
}

contract NFTInfo {
//Arrays///
uint[] private blockednfts; //Array to handle a blocked nfts
//Std Variables///
address public wrapperaddress; //Address of Wrapper Contract
address public ruggedproject = 0x4bCa2A290bf88bdf3fc6Ea25c377134aE1C7cFed; //Address of the Rugged Project
address public Owner;
address public upgradecontract; //Additional contract which will be allowed to manage the TOKEN URI's
uint private numwraps;
uint public numholders;
uint public numblocked;
///////Important Mappings///////
mapping(address => bool) internal wrapped; //Whether a holder has wrapped
mapping(address => bool) internal holder; //Whether they are a holder
mapping(uint => bool) internal blocked; //blocking due to mapping
mapping(uint => uint) internal artenabled; //Dynamic mapping of art path selection
mapping(address => bool) internal blockedaddresses; //Additional addresses to blacklist
///////Array for holders////////
address[] internal holderaddresses; //array to store the holders
////////////////////////////////

modifier onlyOwner() {
require(msg.sender == Owner);
_;
}

constructor() public {
Owner = msg.sender; //Owner of Contract
}

///Configure the important addresses for the contract
function configNBAddresses(uint option, address _address)
external
onlyOwner
{
if (option == 1) {
wrapperaddress = _address;
}
if (option == 2) {
ruggedproject = _address;
}
if (option == 3) {
upgradecontract = _address;
}
}

//Users to upgrade the TOKEN at a global level i.e Default URI
//The options are the following for _pathtype:
// 1) 0 = Default URI for all tokens
// 2) 1 = Art Path 1 -> Customizable for all tokens
// 3) 2 = Art Path 1 -> Customizable for all tokens
// 4) 3 = Specifies custom art for a single token
// 5) 4 = Sets token back to default URI (negates option 3)
function setARTinWrapper(
uint _pathtype,
string memory _path,
uint _tokenid,
address _holder
) public {
require(
msg.sender == Owner || msg.sender == upgradecontract,
"Not Auth(U)"
);
wrapper(wrapperaddress).setStringArtPaths(
_pathtype,
_path,
_tokenid,
_holder
);
}

//Obtain Art status for Token
function getArtStatus(uint _tokenid)public view returns(uint)
{
uint temp;
temp = artenabled[_tokenid];
return temp;
}

//This is slightly different from the above as this is used to set the ONLY the PATH for a token and not the Custom one (4)
//This is used when the token needs to be running a different path or reset back.
function setArtPath(
uint _tokennumber,
address _holder,
uint _pathno
) external {
bool temp;
string memory dummy = ""; //dummy string to pass in
require(
msg.sender == Owner || msg.sender == upgradecontract,
"Not Auth!"
);
temp = wrapper(wrapperaddress).getArtApproval(_tokennumber, _holder); //requires the users approval to adjust the path!
require(temp == true, "Owner not approved!");
if (_pathno == 0) {
setARTinWrapper(4, dummy, _tokennumber, _holder); //Resets to default URI
artenabled[_tokennumber] = 0;
}
if (_pathno == 1) {
setARTinWrapper(4, dummy, _tokennumber, _holder); //Art path 1
artenabled[_tokennumber] = 1;
}
if (_pathno == 2) {
setARTinWrapper(4, dummy, _tokennumber, _holder); //Art path 2
artenabled[_tokennumber] = 2;
}
```

```
}

//Function to Verify whether an NFT is blocked
function isBlockedNFT(uint _tokenID) external view returns (bool, uint256) {
bool temp;
address tempaddress;
temp = blocked[_tokenID]; //Is the block at Token Level?
if (temp == false) // If not at token level,lets verify at address level
{
tempaddress = ownerOfToken(_tokenID);
temp = blockedaddresses[tempaddress]; // returns Bool dependant on block at address level
}

return (temp, 0);
}

//Function to return whether they are a holder or not
function isHolder(address _address) external view returns (bool) {
bool temp;
if (holder[_address] == true) {
temp = true;
}
return temp;
}

//Manage the user status i.e wrap=holder, unwarp=not a holder
function manageHolderAddresses(bool status, address _holder) external {
require(
msg.sender == wrapperaddress || msg.sender == Owner,
"Not Oracle/Owner!"
);
if (status == true) {
//Add user to array!
(bool _isholder, ) = isHolderInArray(_holder);
if (!_isholder) holderaddresses.push(_holder);
}
if (status == false) {
(bool _isholder, uint256 s) = isHolderInArray(_holder);
if (_isholder) {
holderaddresses[s] = holderaddresses[
holderaddresses.length - 1
];
holderaddresses.pop();
}
holder[_holder] = status;
}
}

/////To keep track of holders for future use
function manageNumHolders(uint _option) external {
require(
msg.sender == wrapperaddress || msg.sender == Owner,
"Not Oracle/Owner!"
);
if (_option == 1) //remove holder
{
numholders -= numholders - 1;
}
if (_option == 2) //add holder
{
numholders += 1;
}
}
```

```

/////Returns whether the user is stored in the array////////
function isHolderInArray(address _wallet) public view returns (bool, uint) {
for (uint256 s = 0; s < holderaddresses.length; s += 1) {
if (_wallet == holderaddresses[s]) return (true, s);
}
return (false, 0);
}

////////////////////////

///Function to manage addresses
function manageBlockedNFT(
int option,
uint _tokenID,
address _wallet,
uint _numNFT,
bool _onoroff
) external onlyOwner {
address temp;
if (option == 1) // Add NFT to block list
{
blocked[_tokenID] = true;
numblocked += 1;
}
if (option == 2) //Remove from mapping
{
bool _isblocked = blocked[_tokenID];
if (_isblocked) {
blocked[_tokenID] = false;
if (numblocked > 0) {
numblocked -= 1;
}
}
}
if (
option == 3
) //Iterate through entire colletion and add. Added as a nice to have, but an iteration through an enite collection is expensive
{
for (uint256 s = 0; s < _numNFT; s += 1) {
if (s > 0) {
temp = ownerOfToken(s);

if (temp == _wallet) {
blocked[s] = true;
numblocked += 1;
}
}
}
}
if (option == 4) {
//setup blocking of addresses
blockedaddresses[_wallet] = _onoroff;
}
}

//Set the status of a user if they have wrapped!
function setUserStatus(address _wrapper, bool _haswrapped) external {
require(
msg.sender == Owner || msg.sender == wrapperaddress,
"Not Auth(WS)"
);
wrapped[_wrapper] = _haswrapped;
numwraps += 1; //track number of wraps
}

//Returns whether a user has wrapped before..
function getWrappedStatus(address _migrator) external view returns (bool) {
bool temp;
if (wrapped[_migrator] == true) {
temp = true;
}
return temp;
}

//Returns stats based off
// 1) numholders based off the number of wrappers
// 2) The length of the array with addresss of wrappers
// 3) The number of current blockedNFT's
function getNumHolders(uint _feed) external view returns (uint) {
uint temp;
if (_feed == 1) {
temp = numholders;
}
if (_feed == 2) {
temp = holderaddresses.length;
}
if (_feed == 3) {
temp = blockednfts.length;
}
return temp;
}

///Returns the holder address given an Index
function getHolderAddress(uint _index)
external
view
returns (address payable)
{
address temp;
address payable temp2;
temp = holderaddresses[_index];
temp2 = payable(temp);
return temp2;
}

//Returns OwnerOf the original Rugged NFT itself
//Saves having to add an additional ABI in a webpage/contract to verify
function ownerOfToken(uint _tid) public view returns (address) {
address temp;
temp = ruggedNFT(ruggedproject).ownerOf(_tid);
return temp;
}
}
```