MythX

REPORT 617A24594080B20019877B7E

Created	Thu Oct 28 2021 04:17:29 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	617a1df543f2c3936612f47c

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
164ebc2c-379b-4e6e-b0ab-e1c89f2c36fe	Pharmacy.sol	3

Analysis 164ebc2c-379b-4e6e-b0ab-e1c89f2c36fe MythX

Started	Thu Oct 28 2021 04:17:34 GMT+0000 (Coordinated Universal Time)
Finished	Thu Oct 28 2021 04:19:45 GMT+0000 (Coordinated Universal Time)
Mode	Quick
Client Tool	Mythx-Cli-0.6.22
Main Source File	Pharmacy.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	3

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.8.0"". 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
Pharmacy.sol
Locations

```
1 pragma solidity ^0.8.0;
2
3 interface IERC165 {
```

LOW

SWC-107

A call to a user-supplied address is executed.

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file
Pharmacy.sol
Locations

```
795 function sendFunds (address _target, uint256 _amount) external onlyAdmin {
796 require(address(this).balance >= _amount);
797 (bool success, ) = _target.call{value: _amount}("");
798 require(success, "Transfer failed.");
799 }
```

LOW

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).

SWC-123

Source file

Pharmacy.sol

Locations

```
795 function sendFunds (address _target, uint256 _amount) external onlyAdmin {
796 require(address(this).balance >= _amount);
797 (bool success, ) = _target.call{value: _amount}("");
798 require(success, "Transfer failed.");
799 }
```

Source file

Pharmacy.sol

Locations

```
562 }
563
564 contract Pharmacy is AccessControl{
565 using SafeMath for uint256;
566 using SafeMath for uint144;
567 using SafeMath for uint32;
568
569 /// @dev The contract deployer is assigned the DEFAULT_ADMIN_ROLE as per AccessControl.sol
570 constructor() {
571 _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
572 }
573
574 bytes32 public constant PRESCRIBER_ROLE = keccak256("PRESCRIBER_ROLE");
575
576 /// @dev Modifier to restrict access to accounts that DEFAULT_ADMIN_ROLE has granted the PRESCRIBER_ROLE
577 modifier onlyPrescriber() {
578 require(hasRole(PRESCRIBER_ROLE, msg.sender), "You are not a prescriber");
579 _;
580 }
581
582 /// @dev Modifier to restrict access to DEFAULT_ADMIN_ROLE
583 modifier onlyAdmin() {
584 require(hasRole(DEFAULT_ADMIN_ROLE, msg.sender), "You are not a pharmacy admin");
585 _;
586 }
587
588 /// @dev Modifier to check that a prescriptionId is valid as a function input
589 /// @param _prescriptionId The prescription ID number
590 modifier isPrescriptionValid(uint256 _prescriptionId) {
591 require(prescriptionCount >= _prescriptionId, "This prescription doesn't exist yet");
592 require(scripts[_prescriptionId].prescriptionValid == true, "This script is invalid");
593 require(scripts[_prescriptionId].dispensed == false, "This prescription has already been purchased");
594 _;
595 }
596
597 event NewScript(uint256 indexed prescriptionId, address indexed patient, string indexed medication);
598 event ScriptCancelled(uint256 indexed prescriptionId);
599 event ScriptEdited(uint256 indexed prescriptionId);
600 event ScriptDispensed(uint256 indexed prescriptionId, address indexed patient, string indexed medication);
601
602 /// @dev struct to represent a script
603 struct Script {
604 uint256 prescriptionId;
605 address prescriber;
606 address patient;
```

```
string medication; // Tried declaring as a "bytes32" first, but decided it was simpler for the dev and user experience to use "string", even if gas costs are higher
// Pack the following variables into a single 256-bit storage slot
uint32 timePrescribed; // uses single storage slot - 32 bits
uint32 timeDispensed; // uses single storage slot - 32 bits
bool prescriptionValid; // uses single storage slot - 8 bits
bool dispensed; // uses single storage slot - 8 bits
uint32 dose; // uses single storage slot - 32 bits
uint144 price; // uses single storage slot - 144 bits
// 32 + 32 + 8 + 8 + 32 + 144 = 256 bits
string instructions; // Store unit, repeats, quantity, indication, route here
}

/// @dev For a public array of structs, Solidity has a limitation of 12 properties or else it calls a "Stack Too Deep" error
Script[] private scripts;

uint public prescriptionCount; //How many total prescriptions are there?
mapping(address => uint256) private prescriberActivePrescriptionCount; //How many active prescriptions does a prescriber have?
mapping(address => uint256) private patientActivePrescriptionCount; //How many active prescriptions does a patient have?
mapping(address => uint256[]) private prescriberPrescriptions; //What presciptions has this prescriber created?
mapping(address => uint256[]) private patientPrescriptions; //What prescriptions has this patient been prescribed?

/* PRESCRIBER FUNCTIONS */

/** @notice Create a prescription - PRESCRIBER ONLY
* @param _patient Patient address
* @param _medication Medication as a string
* @param _dose Dose
* @param _instructions Prescription instructions as a string
* @return uint256 Returns the prescriptionId of the newly created prescription
*/
function createPrescription(
address _patient,
string memory _medication,
uint32 _dose,
string memory _instructions
) public onlyPrescriber returns (uint256) {
require (msg.sender != _patient, "You are not allowed to prescribe for yourself");

uint256 prescriptionId = prescriptionCount++;

scripts.push(Script(
prescriptionId,
msg.sender,
_patient,
_medication,
uint32(block.timestamp),
0, //If I declare 0 here, is it a uint32 or a uint256?
true,
false,
_dose,
10**16, // Set default price of 0.01 ETH, deciding price mechanism for later
_instructions
));

prescriberActivePrescriptionCount[msg.sender]++;
patientActivePrescriptionCount[_patient]++;
prescriberPrescriptions[msg.sender].push(prescriptionId);
patientPrescriptions[_patient].push(prescriptionId);

emit NewScript(prescriptionId, _patient, _medication);

return prescriptionId;
}
```

```
670
671 /** @notice Cancel a prescription - CAN ONLY BE USED BY THE PRESCRIBER FOR THEIR OWN CREATED PRESCRIPTIONS
672 * @param _prescriptionId Prescription ID number
673 * @return bool Return 'true' if the function is successful
674 */
675 function cancelPrescription(uint256 _prescriptionId) public onlyPrescriber isPrescriptionValid(_prescriptionId) returns (bool) {
676 require(scripts[_prescriptionId].prescriber == msg.sender, "You did not create this prescription");
677 scripts[_prescriptionId].prescriptionValid = false;
678 patientActivePrescriptionCount[scripts[_prescriptionId].patient]--;
679 prescriberActivePrescriptionCount[scripts[_prescriptionId].prescriber]--;
680 emit ScriptCancelled(_prescriptionId);
681 return true;
682 }
683
684 /** @notice Edit a prescription - CAN ONLY BE USED BY THE PRESCRIBER FOR THEIR OWN CREATED AND ACTIVE PRESCRIPTIONS
685 * @param _prescriptionId Prescription ID number of the script we want to edit
686 * @param _medication What we want to change the medication to
687 * @param _dose What we want to change the medication to
688 * @param _dose What we want to change the dose to
689 * @param _instructions What we want to change the instructions to
690 * @return bool Return 'true' if the function is successful
691 */
692 function editPrescription(
693 uint256 _prescriptionId,
694 string memory _medication,
695 uint32 _dose,
696 string memory _instructions
697 ) public onlyPrescriber isPrescriptionValid(_prescriptionId) returns (bool) {
698 require(scripts[_prescriptionId].prescriber == msg.sender, "You did not create this prescription");
699
700 scripts[_prescriptionId].medication = _medication;
701 scripts[_prescriptionId].dose = _dose;
702 scripts[_prescriptionId].instructions = _instructions;
703
704 emit ScriptEdited(_prescriptionId);
705
706 return true;
707 }
708
709 /* GETTER FUNCTIONS */
710
711 /** @notice Get the details for a specific script
712 * @dev Starting in Solidity 0.8.0, functions can return structs
713 * @param _prescriptionId Prescription ID number of the script we want details for
714 * @return struct Script with corresponding prescription ID
715 */
716 function getScriptInformation(uint256 _prescriptionId) public view returns (Script memory) {
717 require(hasRole(PRESCRIBER_ROLE, msg.sender) || msg.sender == scripts[_prescriptionId].patient || hasRole(DEFAULT_ADMIN_ROLE, msg.sender), "You are not allowed to view this
718 script");
719 return scripts[_prescriptionId];
720 }
721
722 /** @notice Get the number of active scripts for a prescriber - A prescriber can only call this for themselves
723 * @param _prescriber Prescriber address
724 * @return prescriptionCount
725 */
726 function get_prescriberActivePrescriptionCount(address _prescriber) public view returns (uint256 prescriptionCount) {
727 require(hasRole(PRESCRIBER_ROLE, msg.sender) || hasRole(DEFAULT_ADMIN_ROLE, msg.sender), "You are not allowed to use this getter function");
728 require(msg.sender == _prescriber, "You can only see your own prescription count");
729 prescriptionCount = prescriberActivePrescriptionCount[_prescriber];
730 }
731
732 /** @notice Get the number of active scripts for a patient - A patient can only call this themselves
```

```
* @param _patient Patient address
* @return prescriptionCount
*/
function get_patientActivePrescriptionCount(address _patient) public view returns (uint256 prescriptionCount) {
require(hasRole(PRESCRIBER_ROLE, msg.sender) || hasRole(DEFAULT_ADMIN_ROLE, msg.sender) || msg.sender == _patient, "You are not allowed to use this getter function");
prescriptionCount = patientActivePrescriptionCount[_patient];
}

// We allow prescribers only to see their own prescriptions, or the DEFAULT_ADMIN_ROLE

/** @notice Get the scripts that a prescriber has created - A prescriber can only call this for themselves, and patients cannot use this function
* @param _prescriber Prescriber address
* @return prescriptionIds Dynamic array containing prescription IDs that the prescriber was created
*/
function get_prescriberPrescriptions(address _prescriber) public view returns (uint256[] memory prescriptionIds) {
require(hasRole(PRESCRIBER_ROLE, msg.sender) || hasRole(DEFAULT_ADMIN_ROLE, msg.sender), "You are not allowed to use this getter function");
require(msg.sender == _prescriber, "You can only see your own prescription count");
prescriptionIds = prescriberPrescriptions[_prescriber];
}

/** @notice Get the scripts that a prescriber has created - A prescriber can only call this for themselves, and patients cannot use this function
* @param _patient Patient address
* @return prescriptionIds Dynamic array containing prescription IDs of the scripts that the patient has been assigned
*/
function get_patientPrescriptions(address _patient) public view returns (uint256[] memory prescriptionIds) {
require(hasRole(PRESCRIBER_ROLE, msg.sender) || hasRole(DEFAULT_ADMIN_ROLE, msg.sender) || msg.sender == _patient, "You are not allowed to use this getter function");
prescriptionIds = patientPrescriptions[_patient];
}

/* PATIENT FUNCTIONS */

/** @notice Purchase a script - requires sending ETH as payment
* @param _prescriptionId Prescription ID of the script we want to purchase
* @return bool true if the function is successful
*/
function purchase (uint256 _prescriptionId) payable public isPrescriptionValid(_prescriptionId) returns (bool) {
require(msg.sender == scripts[_prescriptionId].patient, "This is not your script");
require(msg.value >= scripts[_prescriptionId].price, "You did not pay enough");

scripts[_prescriptionId].timeDispensed = uint32(block.timestamp);
scripts[_prescriptionId].prescriptionValid = false;
scripts[_prescriptionId].dispensed = true;

emit ScriptDispensed(_prescriptionId, scripts[_prescriptionId].patient, scripts[_prescriptionId].medication);

return true;
}

/* PHARMACY ADMIN FUNCTIONS */

/** @notice Withdraw ETH from the Pharmacy smart contract
* @param _amount Amount of ETH desired for withdrawal
*/
function withdrawFunds (uint256 _amount) external onlyAdmin {
require(address(this).balance >= _amount);
(bool success, ) = msg.sender.call{value: _amount}("");
require(success, "Transfer failed.");
}

/** @notice Send ETH from the Pharmacy smart contract to a desired address
* @param _target Desired target address for sending funds
* @param _amount Amount of ETH desired to send
*/
```

```
function sendFunds (address _target, uint256 _amount) external onlyAdmin {
require(address(this).balance >= _amount);
(bool success, ) = _target.call{value: _amount}("");
require(success, "Transfer failed.");
}

/* BACKDOOR FUNCTIONS FOR BOOTCAMP ASSESSMENT PURPOSES */

/** @notice Become a prescriber
* @dev This function is only included for demonstration purposes so the assessor can have easy access to both the prescriber and patient UIs
* @dev This function should be deleted for actual use
* @dev We are using _setupRole() outside of the constructor function, which is circumventing the admin system imposed by AccessControl.sol
*/
function becomePrescriber() public {
_setupRole(PRESCRIBER_ROLE, msg.sender);
}
}
```