COMP30720 - Ethical Computer Hacking

Assignment 1

Michael Mullin, 17360573

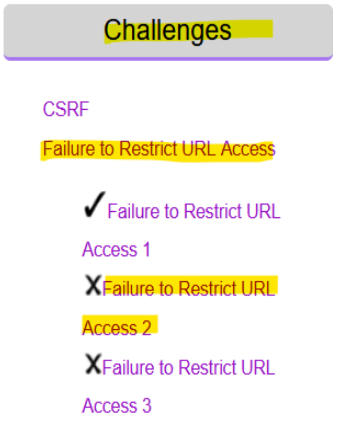
In this report, as part of our work for the OWASP Security Shepherd project, we have documented 4 different vulnerabilities which we have found on the webpage. We have detailed each vulnerability with the steps necessary to reproduce and appropriate screenshots attached.

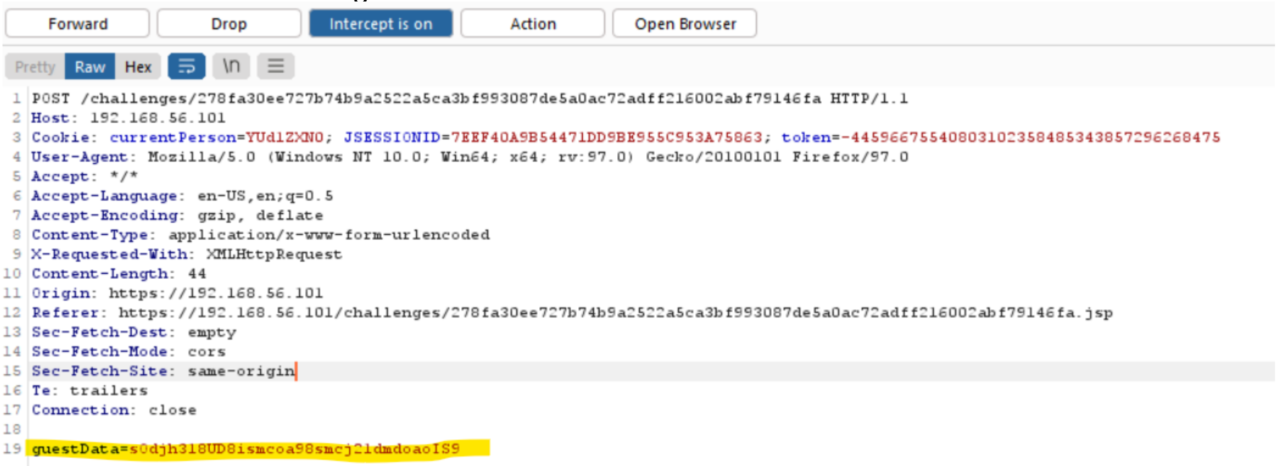
High: Failure to Restrict URL Access Challenge 2 [CWE-817]

Failure to restrict URL access is where users are able to access pages or areas on a webpage that are meant to be restricted or hidden. This occurs when there is an error in access-control settings. This presents a security concern as these pages frequently are less protected than pages that are meant for public access, and unauthorized users are able to reach the pages anonymously. Developers must ensure proper authentication policies are in place and ensure that all URLs are protected by an access control system to combat this attack.

Steps to reproduce:

1. Download and run Burp Suite from this link (also make sure you have Oracle Java installed)
2. Using Firefox web browser, set the system proxy to route traffic through Burp: Type in “about:preferences” in the search bar -> Scroll down and click “Settings” underneath Network Settings -> Select option “Manual Proxy Configuration” -> Set HTTP Proxy to 127.0.0.1 and Port to 8080 -> Click Ok.
3. Go to Security Shepherd https://192.168.56.101
4. Confirm that Burp can see and capture requests and turn off intercept in Burp.
5. Go to Challenges -> Failure to Restrict URL Access -> Failure to Restrict URL Access 2
6. We are tasked with finding an administrator only button considering we are “a mere guest”.
7. Right-click on the web page and click inspect. Turn on Intercept in Burp. Press “Get Guest Info”.
8. We can see the following request in Burp. The request uses guestData and a code when clicking “Get Guest Info”.

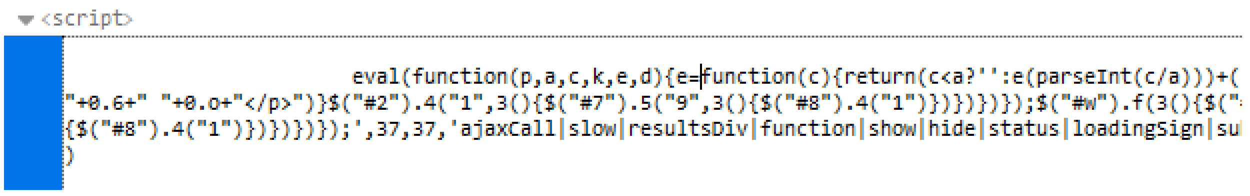

Challenges
CSRF
Failure to Restrict URL Access
✓Failure to Restrict URL Access 1
XFailure to Restrict URL Access 2
XFailure to Restrict URL Access 3



```
POST /challenges/278fa30ee727b74b9a2522a5ca3bf993087de5a0ac72adff216002abf79146fa HTTP/1.1
Host: 192.168.56.101
Cookie: currentPerson=YUd1ZXN0; JSESSIONID=7EEF40A9B54471DD9BE955C953A75863; token=-44596675540803102358485343857296268475
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:97.0) Gecko/20100101 Firefox/97.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Content-Length: 44
Origin: https://192.168.56.101
Referer: https://192.168.56.101/challenges/278fa30ee727b74b9a2522a5ca3bf993087de5a0ac72adff216002abf79146fa.jsp
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

guestData=s0djh318UD8ismcoa98smcj21dmdoaoIS9
```

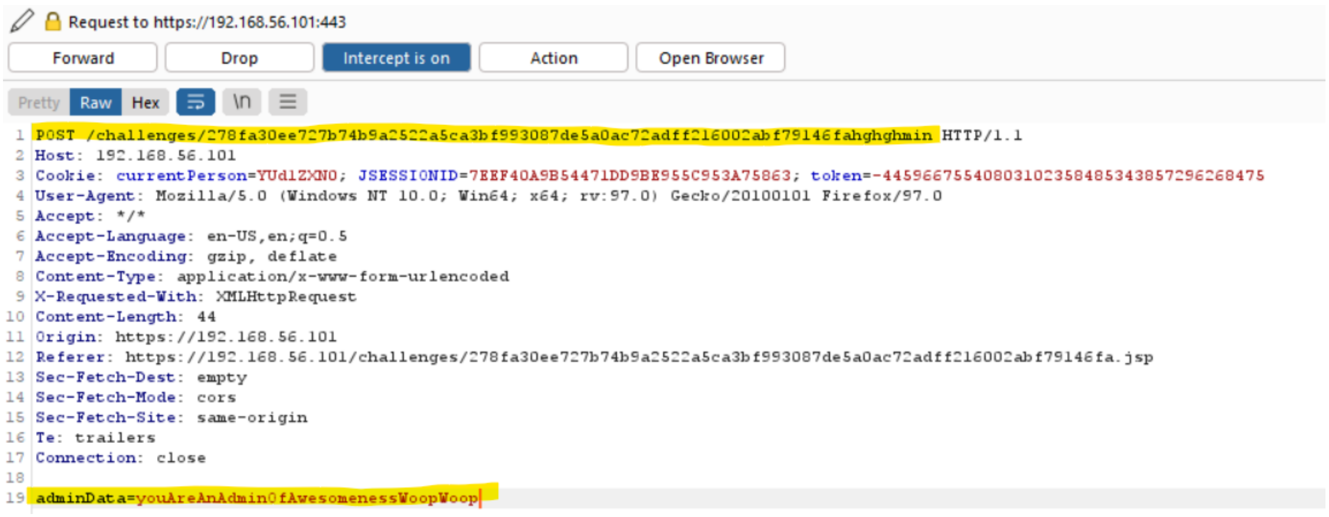
9. Inspecting the source code we can find a function called eval() which shows guestData, the code (both highlighted above) and the URL associated with these two. It also shows the same details for “IeAdministratorFormOfAwesomeness”.



```
<script>
                                        eval(function(p,a,c,k,e,d){e=function(c){return(c<a?'':e(parseInt(c/a)))+(
"+0.6+" "+0.o+"</p>")}$("#2").4("1",3(){$("#7").5("9",3(){$("#8").4("1")})})})});$("#w").f(3(){$("
{$("#8").4("1")})})})})});',37,37,'ajaxCall|slow|resultsDiv|function|show|hide|status|loadingSign|su
)
```

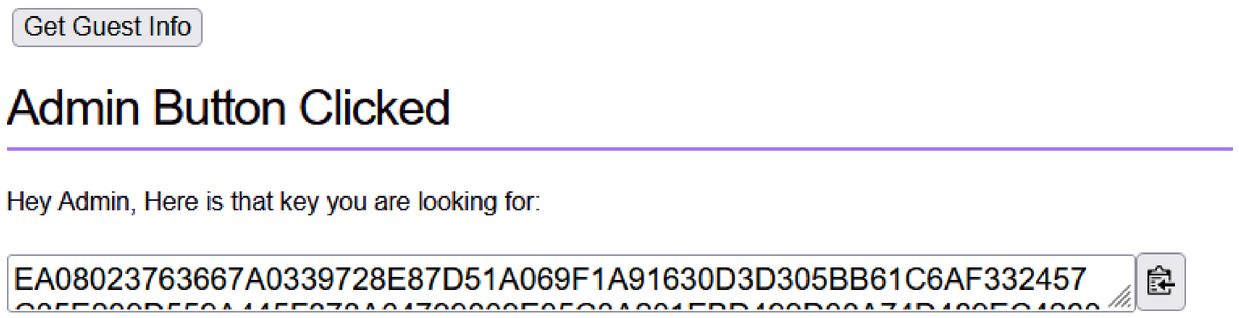
10. We can then substitute the adminData (shown below) into the request to gain administrative access.

Line 1: 278fa30ee727b74b9a2522a5ca3bf993087de5a0ac72adff216002abf79146fahghghmin
Line 19: adminData=youAreAnAdminOfAwesomenessWoopWoop



```
1  POST /challenges/278fa30ee727b74b9a2522a5ca3bf993087de5a0ac72adff216002abf79146fahghghmin HTTP/1.1
2  Host: 192.168.56.101
3  Cookie: currentPerson=YUd1ZXN0; JSESSIONID=7EEF40A9B54471DD9BE955C953A75863; token=-44596675540803102358485343857296268475
4  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:97.0) Gecko/20100101 Firefox/97.0
5  Accept: */*
6  Accept-Language: en-US,en;q=0.5
7  Accept-Encoding: gzip, deflate
8  Content-Type: application/x-www-form-urlencoded
9  X-Requested-With: XMLHttpRequest
10 Content-Length: 44
11 Origin: https://192.168.56.101
12 Referer: https://192.168.56.101/challenges/278fa30ee727b74b9a2522a5ca3bf993087de5a0ac72adff216002abf79146fa.jsp
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Te: trailers
17 Connection: close
18
19 adminData=youAreAnAdminOfAwesomenessWoopWoop
```

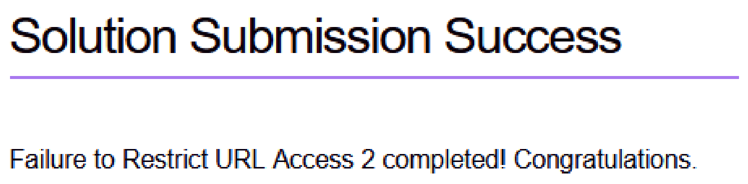
11. Click “Forward” on Burp and the request will go through. This passed the challenge and we have successfully gained Admin access.



Admin Button Clicked

Hey Admin, Here is that key you are looking for:

EA08023763667A0339728E87D51A069F1A91630D3D305BB61C6AF332457



Solution Submission Success

Failure to Restrict URL Access 2 completed! Congratulations.

CVSS Score 7.5 (High)

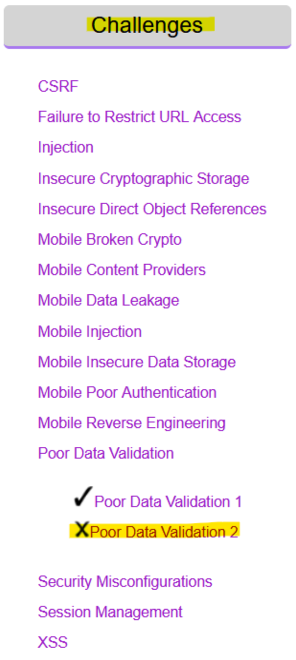
Attack Vector	**Network**
Attack Complexity	**Low**
Privileges Required	**None**
User Interaction	**None**
Scope	**Unchanged**
Confidentiality	**High**
Integrity	**None**
Availability	**None**

High: Poor Data Validation Challenge 2 [CWE-20]

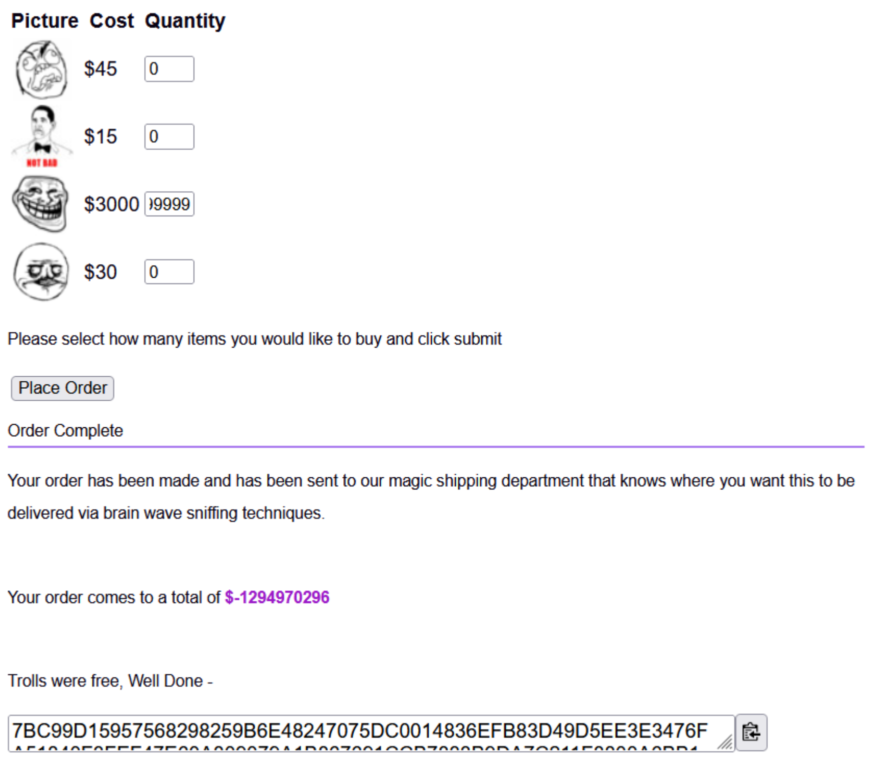
Poor Data Validation occurs when an application does not validate submitted data correctly or sufficiently. Poor Data Validation issues are generally low severity but are more likely to be coupled with other security risks to increase their impact. If all data submitted to an application is validated correctly, security risks are significantly more difficult to exploit. Data from all potentially untrusted sources should be subject to input validation.

Steps to reproduce:

1. Download and run Burp Suite from this link (also make sure you have Oracle Java installed)
2. Using Firefox web browser, set the system proxy to route traffic through Burp: Type in “about:preferences” in the search bar -> Scroll down and click “Settings” underneath Network Settings -> Select option “Manual Proxy Configuration” -> Set HTTP Proxy to 127.0.0.1 and Port to 8080 -> Click Ok.
3. Go to Security Shepherd https://192.168.56.101
4. Confirm that Burp can see and capture requests and turn off intercept in Burp.
5. Go to Challenges -> Poor Data Validation -> Poor Data Validation 2
6. Here we are trying to navigate around the poor data validation measures that are in place by the webpage when ordering “trolls”.
7. Entering an arbitrarily large number might overload the system if it is outside the parameters for which the webpage accepts. We entered 999999 for the $3000 troll.


Challenges

- CSRF
- Failure to Restrict URL Access
- Injection
- Insecure Cryptographic Storage
- Insecure Direct Object References
- Mobile Broken Crypto
- Mobile Content Providers
- Mobile Data Leakage
- Mobile Injection
- Mobile Insecure Data Storage
- Mobile Poor Authentication
- Mobile Reverse Engineering
- Poor Data Validation
  - ✓ Poor Data Validation 1
  - ✗ Poor Data Validation 2
- Security Misconfigurations
- Session Management
- XSS



Picture	Cost	Quantity
	$45	0
	$15	0
	$3000	9999
	$30	0

Please select how many items you would like to buy and click submit

Place Order

Order Complete

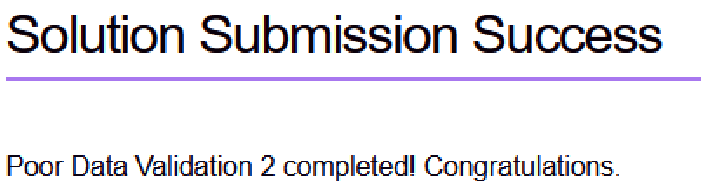
Your order has been made and has been sent to our magic shipping department that knows where you want this to be delivered via brain wave sniffing techniques.

Your order comes to a total of **$-1294970296**

Trolls were free, Well Done -

```
7BC99D15957568298259B6E48247075DC0014836EFB83D49D5EE3E3476F
```

8. This passed the challenge and we were able to buy the trolls for free.


Solution Submission Success

Poor Data Validation 2 completed! Congratulations.

CVSS Score 7.5 (High)

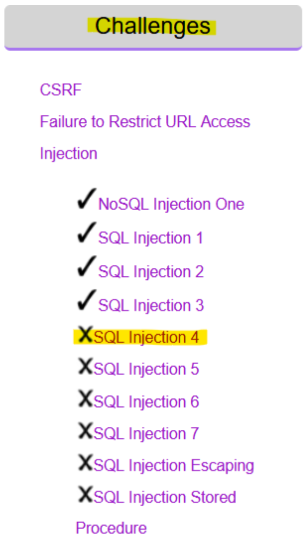
Attack Vector	**Network**
Attack Complexity	**Low**
Privileges Required	**None**
User Interaction	**None**
Scope	**Unchanged**
Confidentiality	**None**
Integrity	**High**
Availability	**None**

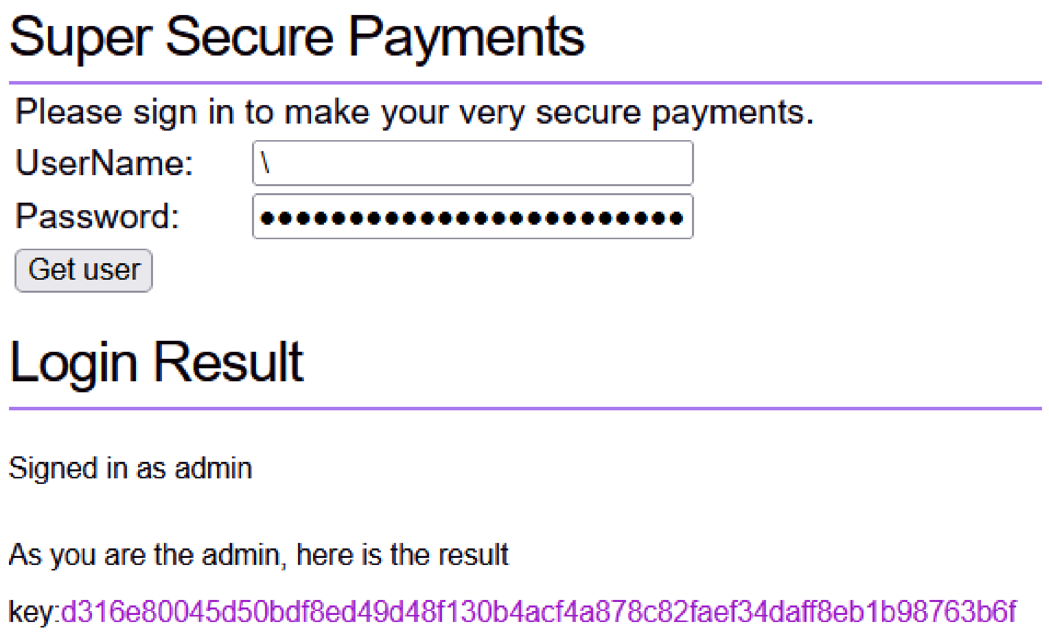
High: SQL Injection Challenge 4 [CWE-89]

SQL Injection occurs when hostile data is sent to the interpreter as part of a command or query. The hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data. SQL Injection attacks are one of the most common web hacking techniques and are of high severity. A successful SQL Injection exploit can read sensitive data from the database, modify the database, execute admin operations, recover file information, and issue commands to the operating system. These attacks are easily exploitable as they can be initiated by anyone who can interact with the system through any data they pass to the application.

Steps to reproduce:

1. Download and run Burp Suite from this link (also make sure you have Oracle Java installed)
2. Using Firefox web browser, set the system proxy to route traffic through Burp: Type in "about:preferences" in the search bar -> Scroll down and click "Settings" underneath Network Settings -> Select option "Manual Proxy Configuration" -> Set HTTP Proxy to 127.0.0.1 and Port to 8080 -> Click Ok.
3. Go to Security Shepherd https://192.168.56.101
4. Confirm that Burp can see and capture requests and turn off intercept in Burp.
5. Go to Challenges -> Injection-> SQL Injection 4
6. This challenge requires us to sign in as an administrator.
7. We're able to input the following to gain admin access:
   - UserName: \
   - Password: or username="admin";-- -
8. The username (\) alters the username/password query in the web application by getting rid of an apostrophe. The password tricks the interpreter into thinking we have administrative access.





Super Secure Payments

Please sign in to make your very secure payments.

UserName: \

Password: ••••••••••••••••••••••••••

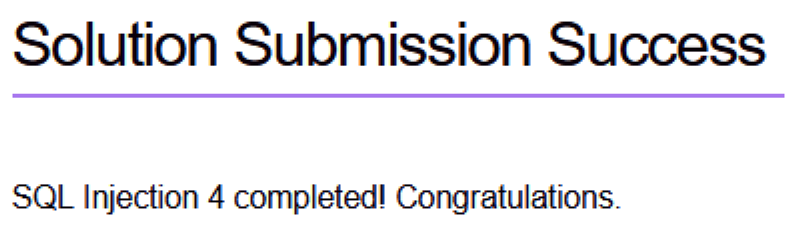
Get user

Login Result

Signed in as admin

As you are the admin, here is the result

key:d316e80045d50bdf8ed49d48f130b4acf4a878c82faef34daff8eb1b98763b6f



Solution Submission Success

SQL Injection 4 completed! Congratulations.

CVSS Score 7.2 (High)

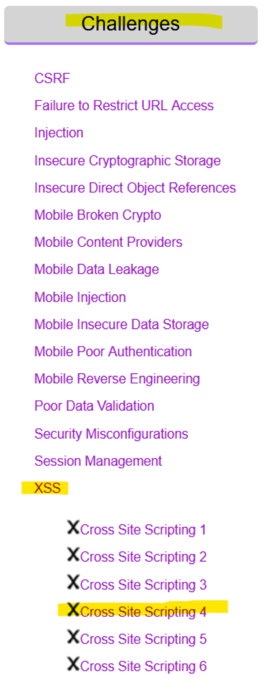
Attack Vector	**Network**
Attack Complexity	**Low**
Privileges Required	**None**
User Interaction	**None**
Scope	**Changed**
Confidentiality	**Low**
Integrity	**Low**
Availability	**None**

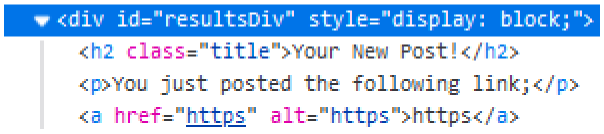
Medium: Cross Site Scripting Challenge 4 [CWE-79]

Cross Site Scripting (XSS) is a type of attack in which malicious scripts are injected into websites or web applications for the purpose of running on the end user's device. These attacks are widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it. If the user has privileged access within the web application, then the attacker might be able to gain full control over all of the application's functionality and data. It is estimated that more than 60% of web applications are susceptible to XSS attacks, which account for more than 30% of all web application attacks. (per source)

Steps to reproduce:

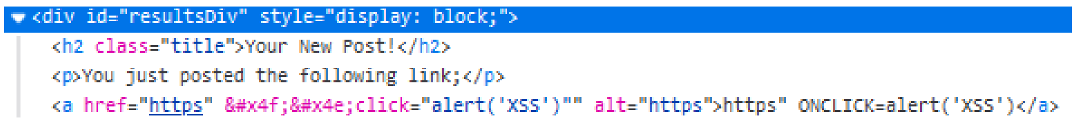
1. Download and run Burp Suite from this link (also make sure you have Oracle Java installed)
2. Using Firefox web browser, set the system proxy to route traffic through Burp: Type in "about:preferences" in the search bar -> Scroll down and click "Settings" underneath Network Settings -> Select option "Manual Proxy Configuration" -> Set HTTP Proxy to 127.0.0.1 and Port to 8080 -> Click Ok.
3. Go to Security Shepherd https://192.168.56.101
4. Confirm that Burp can see and capture requests and turn off intercept in Burp.
5. Go to Challenges -> XSS -> Cross Site Scripting 4.
6. The challenge asks us to enter a URL that we want to post to our public profile. By entering https and opening up the Inspect interface, we found out the website allows a string starting with "https".





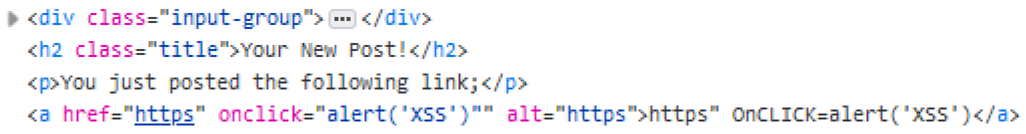
```
▼<div id="resultsDiv" style="display: block;">
    <h2 class="title">Your New Post!</h2>
    <p>You just posted the following link;</p>
    <a href="https" alt="https">https</a>
```

7. Using this information we added "ONCLICK=alert('XSS')" to the previous input. Inspecting the result again we can see that there is an issue with the word "ON" within our attack vector.

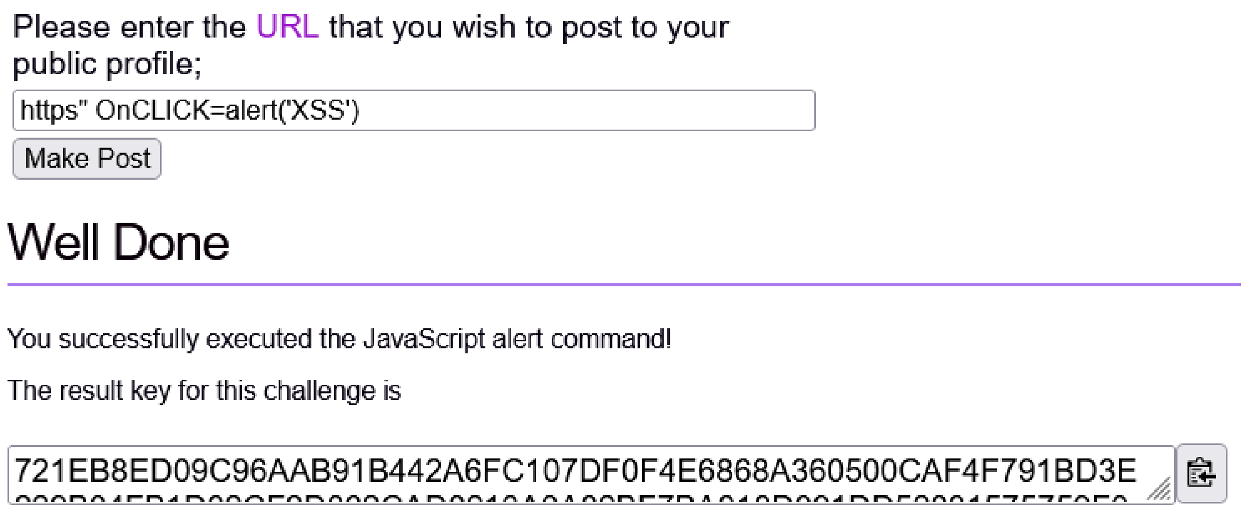


```
▼<div id="resultsDiv" style="display: block;">
    <h2 class="title">Your New Post!</h2>
    <p>You just posted the following link;</p>
    <a href="https" &#x4f;&#x4e;click="alert('XSS')"" alt="https">https" ONCLICK=alert('XSS')</a>
```

8. By changing the case of the "N" in at the start of our attack vector we are able to input: https" OnCLICK=alert('XSS'). This passed the challenge and successfully executed the JavaScript alert command.



```
▶<div class="input-group">⋯</div>
 <h2 class="title">Your New Post!</h2>
 <p>You just posted the following link;</p>
 <a href="https" onclick="alert('XSS')"" alt="https">https" OnCLICK=alert('XSS')</a>
```



Please enter the URL that you wish to post to your public profile;

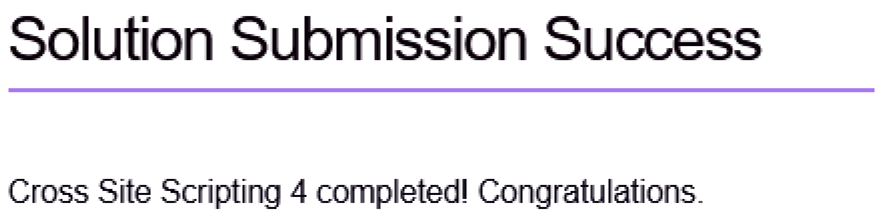
https" OnCLICK=alert('XSS')

Well Done

You successfully executed the JavaScript alert command!

The result key for this challenge is

721EB8ED09C96AAB91B442A6FC107DF0F4E6868A360500CAF4F791BD3E



Solution Submission Success

Cross Site Scripting 4 completed! Congratulations.

CVSS Score 6.4 (Medium)

Attack Vector	**Network**
Attack Complexity	**Low**
Privileges Required	**Low**
User Interaction	**None**
Scope	**Changed**
Confidentiality	**Low**
Integrity	**Low**
Availability	**None**