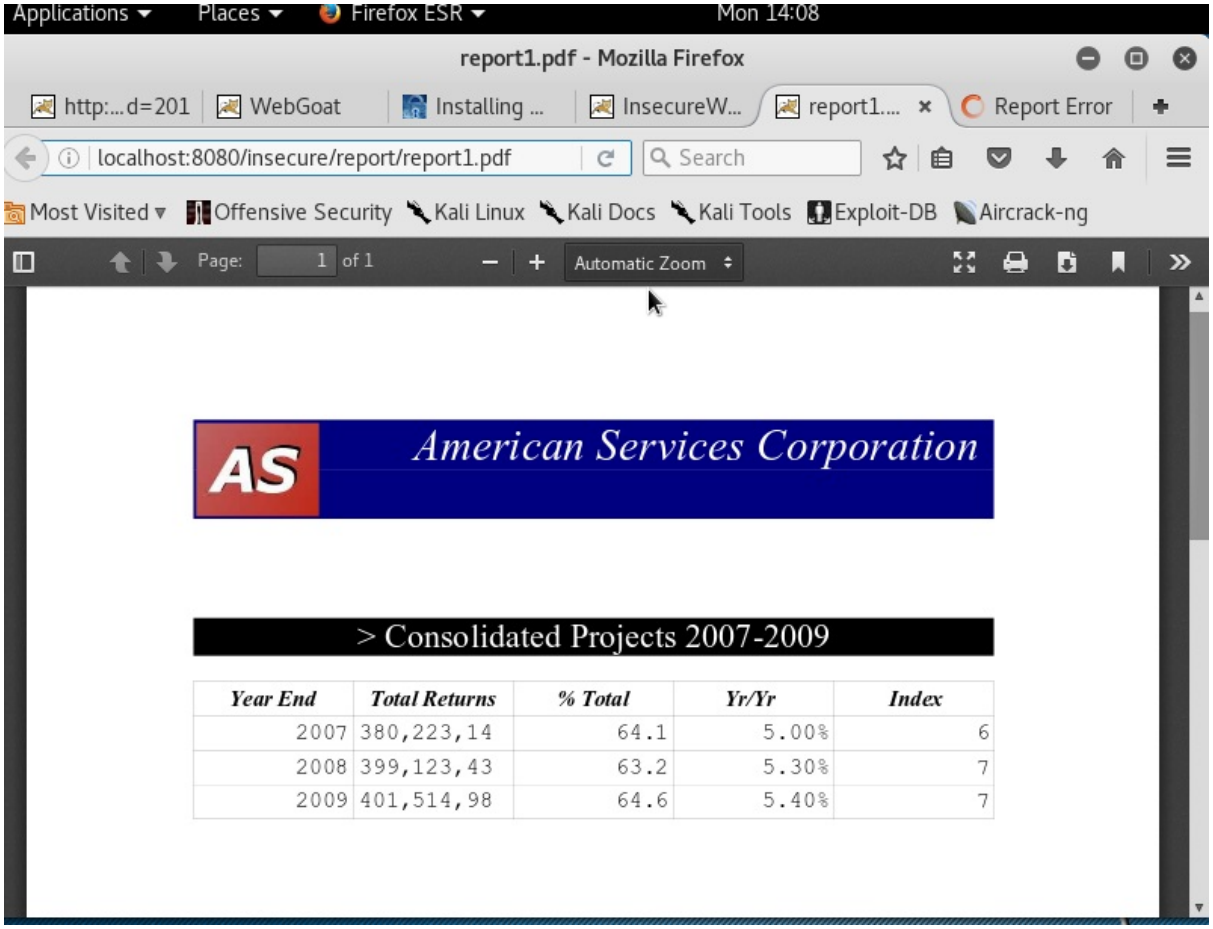
Puneet Chhabra

Assignment-2 (V00871808)

Challenge 5

1. Find a page that is vulnerable to cross site scripting (XSS)

Answer: I found a page which was vulnerable after signing in as "admin" is shown below:



Here below you can see how I tried to find the vulnerability in the site by using "OWASP-ZAP". The screenshot is attached for the purpose (

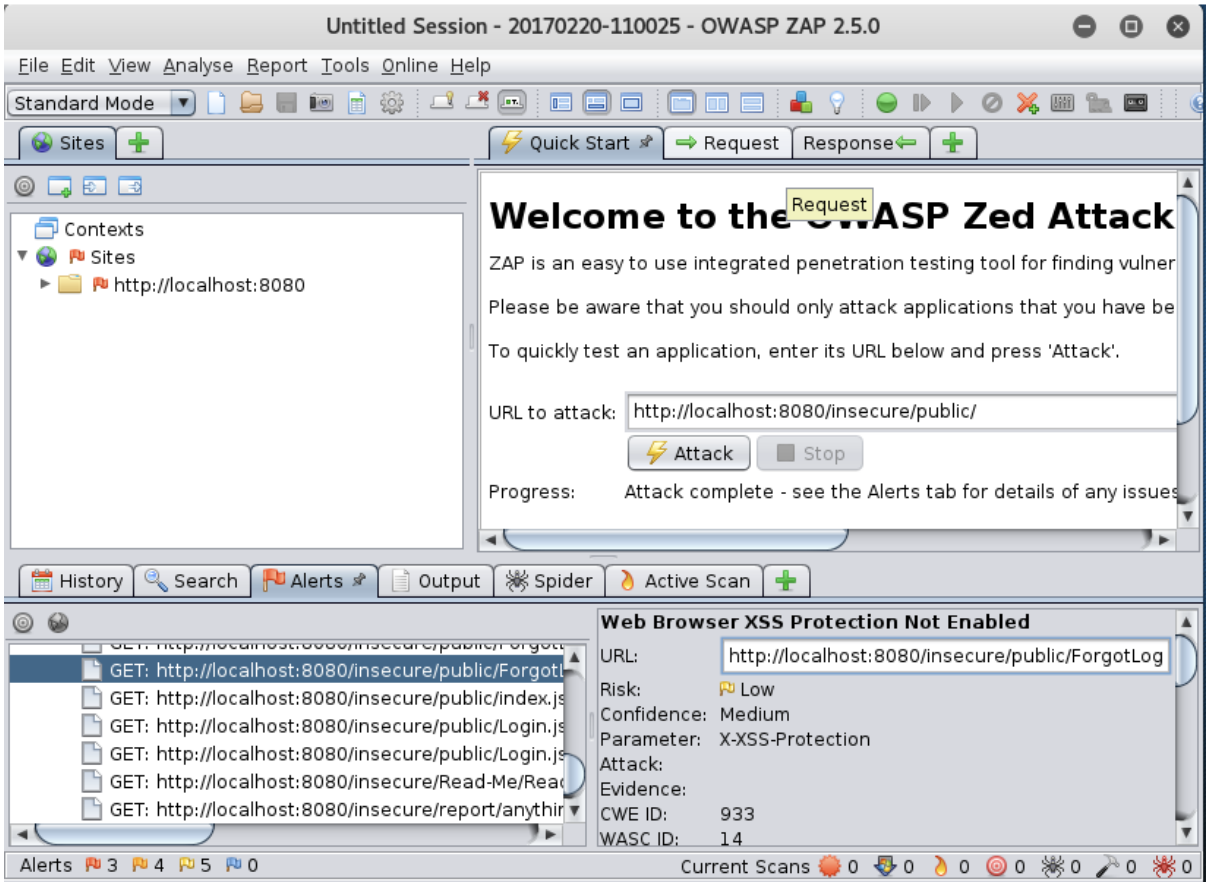
Other websites which I found was vulnerable were:

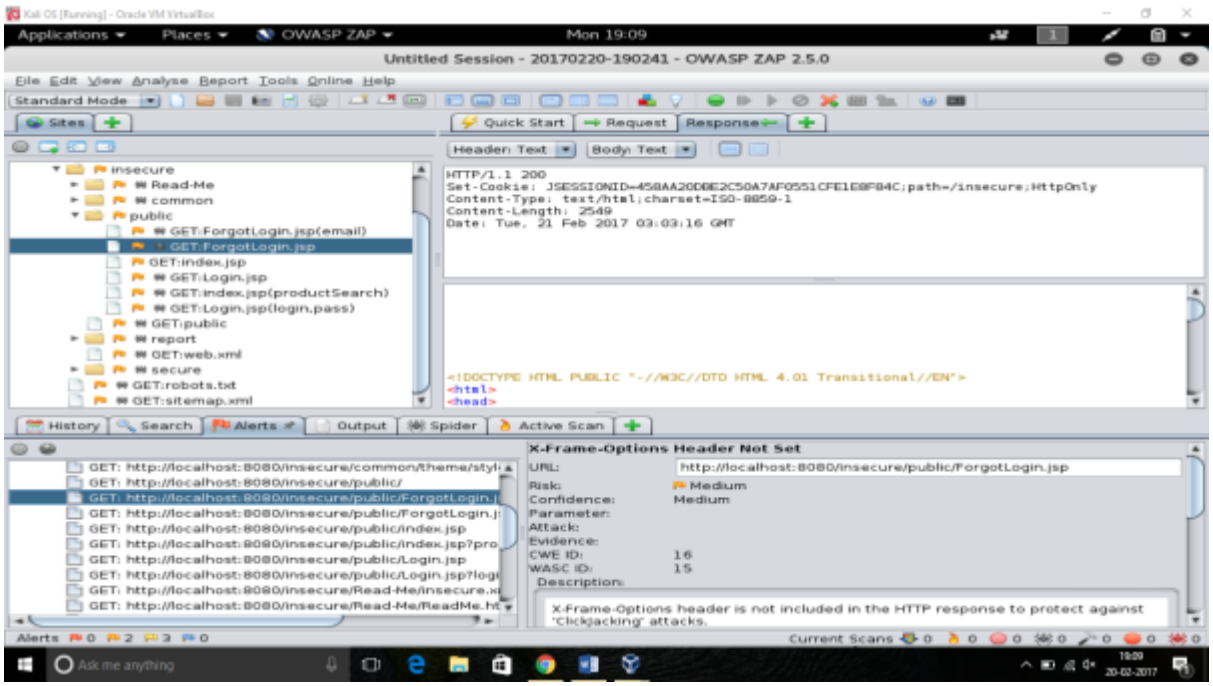
localhost:8080/insecure/secure/sqltest.jsp/

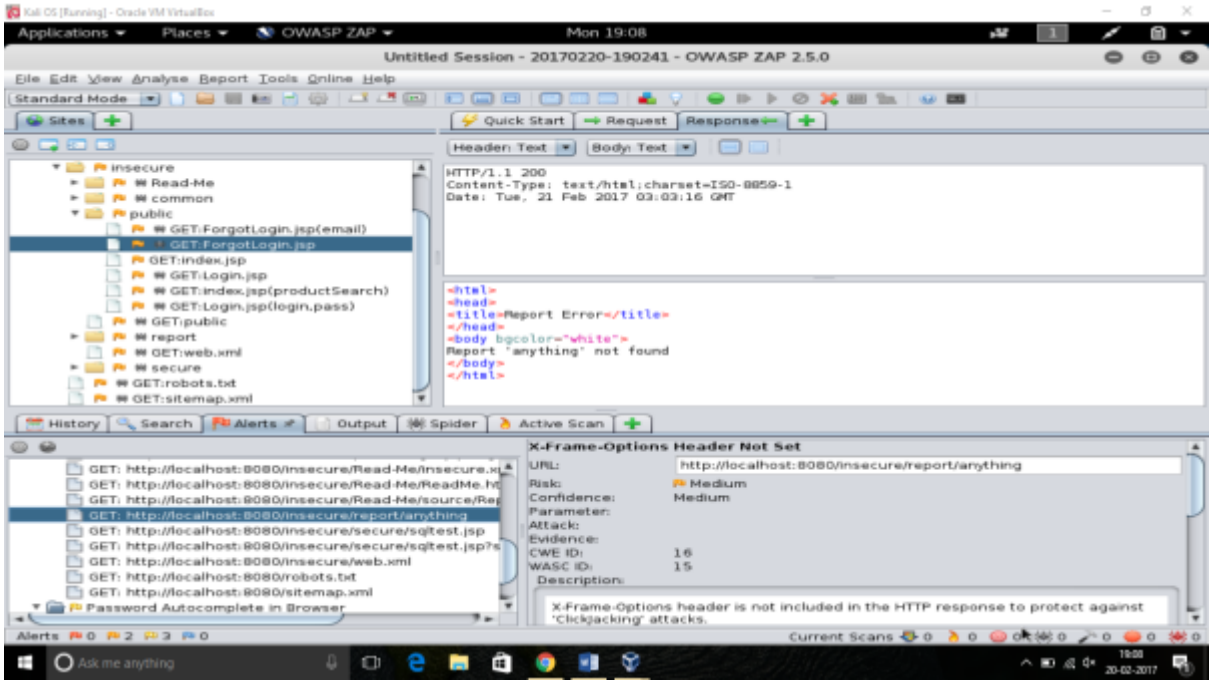
http://localhost:8080/insecure/report/

http://localhost:8080/insecure/public/ForgotLogin.jsp

) :

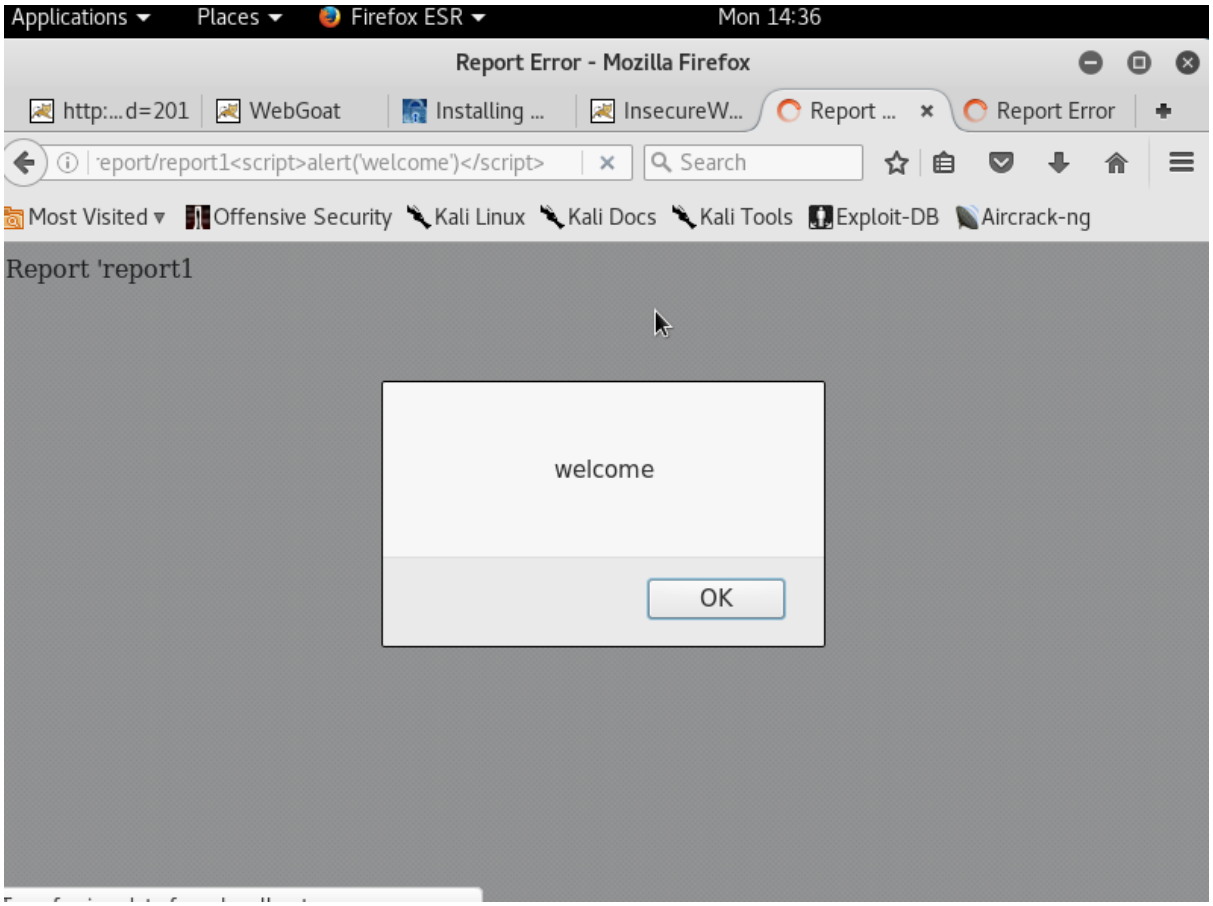

Untitled Session - 20170220-110025 - OWASP ZAP 2.5.0
File Edit View Analyse Report Tools Online Help
Standard Mode
Sites
Quick Start
Request
Response
Contexts
Sites
http://localhost:8080
Welcome to the OWASP Zed Attack
ZAP is an easy to use integrated penetration testing tool for finding vulner
Please be aware that you should only attack applications that you have be
To quickly test an application, enter its URL below and press 'Attack'.
URL to attack: http://localhost:8080/insecure/public/
Attack
Stop
Progress: Attack complete - see the Alerts tab for details of any issues
History
Search
Alerts
Output
Spider
Active Scan
GET: http://localhost:8080/insecure/public/ForgotL
GET: http://localhost:8080/insecure/public/index.js
GET: http://localhost:8080/insecure/public/Login.js
GET: http://localhost:8080/insecure/public/Login.js
GET: http://localhost:8080/insecure/Read-Me/Rea
GET: http://localhost:8080/insecure/report/anythir
Web Browser XSS Protection Not Enabled
URL: http://localhost:8080/insecure/public/ForgotLog
Risk: Low
Confidence: Medium
Parameter: X-XSS-Protection
Attack:
Evidence:
CWE ID: 933
WASC ID: 14
Alerts 3 4 5 0
Current Scans 0 0 0 0 0 0 0


Applications
Places
OWASP ZAP
Mon 19:09
Untitled Session - 20170220-190241 - OWASP ZAP 2.5.0
File Edit View Analyse Report Tools Online Help
Standard Mode
Sites
Quick Start
Request
Response
Header: Text
Body: Text
insecure
Read-Me
common
public
GET:ForgotLogin.jsp(email)
GET:ForgotLogin.jsp
GET:index.jsp
GET:Login.jsp
GET:index.jsp(productSearch)
GET:Login.jsp(login,pass)
GET:public
report
GET:web.xml
secure
GET:robots.txt
GET:sitemap.xml
HTTP/1.1 200
Set-Cookie: JSESSIONID=458AA20DBE2C50A7AF0551CFE1E8F84C;path=/insecure;HttpOnly
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 2549
Date: Tue, 21 Feb 2017 03:03:16 GMT
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
History
Search
Alerts
Output
Spider
Active Scan
X-Frame-Options Header Not Set
URL: http://localhost:8080/insecure/public/ForgotLogin.jsp
Risk: Medium
Confidence: Medium
Parameter:
Attack:
Evidence:
CWE ID: 16
WASC ID: 15
Description:
X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.
Alerts 0 2 3 0
Current Scans 0 0 0 0 0 0 0
Ask me anything



2. Demonstrate that the page is vulnerable by generating an alert.

Ans. Below you can see I have demonstrated the page as "Welcome" after injecting the script is shown below:



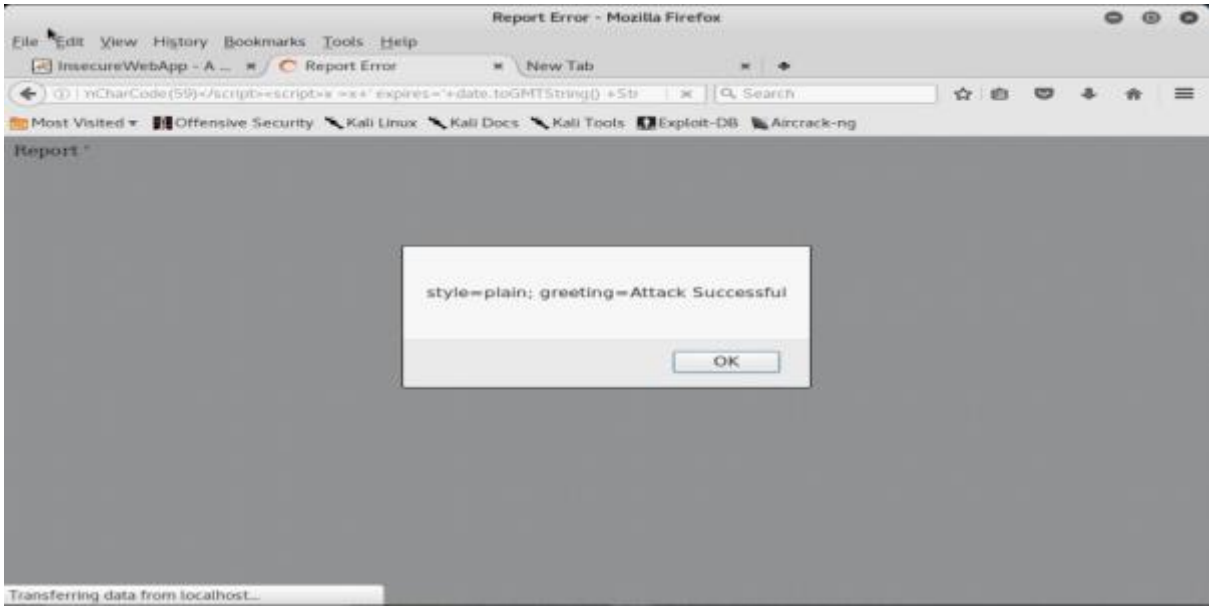
http://localhost:8080/insecure/report/<script>alert<'welcome'></script>

3. Use XSS to change the user preference's salutation cookie

Answer:

Here is the code that has been used to transform the user preference's salutation cookie:

http://localhost:8080/insecure/report/
<script>var date = new Date() </script>
<script>date.setTime(date.getTime()+(10055*66*360*6000*100000))</script>
<script>var x ='greeting=Attack Successful'+ String.fromCharCode(59)</script>
<script>x =x+' expires='+date.toGMTString() +
String.fromCharCode(59)</script>
<script>x =x+ ' path=/'</script>
<script>document.cookie=(x)</script>
<script>alert(document.cookie)</script>



As Mozilla misinterprets "," I have written the asci code for "," as String.fromCharCode(59)

4. How could you use this to automatically inject malicious code into every secure page?

Answer:

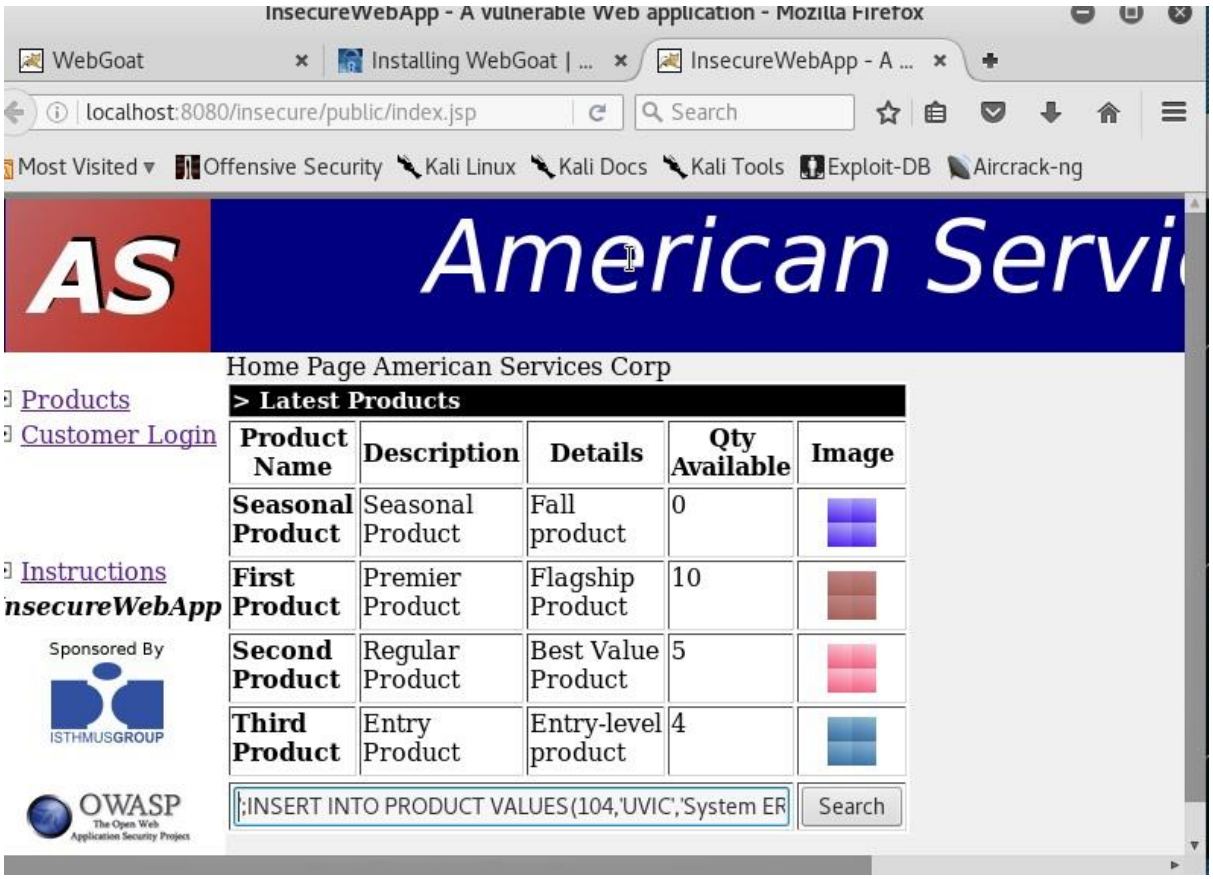
After logging in the site as an administrator and then entering the following search query in the product page I was able to enter into each page on the site.

Also, before injecting the page I also tried logging as normal user and I was not able to search. First I logged in as admin and then entered the search query.

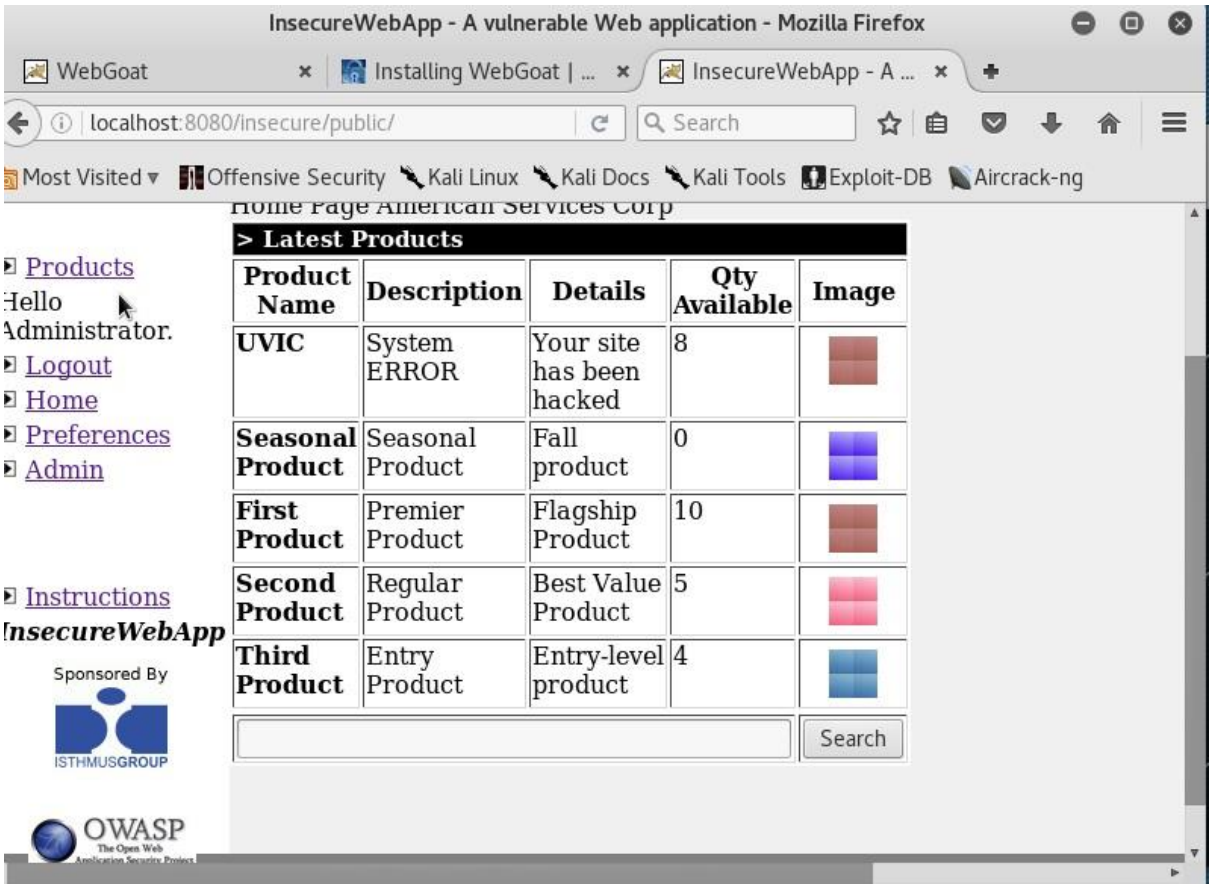
Search query:

';INSERT INTO PRODUCT VALUES (104, 'UVIC', 'your site has been hacked', 'product1001.jpg>', 8)

Below you can see the script is being searched in the search bar:



Here you can see the error being displayed in the latest products section:



5. What would be a minimum fix and what would be a good fix?

Answer:

As you know that, XSS is an input validation attack. So if you can replace "<" in the script with something else like " %3E " than that would be a minimum fix because the attacker will not know what the website has used instead of the " < > ".

For example, if we implement the above method than the attack performed in the first step should look something like this.

http://localhost:8080/insecure/report/report1<script>alert%28'welcome'%29</script>

Now it is difficult for an attacker to guess the encoded values of "<" & ">". It is not impossible for him to guess but still this would be a minimum fix for the website

More permanent fix would be to not let JavaScript run that are not from trusted sources. Also the external user should not be able to access internal resources.

To ensure that malicious scripting code is not output as part of a page, the application needs to encode all variable strings before they're displayed on a page. Encoding is merely converting every character to its HTML entity name.

For example, HTML encoding for the input String "<script>alert(\"abc \")</script>" can be converted in to <script>alert("abc")</script>.

Every character of the string will be converted into its entity name.

Encoding URL Parameters:

If you want to build a URL query string with untrusted input as a value use the UrlEncoder to encode the value. For example

var example = "\"Quoted Value with spaces and &\""; var encodedValue = _urlEncoder.Encode(example);

After encoding the encoded Value variable will

contain %22Quoted%20Value%20with%20spaces%20and%20%26%22. Spaces, quotes, punctuation and other unsafe characters will be percent encoded to their hexadecimal value, for example a space character will become %20.

How Cross-site Scripting works

In order for an XSS attack to take place the vulnerable website needs to directly include user input in its pages. An attacker can then insert a string that will be used within the web page and treated as code by the victim's browser.

The following server-side pseudo-code is used to display the most recent comment on a web page.

```
print "<html>"
print "<h1>Most recent comment</h1>"
print database.latestComment
print "</html>"
```

The above script is simply printing out the latest comment from a comments database and printing the contents out to an HTML page, assuming that the comment printed out only consists of text.

The above page is vulnerable to XSS because an attacker could submit a comment that contains a malicious payload such as <script>doSomethingEvil();</script>.

Users visiting the web page will get served the following HTML page.

```
<html>
<h1>Most recent comment</h1>
<script>doSomethingEvil();</script>
</html>
```

I recommend

a) **HTML encoding**

b) **JavaScript encoding all untrusted input, as shown in these examples:**

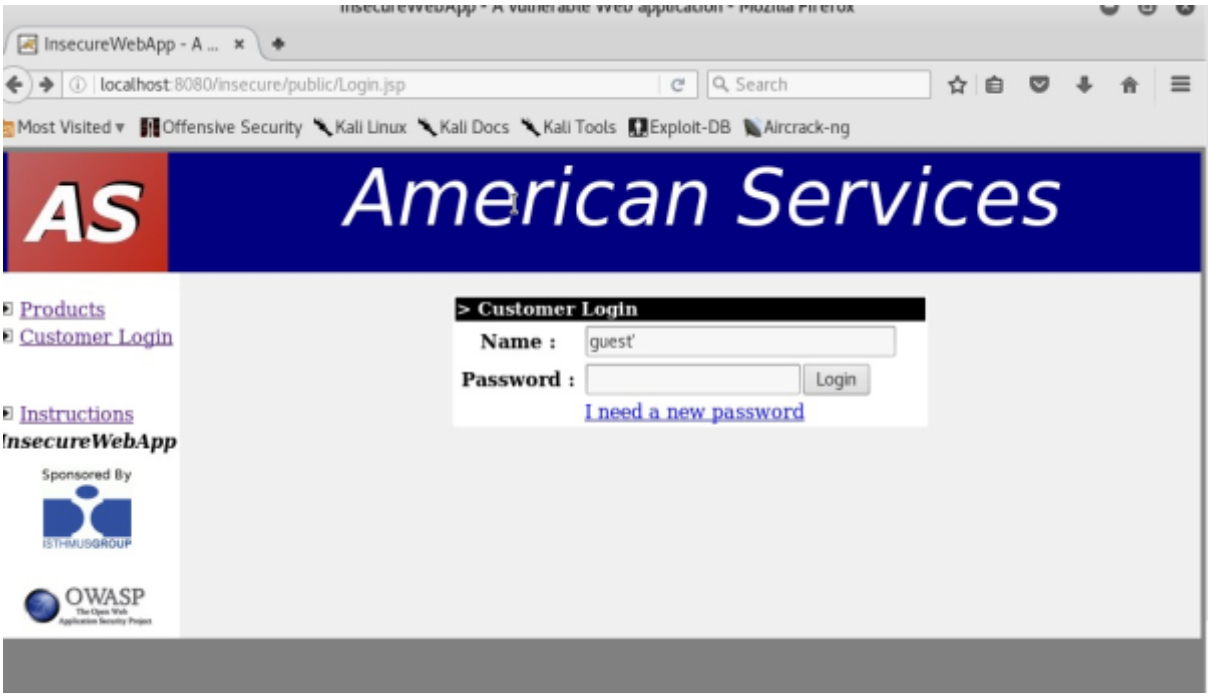
```
 element.innerHTML =
"<%=Encoder.encodeForJS(Encoder.encodeForHTML(untrustedData))%>";
element.outerHTML =
"<%=Encoder.encodeForJS(Encoder.encodeForHTML(untrustedData))%>";

document.write("<%=Encoder.encodeForJS(Encoder.encodeForHTML(untrustedData)
)%>");
document.writeln("<%=Encoder.encodeForJS(Encoder.encodeForHTML(untrustedDat
a))%>");
```

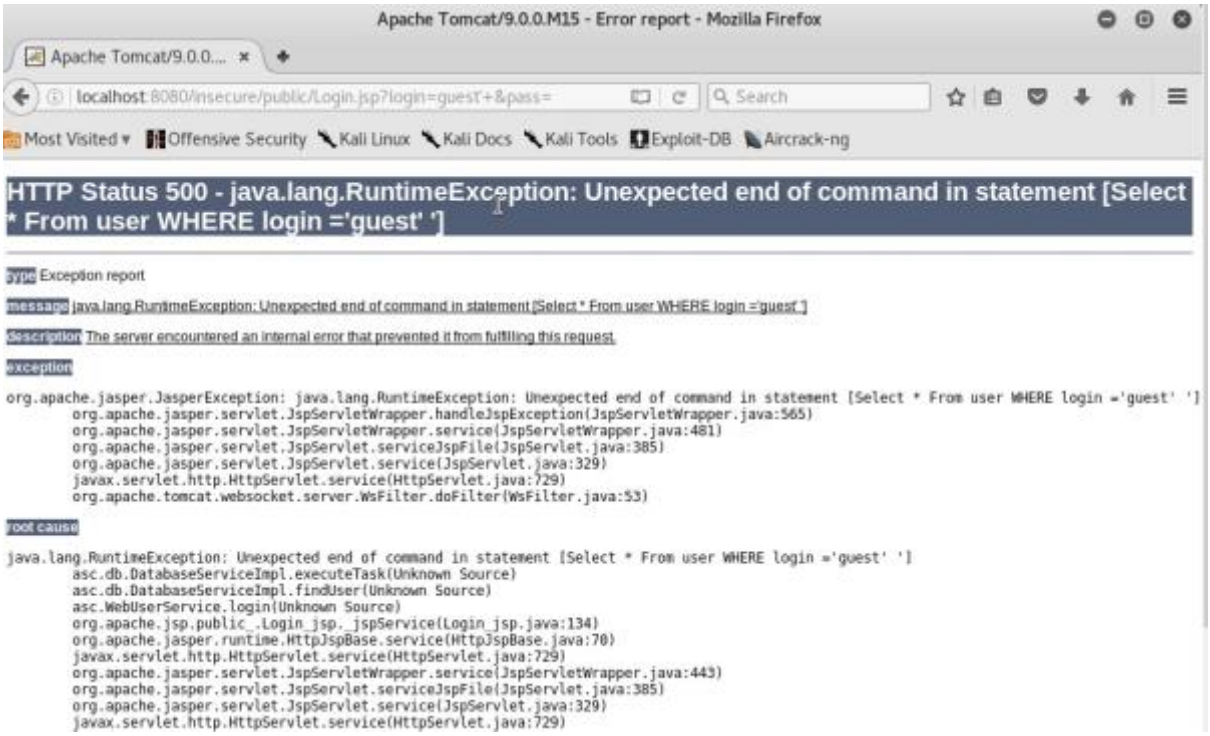
Challenge #6 - Permanent Cross Site Scripting

- **Use SQL injection to inject JavaScript into every secure page.**

Answer:

I tried to find the database name for user by injecting the following code: guest'. This is what I have got:

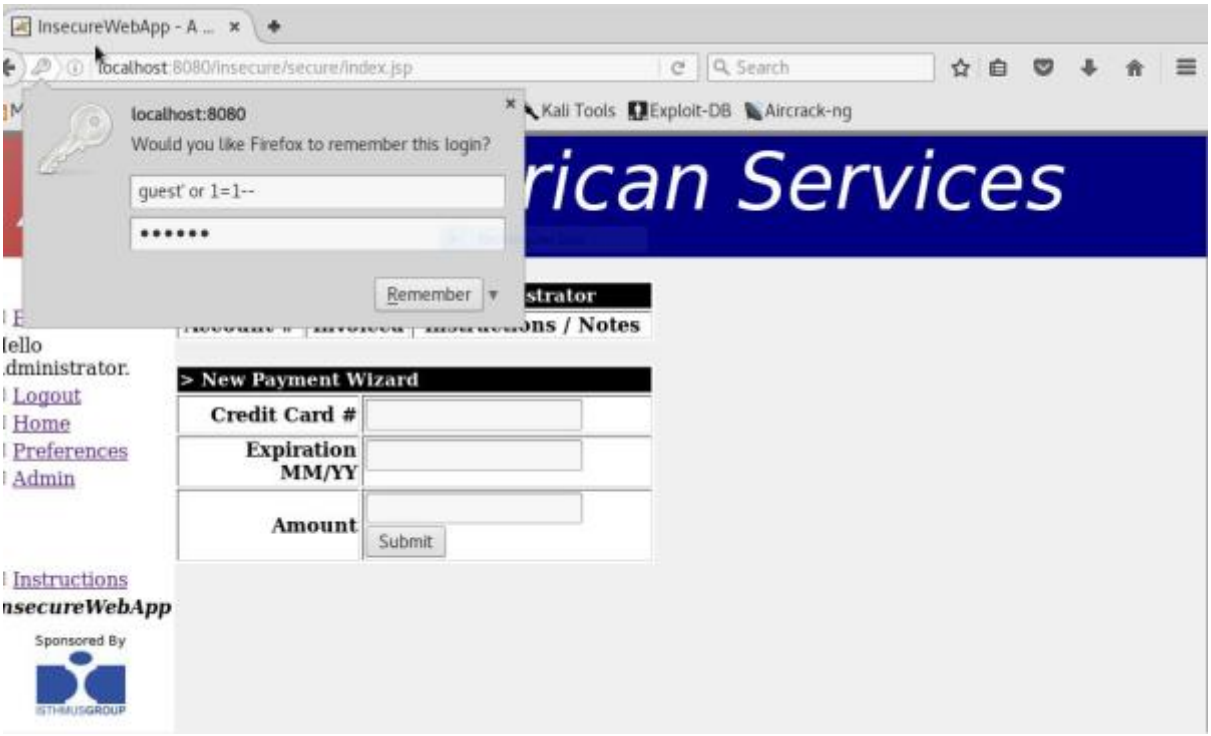




Now, I have added 1=1 at the end of username and – as it will comment out the rest of the sql query.

By doing this we will be able to login as administrator whether we have username or password or not. And by being able to login as admin we can implement further vulnerabilities as explained in the challenge 6 and can enter various product entry into the product page and also change the users' preference cookie.

The script " *guest' or 1=1--* " helps in injecting and logging into the system after putting any password in the bar. You can see the following result:



On the left corner, you can see I was able to login as an "admin".

- **What would be a minimum fix and what would be a good fix?**

Answer: Parameterized queries:

Parameterized queries are simple to write and understand. They force you to define the SQL query beforehand, and use placeholders for the user-provided variables within the query. You can then pass in each parameter to the query after the SQL statement is defined, allowing the database to be able to distinguish between the SQL command and data inputted by a user. If SQL commands are inputted by an attacker, the parameterized query would treat these as untrusted input, and the injected SQL commands will never get to execute.

Using Input validation

Ensuring that text entered in email address is actually matching with format of email-id. Allowing only numbers in phone number field and things like that.

To encode the HTML input coming from the textboxes and add the exceptions for the basic html tags. When the attacker inputs the script it is encoded and displays as it is.

A good fix would be to use an account that has restricted permissions in the database. Ideally, it should only grant execute permissions to selected stored procedures in the database and provide no direct table access. Avoid disclosing database error information. In the event of database errors, make sure it does not disclose detailed error messages to the user

References:

- **https://www.acunetix.com/websitesecurity/cross-site-scripting/**
- **https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet**