OCCAM'S RAZOR

ISSN: 1998-0537

Porque lo más sencillo es lo más probable

Número 4 · 1ª Edición · 2009

NMAP. Escaneando Puertos
Exploramos TCP/IP de la mano de esta Mala Bestia

librerías
Capturando Paquetes con LIBPCAP

mú rápido
Monta tu propio sistema de Videovigilancia

reverso
Inyectando paquetes de red. Los Sockets RAW

historia
Historia de la Criptografía

NUEVO

LA CACHARRERÍA
Convierte tu webcam en una cámara IR

tecnología
Reconocimiento Biométrico de Caras

... y nuestras secciones de siempre
- Trucos
- El rincón de los lectores

NO TE CORTES CON LA NAVAJA DE OCCAM

OCCAM'S RAZOR. Número 4. 2009
Porque lo más sencillo es lo más probable

SUMARIO

Occam's Razor

Número 4, Año 2009

Dirección:
David Martínez Oliveira

Editores:
David Martínez Oliveira
Fernando Martín Rodríguez

Colaboradores:
Fernando Martín Rodríguez, David Martínez Oliveira, Snortel, Mappy Porto, Er Boyer, Huakin Paquete, Chinao, Tamariz el de la Perdiz.

Maquetación y Grafismo
DeMO LiR

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

Impresión
Por ahora tu mismo... Si te apetece

Editorial

Seguimos en Marcha

by The Occam Team

Ha pasado más de un año y medio desde el último número de Occam's Razor y muchos de vosotros os habéis estado preguntando cuando llegaría el próximo número. Bien, pues aquí lo tenéis. Como se suele decir más vale tarde que nunca... o no hay tres sin cuatro.

Una revista libre tiene un montón de ventajas. Todos las conocéis, pero tiene una desventaja muy grande. Solo se le puede dedicar el tiempo libre, algo que, en estos días, escasea tanto como el sentido común.

Este último año y medio ha sido un año de grandes cambios para nuestros principales colaboradores. Nuevos trabajos, nuevos países y nuevas criaturas sobre la faz de la tierra. A ellas va dedicado este número.

Grandes cambios que consumen mucho tiempo. Pero, poco a poco, arañando horas a los días hemos conseguido traer a la luz un nuevo número de Occam's Razor. Esta vuestra revista.

En este número os encontraréis menos artículos, pero más extensos. Esperamos que os agrade el formato y sobre todo, vuestros comentarios al respecto.

Además de la secciones habituales (con la excepción de "En la Práctica" que intentaremos que vuelva en el próximo número), estrenamos una nueva sección: "La cacharrería", con la intención de que os guste, tanto como nos ha gustado a nosotros escribirla, y que se acabe convirtiendo en una sección habitual de Occam's Razor.

Otra novedad en este número es la inclusión de hiper enlaces en el documento PDF. Esto fue sugerido por un lector tras la edición del segundo número (gracias Cruz Enrique Borges Hernández) y finalmente lo hemos incorporado. Aprovechando los enlaces, hemos añadido en la mayoría de los artículos un cuadro de recursos en Internet, para que os resulte más sencillo conseguir información adicional.

Como os habíamos anunciado este número es un especial sobre seguridad, donde podréis encontrar artículos sobre criptografía, vídeo vigilancia o reconocimiento biométrico.

A modo de dossier, hemos preparado tres artículos en los que os destripamos los entresijos de las redes TCP desde tres niveles diferentes. Las aplicaciones, las librerías y las llamadas al sistema. Todo lo que necesitáis saber sobre paquetes y protocolos.

Que disfrutéis este número y, como siempre, vuestros comentarios y colaboraciones para seguir mejorando, serán muy bienvenidos.

The Occam's Razor Team

El Rincón de los lectores

Vuestros comentarios, sugerencias,...

por The Occam's Razor Team

Erratas

enviado por Roberto González Cardenete

Hola,

no sé si esta errata os la han comunicado ya. Es en el tercer número, artículo sobre inteligencia artificial y aprendizaje máquina, tabla 3. Para la previsión "lluvioso.^{el} error es 2/5 y para "nublado.^{es} 0/4. En la tabla aparece de forma opuesta, 0/4 para "lluvioso" 2/5 para "nublado.

De igual manera ocurre para el atributo "Hace viento". La regla "SI"tiene un error de 3/6 y la regla "NO"de 2/8.

Enhorabuena por la revista y muchas gracias: es un placer leerla.

Un saludo,

–

Muchas gracias Roberto por apuntarnos esta errata. Ya le hemos incluídos estos cambios en la última edición.

Otra Utilidad de vim

enviado por Luis Rodríguez

Se os ha olvidado comentar, como utilidad para programadores, que si presionamos la tecla "%.^{en} modo comando sobre un {,(o [, el cursor se nos moverá automáticamente hacia el correspondiente },),], lo cual es tremendamente útil a la hora de depurar programas que utilizan esta sintaxis de llaves, paréntesis o corchetes para anidar estructuras de control, ya que un ítem de estos sin emparejar no se moverá.

Un saludo,

–

Totalmente de acuerdo, una muy útil funcionalidad de vim. Como os comentábamos en el artículo, hay cientos de interesantes comandos ofrecidos por vim, como este que nos comenta nuestro amigo Luis. Simplemente ejecutad el comando help (ya sabéis ESC : help) y a leer!.

Fuentes LaTeX

enviado por muchos de vosotros

Muchos de vosotros nos habéis escrito solicitando las fuentes de todos los números de Occam's Razor. Eso es lo que decíamos en la web. Como ya sabréis tenemos algunos problemas de espacio y hasta ahora solo nos ha sido posible mantener las fuentes de la última revista, puesto que los ficheros de fuentes son bastante voluminosos.

Queremos agradecer a todos los que os habéis ofrecido para hacer un mirror de estos ficheros y comunicaros que a partir de este número todos los ficheros estarán accesibles para descargar.

Todos los que todavía estéis interesados en hacer un mirror de estos ficheros podéis poneros en contacto con nosotros de nuevo (para que no se nos despiste nadie entre todos los mails archivados) y en la página oficial de la revista (`http://webs.uvigo.es/occams-razor`)mantendremos una lista con todos estos mirrors y el tipo de acceso que proporciona.

Felicitaciones

enviado por muchos de vosotros

Seguimos recibiendo y agradeciendo todas vuestras felicitaciones que nos animan a continuar con esta vuestra revista, aunque a veces sea un poco a trompicones. Intentamos que esta sección, al igual que todas las demás de la revista os resulte útil y por esa razón no incluimos todos estos mensajes, que aunque muy necesarios para nosotros.

En cualquier caso.

MUCHAS GRACIAS POR VUESTRO APOYO!!

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!!!!

libpcap. Tu propio sniffer en 5 líneas

... bueno, 5 líneas y un call-back

por Snortel

Capturar paquetes es una de esas cosas "Sinatra", vamos, que cada sistema hace "a su manera". Afortunadamente para nosotros algunas personas se han preocupado de escribir una librería para poder llevar a cabo esta tarea de una forma portable. Estamos hablando de la librería libpcap.

Una de las herramientas básicas relacionadas con la seguridad informática es el sniffer o capturador de paquetes o analizador de protocolos, o... Bueno, tienen muchos nombres, dependiendo del uso que se le quiera dar.

Estos programas normalmente utilizan lo que se conoce como "sockets RAW", los cuales, desafortunadamente, tienen un interfaz diferente en cada sistema operativo. La librería libpcap nos permite utilizar esta funcionalidad independientemente del sistema operativo.

EL CÓDIGO

Sin más preámbulos vamos a ver el código de nuestro sniffer de cinco líneas y un callback utilizando libpcap.

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

static int count = 1;

void ip_cb(u_char *args,
           const struct pcap_pkthdr* ph,
           const u_char* p) {
  printf ("Hemos recibido %d paquetes!!!. "
          "Ha, ha, ha!!!!"
          "%d paquetes!!!\n", count++, count);
}
int main(int argc,char **argv) {
  char                  err[PCAP_ERRBUF_SIZE];
  pcap_t*               h;
  struct bpf_program    fp;
  bpf_u_int32           maskp;
  bpf_u_int32           netp;
  u_char*               args = NULL;

  pcap_lookupnet ("eth0", &netp, &maskp, err);
  h = pcap_open_live("eth0",BUFSIZ,1,-1,err);
  if(argc > 1) {
    pcap_compile (h, &fp, argv[1], 0, netp);
    pcap_setfilter (h, &fp);
  }
  /* Cuenta!!!, Cuenta!!! Maldito!!!
     Ha, ha,ha !!!! */
  pcap_loop (h, -1, ip_cb, args);
  return 0;
}
```

Vale, no son exactamente cinco líneas, pero sí cinco llamadas a funciones de libpcap. Aunque aparentemente este programa es muy tonto (cuenta paquetes en el interfaz eth0), enseguida veréis que es mucho más potente de lo que parece.

Lo primero que aparece es nuestro callback. Esta es la función que libpcap ejecuta cada vez que captura un paquete. En nuestro caso, esta función utiliza una variable estática para llevar la cuenta de los paquetes capturados.

En el programa principal es donde está lo interesante.

CONFIGURANDO LA CAPTURA

El programa comienza obteniendo información sobre la configuración del interfaz eth0. Esta llamada os la podéis ahorrar si no vais a trabajar con paquetes de "broadcast" ya que su única finalidad es obtener la máscara de red asociada con el interfaz.

A continuación le decimos a libpcap que queremos capturar paquetes "en directo" en el interfaz "eth0". De especial interés es el tercer parámetro. Este parámetro le dice a libpcap que intente configurar el interfaz de red en modo promiscuo, es decir, la tarjeta de red informará al sistema operativo de todos los paquetes que ve, no solo de los paquetes dirigidos a ella.

En estos momentos ya estamos en condiciones de capturar paquetes, y la sentencia if (y la siguiente sección) que sigue a continuación os la podéis ahorrar si no queréis utilizar el filtrado de paquetes que ofrece la librería.

FILTRANDO PAQUETES

Como os decíamos, la parte central del programa utiliza el sistema de filtrado de paquetes proporcionado por libpcap. En nuestro ejemplo, solamente activamos el filtro si hemos recibido algún parámetro a través de la línea de comandos (argc mayor que 1).

El sistema de filtrado de paquetes de libpcap es muy potente

Como podéis ver,simplemente pasamos el primer parámetro del programa a la función `pcap_compile`.

Y a continuación asignamos el nuevo filtro a nuestro manejador pcap.

Supongo que ahora os estaréis preguntando . . . ¿Y cómo son esos filtros? Bueno, pues para tener una descripción detallada de como definirlos solo tenéis que consultar la página del manual del programa tcpdump. Sí, tcpdump, el capturador de paquetes por excelencia utiliza libpcap :).

EL BUCLE PRINCIPAL

Finalmente, utilizamos la función `pcap_loop` para iniciar la captura. El programa entrará en un bucle infinito en el que, cada vez que capture un paquete, ejecutará nuestra función `ip_cb` pasándole como primer parámetro el paquete capturado (bueno, pasa algo más, pero en esencia eso es lo que hace). El segundo parámetro de `pcap_loop`, nos permite especificar el número de paquetes que queremos capturar. En este caso, el valor -1 indica que no hay límite.

PROBANDO NUESTRO SNIFFER

Pues ya solo nos queda probar que todo funciona. Lo primero que haremos es compilar nuestro programa de la forma habitual.

```
$ gcc -o count_packet count_packet.c -lpcap
```

Como en nuestro programa, hemos decidido poner el interfaz de red en modo promiscuo, tendremos que ejecutar nuestro sniffer como usuario root. Para probar el programa, vamos además a utilizar un sencillo filtro y la utilidad ping (el parámetro -c2 le dice a ping que envíe solo dos paquetes).

```
$ ping -c2 una_ip
```

Esto es lo que obtendremos, al ejecutar nuestro programa en la misma máquina.

```
# ./count_packet icmp
Hemos recibido 1 paquetes!.Ha, ha, ha! 1 paquetes!
Hemos recibido 2 paquetes!.Ha, ha, ha! 2 paquetes!
Hemos recibido 1 paquetes!.Ha, ha, ha! 3 paquetes!
Hemos recibido 2 paquetes!.Ha, ha, ha! 4 paquetes!
```

Como podéis ver, estamos capturando el paquete ICMP `ECHO` que ping envía y el paquete ICMP `ECHO-REPLY` que recibimos de la máquina remota. Observad que si la ip que habéis utilizado no está accesible, solamente recibiréis dos paquetes ICMP `HOST_UNREACH` y, obviamente, ninguna respuesta del sistema remoto (que no existe).

Probemos ahora algo más elaborado.

```
# ./count_packet 'icmp[icmptype] != icmp-echo'
Hemos recibido 1 paquetes!.Ha, ha, ha! 1 paquetes!
Hemos recibido 2 paquetes!.Ha, ha, ha! 2 paquetes!
```

Si volvemos ha ejecutar nuestro ping... veremos que ahora nuestro programa solo muestra dos paquetes... claro, en nuestro filtro estamos diciendo que solo nos interesan los paquetes `ECHO-REPLY` y no los `ECHO`.

¿Qué obtendríais ahora haciendo ping a una máquina que no existe?

UN CALLBACK MÁS INTERESANTE

Nuestro conde contador es güay, pero no resulta demasiado útil. Como decíamos más arriba, el callback que pasemos como parámetro a `pcap_loop` recibirá toda la información asociada al paquete que se ha capturado.

Aquí tenéis uno un poco más interesante.

```
#include <netinet/ether.h>
#include <netinet/ip.h>

void ip_cb(u_char *args,const struct pcap_pkthdr* ph,
           const u_char* p) {
  struct ip   *ip_pqt;

  ip_pqt = (struct ip*) (packet +
                         sizeof(struct ether_header));

  printf ("[%16s]->", inet_ntoa(ip_pqt->ip_src));
  printf ("[%16s]: (TTL:%03d proto: %02d) ",
          inet_ntoa(ip_pqt->ip_dst),
          ip_pqt->ip_ttl, ip_pqt->ip_p);
}
```

Lo primero que podéis ver es un par de ficheros de cabecera adicionales. Esto ficheros contienen la definición de las estructuras de datos asociadas a los paquetes de red que vamos a capturar.

La función simplemente muestra alguna información general asociada al nivel IP: las direcciones IP de origen y destino del paquete, el TTL (Time To Live) y el protocolo asociado. Como podéis ver, lo primero que hacemos es saltarnos la cabecera ethernet para poder acceder a los datos IP.

Para escribir un sniffer personalizado, solo necesitáis hacer 5 llamadas a la librería

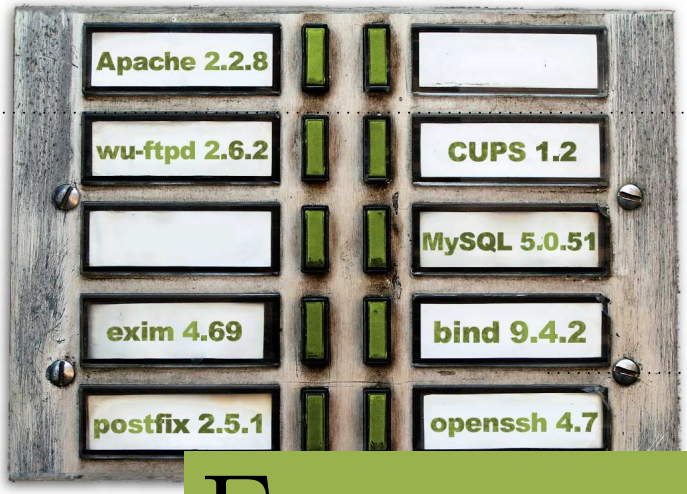
Una explicación detallada de como funciona una pila de protocolos se sale de los objetivos de este pequeño artículo, pero jugando con libpcap, un par de buenos libros y un puñado de RFC, los protocolos de red no tendrán secretos para vosotros.

Como siempre, recordad que en estos ejemplos, por cuestiones de espacio, hemos eliminado todas las comprobaciones de error. Vosotros no debéis hacerlo.

En el caso concreto de este último ejemplo, es posible que obtengáis algunos datos "raros" durante vuestras pruebas... No estamos comprobando si el paquete es realmente un paquete IP, no estamos comprobando la versión del protocolo (v4 o v6), ni si se trata de un paquete completo o un fragmento.

Si tenéis mucha curiosidad, una buena idea es echarle un ojo al código fuente de tcpdump o wireshark. Ambos utilizan libpcap.

Hasta el próximo número. ■



Escaneo de Puertos con nmap

Desvelamos los secretos de TCP/IP

por Mappy Porto

En cualquier ataque informático, el primer paso es siempre la obtención de información sobre la víctima. Existen distintas formas de llevar a cabo esta tarea, dependiendo del objetivo y el tipo de ataque que se desea realizar, pero si estamos hablando de ataques remotos, el análisis de los servicios ofrecidos por la máquina víctima, suele ser obligatorio. Este proceso se conoce normalmente como mapeado o escaneado de puertos. De la mano de nmap, quizás el escáner de puertos más conocido y completo, vamos a introducirnos en los detalles más técnicos de esta proceso.

Antes de meternos de lleno en el tema vamos a refrescar algunos de los conceptos generales que manejaremos a lo largo de este artículo. Y el primer concepto que se nos viene a la cabeza, es el del mar... bueno, navegar, barco, puerto... eso, puerto :)

Lo primero que tenemos que saber es que los puertos están asociados a lo que se llama la capa de transporte y lo que en el mundo TCP-IP equivale a la capa en la que funcionan los protocolos TCP y UDP. Justo bajo la capa de transporte está la capa de red (según la nomenclatura OSI), que "en el mundo real" se corresponde con la capa que ocupa el protocolo IP (entre otros).

El primer paso de cualquier ataque es obtener información

El principal problema que aborda la capa de red es el del enrutamiento, es decir, como hacer que ciertos datos que salen de un ordenador lleguen a otro ordenador en la otra punta del mundo. Para resolver este problema, lo que necesitamos, como mínimo, es una forma de identificar ambas máquinas, y esa es la función de las direcciones IP.

Así que, aquí tenemos el primer concepto que tiene que estar claro. Las direcciones IP están asociadas a máquinas. Punto. Para la mayoría estará claro, pero en una ocasión, leí en un "manual de hackers" una frase que decía algo así como: "enviamos un paquete ICMP al puerto tal de la máquina".

ICMP es un protocolo que, aunque utiliza IP para enviar los datos, se encuentra en la capa de red... es decir, está al lado del IP, no encima como TCP o UDP, y por tanto, el concepto de puerto no existe. Muy pronto entenderemos que es lo que pretendían decir en ese "manual de hackers".

LOS PUERTOS

Los puertos se introducen en la capa de transporte (TCP y UDP) y su finalidad es la de permitir establecer distintos puntos de conexión, en una misma máquina. Así una determinada máquina con una determinada dirección IP puede ofrecer distintos servicios; un servicio de correo, un servicio web, un servicio ftp, etc...

Los puertos se introducen en la capa de transporte

Para poder referenciar cada uno de estos servicios, utilizamos un puerto distinto, o dicho de otra forma, cualquier servicio ofrecido por una máquina se puede identificar a partir de su dirección IP y su puerto asociado.

Podemos verlo como una dirección postal. La dirección IP sería el equivalente a la ciudad, calle y número dentro de esa calle. Esos datos nos identificarían un edificio en el que vive un montón de gente. El botón del portero automático en la puerta del edificio sería el equivalente al puerto. Dependiendo del botón que pulsemos (el puerto al que accedamos), obtendremos una respuesta u otra.

ESCANEO DE PUERTOS

Bien, pues tras este rollo, ¿qué es un escáner de puertos?. Pues, siguiendo nuestro símil, un escáner de puertos es un programa capaz de visitar un montón de edificios muy rápido y pulsar todos los botones del telefonillo, para luego decirnos cuantos apartamentos de ese edificio están ocupados y cuantos están vacíos o simplemente no contestan.

Aunque volveremos sobre esto más tarde... ¿Alguien puede imaginar quién sería el cortafuegos en nuestro ejemplo?.... nadie?... venga si es muy fácil... Exacto, un portero físico. Cuando llegas al edificio, en lugar de encontrarte con el telefonillo y empezar a pulsar botones, lo que te encuentras es un señor que te pregunta a que piso vas.

Si los ocupantes del piso no quieren ser molestados le dirán al portero que no deje pasar a nadie, y no sabremos si realmente hay alguien en ese piso o no. Si el portero nos conoce, o estamos en la lista de "amigos", pues podremos pasar directamente, sin decir más que un simple hola... y eso por que somos educados.

Los escáneres de puertos como nmap, son capaces de proporcionarnos esta información, y alguna más. Por ejemplo, nos dirá si la dirección existe o no (si la máquina está encendida), si se trata de un edificio de oficinas o de apartamentos (el sistema operativo) y si la persona que responde al telefonillo es un hombre, una mujer, un niño, un francés, un inglés o un alemán (el servicio que está corriendo y con suerte el programa que proporciona ese servicio y su versión).

Vamos, una herramienta muy útil.

LA IMPORTANCIA DEL ESCANEO

Para terminar con esta introducción y meternos a saco con el tema, vamos a explicaros porque determinar que puertos están "abiertos" en una máquina es importante para un intruso.

En nuestro ejemplo, hemos dicho que los apartamentos de nuestro edificio son los servicios de la máquina que está siendo escaneada. Si el objetivo de un atacante es tomar el control de la máquina, es decir, entrar en el edificio, pues la forma de entrar será por una puerta o por una ventana.

Si el intruso decide entrar por el séptimo C, y resulta que el séptimo C no existe en ese edificio, pues difícilmente va a poder hacer nada. Cuando llegue al edificio y vea que no hay séptimo C, pues se tendrá que dar vuelta.

En el mundo de los ordenadores, lo que ocurre es que la mayoría de la gente tiene el mismo tipo de apartamentos... la misma cerradura y el mismo tipo de ventanas, de forma que si sabes abrir una de esas cerraduras, podrás entrar en cualquier apartamento de cualquier tipo de cualquier edificio... suponiendo que el edificio no tenga portero o puedas engañarlo :).

Estos tipos de apartamentos son los servicios de la máquina. Así nos encontraremos apartamentos del tipo Apache 1.0, Sendmail 8 o vsftp 3.4. Lo que le interesa al intruso es encontrar esos pisos y saber de que tipo son, todo ello de la forma más sigilosa posible. Cuando encuentre una que conozca y sepa como abrir la cerradura podrá entrar.

Un escaner de puertos determina que servicios corre una máquina

Bien, pues para eso es un escáner de puertos. Obviamente, la comunidad del edificio puede utilizar esa misma herramienta para comprobar si su edificio es seguro, si tiene que poner otro portero, o restringir la entrada y salida de gente.

Finalmente, aunque no vamos a tratar ese tema en este artículo (es otra historia como decía Conan), que un edificio no tenga ni puertas ni ventanas significaría que está desconectado de la red y en ese caso poco se puede hacer (entraríamos en el mundo de la seguridad física). Pero que el portero no deje entrar a nadie y solo deje salir, eso sería un edificio con una única puerta por la que solo pueden salir cosas... pues bueno... si hay una salida, hay una entrada :).

FUNDAMENTOS

El proceso de escaneo de puertos se basa fundamentalmente en dos técnicas. La primera es el análisis de paquetes ICMP, y la segunda es el análisis de los paquetes de la capa de transporte (TCP o UDP), en condiciones atípicas. Veamos en que consiste cada una de ellas.

El análisis de paquetes ICMP es un elemento básico del escaneo de puertos

El protocolo ICMP, como ya os hemos comentado, se encuentra al mismo nivel que el protocolo IP. ICMP es el acrónimo de *Internet Control Message Protocol*, es decir, protocolo de mensajes de control inter-redes. Y eso es precisamente lo que hace, enviar y recibir mensajes de control. Los que queráis ver la lista completa de todos esos mensajes, podéis consultarlos en el RFC 792 que describe este protocolo.

De todos los mensajes que podemos enviar con ICMP, hay unos cuantos que son de especial interés desde el punto de vista del escaneo de puertos. Estos mensajes pertenecen al grupo de "Destino Inalcanzable" representados por el código número 3. Ciertos mensajes ICMP, como los que estamos analizando, requieren de un subcódigo que refina la información proporcionada por el mensaje. Para el grupo que nos interesa, los siguientes subcódigos nos resultarán muy útiles.

- Red inalcanzable (subcódigo 0)
- Máquina inalcanzable (subcódigo 1)
- Protocolo inalcanzable/inexistente (subcódigo 2)
- Puerto inalcanzable (subcódigo 3)

Como podéis imaginar, el mensaje que más nos interesa es el último. Los otros los hemos incluido, puesto que nmap proporciona otras técnicas de escaneo que hacen uso de ellos.

Bueno, pues como funciona todo esto. Los mensajes ICMP son generados por las capas inferiores. Dependiendo del tipo de mensaje, este puede generarse localmente (como en el caso de la red inalcanzable -también se puede generar remotamente-), o remotamente, como en el caso del protocolo inalcanzable.

Si intentamos conectarnos a un determinado puerto de una máquina remota, y ese puerto no existe (ningún proceso lo está utilizando), la máquina remota generará un menaje ICMP para indicar esta situación, de forma que la máquina intentando conectar no tiene que esperar un "tiempo prudencial" para recibir una respuesta. Observad que la respuesta puede tardar porque la red esté congestionada o porque realmente no ha habido respuesta.

Volveremos sobre este concepto cuando hablemos del protocolo UDP.

PAQUETES TCP BÁSICOS

El otro elemento en el que se fundamentan los procesos de escaneo de puertos es en el análisis de los paquetes TCP. Cada vez que nos conectamos, por ejemplo, a una página web, nuestro ordenador envía a través de la red una serie de bloques de datos conteniendo cierta información, y recibe otros bloques de datos con la página web solicitada, y alguna información más.

Esa información que viaja por la red, debe seguir un determinado formato, lo que se conoce como el formato de paquete TCP. En general, todos estos protocolos funcionan de la misma forma.

Toman los datos de una capa superior que en última instancia es el usuario, proporcionando el nombre de una página web (en nuestro ejemplo), y a esos datos les añade una cabecera. El nuevo bloque de datos se pasa al nivel inferior, y así sucesivamente, hasta que el bloque de datos (el paquete) alcanza el último nivel y es enviado a través de nuestro interfaz de red.

En la siguiente figura, podéis ver el formato de un paquete TCP. La parte final, es la que contendrá los datos que queremos enviar o recibir (nuestro mensaje de correo, o la página web que deseamos ver). El resto de campos son añadidos por el sistema, y están ahí para permitir la comunicación fiable entre dos máquinas conectadas a través de una red (eso es lo que hacen los protocolos de transporte como TCP:).

Bit offset	Bits 0–3	4–7	8–15	16–31
0	Source port			Destination port
32	Sequence number			
64	Acknowledgment number			
96	Data offset	Reserved	CWR ECE URG ACK PSH RST SYN FIN	Window Size
128	Checksum			Urgent pointer
160	Options (optional)			
160/192+	Data			

Paquete TCP (Fuente: Wikipedia)

Dependiendo de la operación que deseemos realizar, las máquinas intercambiarán distintos tipos de paquetes. El tipo de paquete lo determina el campo flags. En la discusión que nos ocupa, nosotros estamos interesados especialmente en tres de ellos:

- `SYN`. Este es el tipo de paquete utilizado para establecer conexiones. Nos vamos a aburrir de él a lo largo del artículo :).
- `ACK`. Este es el paquete que una máquina envía a la otra para comunicarle que ha recibido ciertos datos correctamente.
- `RST`. Cuando sucede algo que no debería pasar, la pila TCP envía uno de estos paquetes, para re-iniciar la comunicación.

Los paquetes SYN, ACK y RST son los que más nos interesan

Pues bien, ya estamos en condiciones de empezar a jugar con nmap, sabiendo que es lo que estamos haciendo :).

ESCANEO DE PUERTOS RESUMIDO

Con todo lo que hemos comentado hasta el momento, estamos en condiciones de presentaros las reglas básicas utilizadas para el escaneo de puertos. Aunque veremos que nmap nos proporciona un montón de opciones (y sino lo podéis ver en su página del manual; man nmap), el fundamento de todas ellas es más o menos el mismo.

- Regla 1. Si enviamos un paquete a una máquina que no existe, recibiremos un paquete ICMP indicándolo.
- Regla 2. Si enviamos un paquete a un puerto no utilizado, recibiremos un paquete ICMP indicándolo (con UDP) o un paquete `RST` (con TCP).
- Regla 3. Si enviamos un paquete "ilegal", en general, recibiremos un paquete `RST` como respuesta.

Sí, así de tonto. Estas tres reglas son la base de cualquier escaneo de puertos. En el resto del artículo, veremos como nmap las aplica de diferentes maneras para solucionar diferentes problemas.

DESCUBRIENDO MÁQUINAS

Si has llegado hasta aquí, espero que tengas claro lo que es una dirección IP, un puerto, y cual es la finalidad de los escáneres de puertos. A partir de aquí, y con la ayuda de nmap, vamos a entrar en detalle sobre cómo llamar a los telefonillos, evitar porteros y hacer todo esto sin que se note :)

El primer paso, antes de llevar a cabo ningún tipo de acción más complicada, es determinar si la máquina a analizar existe y está encendida. Este proceso se suele extender a analizar rangos de direcciones IPs, ya que quizás la máquina que interesa al intruso esté muy protegida y el acceso desde fuera de la subred sea difícil, pero quizás otra máquina, en esa misma subred, no lo está tanto y accediendo a ésta, resulte más sencillo alcanzar el objetivo final.

Siguiendo con nuestro ejemplo. El edificio que nos interesa está muy protegido, con un portero que es un hueso, sin ventanas que den a la calle, etc... Sin embargo, tiene ventanas y puertas en un patio interior compartido por otros edificios adyacentes. Si alguno de esos edificios cercanos no está tan protegido, el intruso podrá entrar primero en uno de ellos y luego pasar fácilmente al edificio objetivo a través del patio interior que los comunica.

En otras ocasiones, el intruso simplemente está interesado en localizar máquinas vulnerables que comprometer, para luego utilizarlas en un ataque de tipo DDoS o utilizarla como parte de una cadena de "proxies" para realizar un ataque a cualquier otra máquina.

Así, lo primero que tenemos que averiguar es cómo saber si una determinada dirección IP está asociada a una máquina. Nmap nos proporciona varias formas de averiguarlo.

ESCANEO PING

La primera forma es el denominado *ping scan*, que básicamente hace lo mismo que la herramienta ping. ¿Y cómo funciona ping? os preguntaréis. Bueno, ping, compone un paquete `ICMP ECHO` (código 8), al cual, la máquina destino responderá con otro paquete `ICMP ECHOREPLY` (código 0). Esto os debería resultar familiar.

Así es como está definido el protocolo ICMP, y esto funciona a no ser que un cortafuegos filtre los "pings".

El escaneo PING nos permite saber si una máquina existe

Para poder enviar un paquete ICMP, al igual que la mayoría de los paquetes especiales que veremos a lo largo del artículo, nmap requiere permisos de root puesto que debe utilizar un socket RAW para componer el paquete específico para cada prueba. De la misma forma para capturar el paquete `ECHOREPLY` de respuesta, nmap utiliza una versión modificada de libpcap que se distribuye con sus fuentes. No se pueden enviar y recibir paquetes IP o ICMP (recordad que están al mismo nivel), utilizando sockets "normales".

La opción -sP es la que permite llevar a cabo esta operación. Esta opción, además envía un paquete TCP al puerto 80, como prueba adicional (es común filtrar paquetes ICMP, pero no el servicio web).

ENTORNO DE PRUEBAS

En este momento ya estamos en condiciones de empezar a hacer algunas pruebas para ver como nmap y la pila TCP-IP funcionan. El primer paso para ello es establecer un entorno de pruebas. Tendréis que utilizar una segunda máquina, de forma que no se utilice el interfaz de loopback y los paquetes puedan ser capturados con un sniffer. Por ejemplo, podéis utilizar vuestro router ADSL como máquina de test o montar una red virtual con qemu.

Para nuestras pruebas, lanzaremos por un lado un sniffer. Podéis utilizar el que más rabia os dé. Wireshark (el sniffer anteriormente conocido como Ethereal... vamos el Prince de los sniffers :)) tiene un interfaz gráfico muy sencillo con el que resulta muy fácil ver los resultados de las pruebas. Nosotros vamos a utilizar tcpdump... es que somos asín.

En otra consola lanzamos nuestro escaneo ping contra la máquina de prueba, obteniendo algo similar a lo que nos muestra la Figura 1. La Figura 2 muestra lo que tcpdump capturaría.

PING SCAN EXPLICADO

Bien, sin grandes sorpresas verdad?. Sin embargo tenemos que hacer algunos comentarios. El primero es relativo al uso del flag `--send-ip`. Este flag obliga a nmap a usar paquetes IP. Si no utilizamos este flag, al encontrarnos en una red local y ser un usuario privilegiado, nmap trata de utilizar el protocolo ARP para determinar si la máquina está funcionando.

El segundo comentario es relativo a por qué enviar un paquete TCP al puerto 80 para determinar si una máquina está levantada. Si os fijáis en el log de tcpdump, la máquina razor, responde con un paquete TCP `RST` (la R que aparece en primer lugar). Esto sucede porque lo que nmap envió al puerto 80 de la máquina remota es un paquete `ACK`.

FIGURA 1 - ESCANEO PING

```
occams # nmap --send-ip -sP razor
Starting Nmap 4.10 ( http://www.insecure.org/nmap/ ) at 2008-07-19 19:24 BST
Host razor (192.168.1.1) appears to be up.
MAC Address: 00:18:F8:67:FB:0B (Unknown)
Nmap finished: 1 IP address (1 host up) scanned in 0.254 seconds
```

FIGURA 2 - CAPTURA TCPDUMP

```
# tcpdump 'icmp ||  (tcp port 80) '
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
19:28:52.506536 IP occams > razor: ICMP echo request, id 38285, seq 46094, length 8
19:28:52.506809 IP occams.35381 > razor.www: . ack 42183966 win 2048
19:28:52.507046 IP razor > occams: ICMP echo reply, id 38285, seq 46094, length 8
19:28:52.507504 IP razor.www > occams.35381: R 42183966:42183966(0) win 0
```

FIGURA 3 - ESCANEO CONNECT

```
# nmap --send-ip -sP test

Starting Nmap 4.10 ( http://www.insecure.org/nmap/ ) at 2008-07-19 19:37 BST
Note: Host seems down. If it is really up, but blocking our ping probes, try -P0
Nmap finished: 1 IP address (0 hosts up) scanned in 0.140 seconds
```

FIGURA 4 - CAPTURA TCPDUMP DE ESCANEO CONNECT

```
19:37:21.873327 IP occams > test: ICMP echo request, id 34080, seq 40155, length 8
19:37:21.873361 IP occams.37620 > test.www: . ack 418834206 win 3072
19:37:21.873889 IP razor > occams: ICMP net test unreachable, length 36
19:37:21.874174 IP razor > occams: ICMP net test unreachable, length 48
```

Ese paquete no se debería recibir si no existe una conexión establecida previamente y por lo tanto el sistema remoto responde con una petición de RESET (paquete `RST`). Si la máquina estuviera apagada, obviamente no respondería nada. Veámoslo.

Test es una máquina que no existe. Para evitar mensajes extra, añadid una entrada en el fichero `/etc/hosts` de forma que no sea necesaria una petición al DNS, o utilizad el flag `-n`.

Los resultados los podéis ver en las Figuras 3 y 4.

Como podemos ver, nmap sigue enviando los dos paquetes (`ECHO` y TCP `ACK`), igual que antes, pero en este caso, nuestro router nos devuelve un mensaje ICMP indicando que la red en la que se encontraría test es inalcanzable... Obviamente, cuando añadáis test a /etc/hosts usad una IP que este fuera de vuestra red.

Para explicar esto en detalle tendríamos que entrar en como se lleva a cabo el encaminamiento y algunas cosas más, que ya salen fuera del objetivo de este artículo. Por el momento simplemente quedaros con que alguien es capaz de detectar si una red es inalcanzable y que nos lo comunicará con un mensaje ICMP.

MÁS TÉCNICAS

Hasta ahora hemos visto la forma estándar de determinar si un host está levantado, pero nmap nos proporciona una serie de opciones alternativas para llevar a cabo esta tarea. El uso de una u otra dependerá del caso particular de la máquina que se esté escaneando. Por ejemplo, si la máquina está tras un firewall, los paquetes `ICMP ECHO` y los intentos de conexión hacia la red interna, probablemente sean filtrados, sin embargo el firewall debe dejar pasar los paquetes `ACK` (para poder permitir la comunicación desde dentro). Cada caso es particular y esa es la razón porque son necesarias algunas herramientas más que el escáner de puertos :).

Todas las técnicas para detectar si una máquina existe se basan en hacer un ping, o dicho de otra forma, en enviarles algo que provoque algún tipo de respuesta. Recibir una respuesta implica que la máquina existe.

Así, nmap proporciona distintas técnicas para hacer pings a máquinas. La página del manual de nmap contiene un montón de información sobre cuando interesa utilizar una u otra. Para no reproducirla aquí, nosotros solo os proporcionamos un pequeño resumen sobre qué "ping" hace cada una de ellas.

- `-PS`. Ping con paquete TCP `SYN`. Podemos recibir un paquete `RST` (puerto cerrado), `ACK` (puerto abierto) o un error ICMP (máquina no existe).
- `-PA`. Ping con paquete TCP `ACK`. Podemos recibir un paquete `RST` (puerto abierto/máquina existe) o un error ICMP (máquina no existe).
- `-PU`. Ping con paquete UDP. Podemos recibir un error ICMP "puerto inalcanzable"(máquina existe) u otro error ICMP (máquina no existe). Este tipo de escaneo puede ser muy lento.
- `-PE`, `-PP`, `-PM`. Ping con paquetes ICMP (`ECHO`, `TIMESTAMP`, `ADDRESSMASK`). Una respuesta a estos mensajes indica que la máquina existe. La falta de respuesta indica que la máquina no existe o el firewall ha filtrado los paquetes.
- `-PO`. No ping. Utilizada para no comprobar si la máquina existe :).
- `-PE`. Ping ARP. Esta técnica solo funciona en redes LAN y hace uso del protocolo ARP.

Este último escaneo solo funciona en redes de área local y es el que utiliza nmap por defecto cuando se trata de escaner dirección locales. La ventaja frente a las otras alternativas es que; es mucho más rápido y mucho más fiable. En la Figura 5 podemos ver un pequeño ejemplo.

FIGURA 5 - ESCANEO CARP

```
# nmap -PR razor
....
# tcpdump
11:18:46.760693 arp who-has razor (Broadcast) tell occams
11:18:46.760974 arp reply razor is-at 00:18:f8:67:fb:0b (oui Unknown)
```

En este caso, nmap mantiene una lista de direcciones MAC y su fabricante asociado, siendo capaz, en algunos casos, de informarnos del fabricante de la máquina que estamos escaneando.

Observad que, con la excepción de los escaneos UDP, el no recibir ningún tipo de respuesta, en general significará que hemos topado con un firewall con una regla `DROP` por el camino. Si el firewall utiliza una regla `REJECT`, recibiríamos paquetes `RST`.

ESCANEO CONNECT

Hasta aquí, hemos visto como utilizar nmap para localizar máquinas activas. Una vez que el intruso sabe que la máquina está activa, es decir, existe, es el momento de averiguar toda la información posible sobre ella.

La principal función de nmap es informar de qué puertos en la máquina víctima están abiertos, pero, como ya adelantamos, también es capaz de proporcionar un montón de información adicional muy útil. Todo esto lo veremos a continuación, pero antes de continuar vamos a introducir algunos conceptos más para que os resulte más fácil de entener el resto del artículo.

La forma más sencilla de escaneo consiste en intentar una conexión

La técnica de escaneo más sencilla que existe consiste simplemente en intentar una conexión a un determinado puerto. Vimos como hacer esto con Netcat, en el primer número de la revista... recordáis?. Este tipo de escaneo es proporcionado por nmap, pero se podría calificar como escaneo "ruidoso"... el intento de conexión hace sonar campanas incluso en los sistemas menos protegidos :).

El flag `-sT` (TCP *connect scan*) de nmap nos permite hacer esto. Para verlo, necesitamos utilizar un puerto abierto en nuestra máquina destino. Nosotros vamos a utilizar el puerto de administración https de nuestro router ADSL, pero si disponéis de otra máquina podréis utilizar netcat con los flags `-l` y `-p` seguido del puerto que vayáis a utilizar. Podéis comprobar el resultado en la Figura 6.

Antes de ver lo que nuestro sniffer a capturado vamos a describir rápidamente los flags que hemos utilizado. En primer lugar utilizamos el flag `-PO`, con el que le indicamos a nmap que no lleve a cabo el test de host activo. Esto lo hacemos para que, en la captura de nuestro sniffer no nos aparezcan los paquetes `ECHO` ICMP y el paquete `ACK` al puerto 80... eso ya sabemos como funciona.

El siguiente flag es `-sT` con el que le indicamos que queremos hacer un escaneo de tipo connect. Finalmente, el flag `-p` nos permite indicarle a nmap que solo deseamos escaner una lista concreta de puertos no todos los posibles. En nuestro ejemplo estamos utilizando el puerto HTTPS (443) que nuestro router tiene abierto para permitir la administración remota.

CAPTURA DE UN ESCANEO CONNECT

Y que es lo que nuestro sniffer ha capturado. La figura 7 es la respuesta.

Lo que acabamos de ver es lo que se conoce como "TCP 3-way handshake".. algo así como apretón de manos a tres bandas. Este es el proceso estándar para establecer una conexión TCP.

El proceso se inicia enviando un paquete `SYN` al sistema remoto. Eso es lo que indica la S que nos encontramos al principio de la zona de datos de la primera línea. Cuando el sistema remoto recibe ese tipo de paquetes se inicia el proceso. El sistema remoto responde con un paquete `SYN ACK` (segunda línea). Este paquete sirva para decir "vale, por mi no hay problema para conectarnos... Quieres?". Este paquete tiene ambos flags, `SYN` y `ACK` activados en sus cabeceras.

Cuando el sistema que intenta conectar (occams en nuestro caso), recibe este paquete `SYN ACK`, para que la conexión se complete, debe enviar una respuesta diciendo: "Sí, quiero!". Ese es paquete `ACK` que podemos ver en la tercera línea, también conocido como "paquete de la novia" (esto último es broma :).

FIGURA 6 - ESCANEO CONNECT A UN PUERTO ABIERTO

```
# nmap -P0 -sT razor -p 443
Starting Nmap 4.10 ( http://www.insecure.org/nmap/ ) at 2008-07-20 11:50 BST
Interesting ports on razor (192.168.100.1):
PORT    STATE SERVICE
443/tcp open  https

Nmap finished: 1 IP address (1 host up) scanned in 0.005 seconds
```

FIGURA 7 - CAPTURA TCPDUMP DE UN ESCANEO CONNECT A UN PUERTO ABIERTO

```
11:56:50.970981 IP occams.52355 > razor.https: S 2307061820:2307061820(0) win 5840 <mss 1460,sackOK,timestamp 1645462 0,nop,wscale 2>
11:56:50.971485 IP razor.https > occams.52355: S 2928016144:2928016144(0) ack 2307061821 win 5792 <mss 1460,sackOK,timestamp 684084
                1645462,nop,wscale 0>
11:56:50.971520 IP occams.52355 > razor.https: . ack 1 win 1460 <nop,nop,timestamp 1645462 684084>
11:56:50.971589 IP occams.52355 > razor.https: R 1:1(0) ack 1 win 1460 <nop,nop,timestamp 1645463 684084>
```

En ese momento, la conexión está establecida y el intercambio de información podría empezar. Como nmap está escaneando el puerto y no está interesado en intercambiar ninguna información, inmediatamente envía un paquete RST para terminarla (cuarta línea).

Quizás algunos os estéis preguntado qué es toda esa información adicional que proporciona tcpdump. Bien, todos esos datos son la información contenida en las cabeceras de los protocolos. Los que realmente queráis saber que significa cada una de esas cosas, tenéis dos opciones: O leer los RFCs de TCP e IP, o leer la biblia del TCP-IP, el libro de Stevens "TCP-IP Illustrated".

Nosotros solo os vamos a contar lo que son esos dos número grandes que aparecen en los paquetes SYN. Se trata de números de serie aleatorios que, entre otras cosas, sirven para evitar lo que se conoce como "spoofing".

SPOOFING EXPLICADO

Ya que ha salido el tema, vamos a explicar mú rápido en que consiste el "spoofing" y como esos números aleatorios pueden evitarlo.

Desde el punto de vista más general las técnica de "spoofing" buscan la forma de hacerse pasar por otro. Dependiendo de como se lleve a cabo podemos hablar de DNS spoofing, IP spoofing, o cualquier otra cosa. A nosotros nos interesa el IP spoofing, puesto que es de lo que hemos estado hablando en este artículo.

El IP Spoofing busca permitir la comunicación entre dos máquinas en una red TCP-IP en la que una de estas máquinas utiliza una dirección IP falsa. Generar paquetes con direcciones IP falsas es muy fácil, el problema es que el sistema remoto, va a enviar esos paquetes a la dirección falsa y por lo tanto el sistema del spoofer no va a recibir esas repuestas.

En ese caso tiene dos opciones. O utilizar un snifer para capturar el tráfico, u obviar lo que la víctima diga.... "habla, habla que no te escucho". En el caso de TCP, existe un problema adicional, que acabamos de presentar en la sección anterior; el "3-way handshake".

Como os decíamos, cuando se envía el paquete `SYN`, se incluye un número de serie que el sistema remoto se queda. El sistema remoto envía un paquete `SYN ACK` de vuelta, con su número de serie y el asentimiento del número de serie que había recibido en el paquete `SYN`.

En este punto, el sistema que está intentando establecer la conexión, tiene que enviar un nuevo paquete `ACK` para completar la conexión, como comentamos más arriba. Este paquete `ACK` tiene que referenciar el número de serie recibido de la máquina remota (el del paquete `SYN ACK`). Si el cliente no proporciona el número correcto, la conexión no se puede establecer.

El apretón de manos a tres bandas dificulta el spoofing

Esta es la razón por la que estos números de serie deben ser difíciles de predecir. En el pasado, varias pilas TCP-IP generaban estos números de serie de una forma más determinista y por tanto, era posible "adivinar" cual sería el número de serie que el servidor, o máquina remota nos iba a enviar y así establecer la conexión incluso cuando no se veía el paquete `SYN ACK`.

Nmap nos da algunas pistas sobre esto, y aunque hablaremos sobre el tema más adelante, podéis probar lo siguiente, el comando de la Figura 8.

La opción `-O` habilita la detección del sistema operativo, que comentaremos más adelante, y la opción `-v` es el clásico "verbose". Si os fijáis hacia el final del informe que nmap genera, nos encontramos una línea que dice: "TCP Sequence Prediction". Bueno, ahora ya sabéis que significa :)

FIGURA 8 - SALIDA DE nmap EN MODO VERBOSE Y DETECCIÓN DE S.O.

```
# nmap -v -O razor
Starting Nmap 4.10 ( http://www.insecure.org/nmap/ ) at 2008-07-24 07:28 BST
Initiating ARP Ping Scan against 192.168.100.1 [1 port] at 07:28
The ARP Ping Scan took 0.01s to scan 1 total hosts.
Initiating SYN Stealth Scan against lisa (192.168.100.1) [1679 ports] at 07:28
Discovered open port 443/tcp on 192.168.100.1
The SYN Stealth Scan took 0.57s to scan 1679 total ports.
For OSScan assuming port 443 is open, 1 is closed, and neither are firewalled
Host razor (192.168.100.1) appears to be up ... good.
Interesting ports on razor (192.168.100.1):
Not shown: 1678 closed ports
PORT    STATE SERVICE
443/tcp open  https
MAC Address: 00:18:F8:67:FB:0B (Unknown)
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux 2.4.0 - 2.5.20
Uptime 0.009 days (since Thu Jul 24 07:14:43 2008)
TCP Sequence Prediction: Class=random positive increments
                         Difficulty=2795417 (Good luck!)
IPID Sequence Generation: All zeros
Nmap finished: 1 IP address (1 host up) scanned in 2.736 seconds
               Raw packets sent: 1695 (75.086KB) | Rcvd: 1693 (78.296KB)
```

PUERTOS CERRADOS

En nuestro primer ejemplo nos hemos asegurado de que el puerto estuviera abierto... pero qué sucede cuando el puerto está cerrado?. Recordad que es tan importante saber que el puerto está abierto como que está cerrado. Veamos que sucedería en este caso (Figura 9)

Bien, nmap nos dice que el puerto está cerrado como esperábamos, pero en la captura de tcpdump, vemos como nuestro paquete `SYN` sale sin problemas hacia nuestro objetivo, pero en este caso, el sistema remoto, nos responde inmediatamente con un paquete `RST` indicando que el proceso de conexión debe cancelarse ...porque el puerto está cerrado :) (Regla 2).

Recordad todo lo que hemos estado comentando hasta el momento. En este caso hemos obtenido un paquete `RST` porque la máquina está ahí. Si la máquina no estuviera levantada, probablemente recibiríamos un error ICMP, o en el peor de los casos no recibiríamos nada si un cortafuegos tira nuestro paquete antes de que alcance la máquina destino.

ESCANEOS DE MEDIA CONEXIÓN

Con la información que nos ha dado nuestro sniffer el siguiente tipo de escaneo es obvio. ¿Por qué completar el proceso de conexión con el tercer paquete, si el segundo ya nos ha dado la información que necesitábamos?. Bueno, la respuesta es que el primer tipo de escaneo no necesita permisos especiales, mientras que el que vamos a describir a continuación sí.

El flag de nmap que nos permite hacer este tipo de escaneo es `-sS` (*SYN Scan*). La Figura 10 muestra un ejemplo.

Como podéis observar hemos añadido de nuevo el flag `--send-ip` para que no se generen los paquetes ARP y la captura de nuestro sniffer sea más clara. Si no estáis en una red local podéis prescindir de este flag.

Lo que nuestro sniffer captura en este caso es lo esperado. El proceso es idéntico al del *connect scan* (flag `-sT`), pero en este caso, no se llega a completar la conexión con el tercer paquete `ACK`.

Los escaneos de media conexión son más difíciles de detectar

La ventaja de este escaneo es que los sistemas de log más sencillos no lo detectan. Muchos servidores graban en su log información sobre las distintas conexiones que reciben. En este caso, la conexión no se llega a completar y por tanto nada se grabará en el log.

Sistemas de detección de intrusos más sofisticados son capaces de detectar estos accesos sin grandes complicaciones, después de todo se trata de la opción más sencilla.

Y UDP?

Os preguntaréis algunos. UDP es distinto a TCP puesto que no existe el concepto de conexión. De hecho hacer spoofing sobre una comunicación UDP es bastante sencillo.

El flag de nmap que permite hacer escaneos UDP es `-sU` (obviamente :). En general estos escaneos son más lentos que los TCP, y enseguida veremos por qué.

FIGURA 9 - ESCANEO CONNECT A UN PUERTO CERRADO Y CAPTURA TCPDUMP.

```
# nmap -P0 -sT razor -p 80

Starting Nmap 4.10 ( http://www.insecure.org/nmap/ ) at 2008-07-20 12:29 BST
Interesting ports on razor (192.168.100.1):
PORT   STATE  SERVICE
80/tcp closed http

Nmap finished: 1 IP address (1 host up) scanned in 0.005 seconds
...
12:29:56.261524 IP occams.47587 > razor.www: S 108328587:108328587(0) win 5840 <mss 1460,sackOK,timestamp 2141754 0,nop,wsca
12:29:56.262056 IP razor.www > occams.47587: R 0:0(0) ack 108328588 win 0
```

FIGURA 10 - ESCANEO DE MEDIA CONEXIÓN Y CAPTURA TCPDUMP.

```
# nmap -P0 --send-ip -sS razor -p 80

Starting Nmap 4.10 ( http://www.insecure.org/nmap/ ) at 2008-07-20 12:47 BST
Interesting ports on razor (192.168.100.1):
PORT   STATE  SERVICE
443/tcp open  https
MAC Address: 00:56:28:77:23:AF (Unknown)
Nmap finished: 1 IP address (1 host up) scanned in 0.158 seconds

..
12:49:50.577217 IP occams.45430 > razor.https: S 1685995555:1685995555(0) win 3072 <mss 1460>
12:49:50.577723 IP razor.https > occams.45430: S 1990957085:1990957085(0) ack 1685995556 win 5840 <mss 1460>
12:49:50.577745 IP occams.45430 > razor.https: R 1685995556:1685995556(0) win 0
```

El proceso consiste en enviar un paquete UDP vacío al puerto que se desea comprobar. Si el puerto está cerrado, recibiremos como respuesta un mensaje de error a través del protocolo ICMP (Regla 2), sin embargo si el puerto está abierto, lo más probable es que no recibamos nada. Eso dependerá del servicio UDP que se esté comprobando.

En UDP no existe concepto de conexión, cada paquete viaja independientemente por la red. Tampoco existen paquetes ACK que el sistema remoto envíe de vuelta. Si el servidor remoto envía algún dato tras recibir información desde algún sitio habremos tenido suerte, pero si el servidor no envía nada, no sabremos si hay alguien al otro lado o el cortafuegos ha tirado nuestro paquete.

El gran problema con UDP es que normalmente existe un límite en la cantidad de paquetes ICMP que un sistema puede enviar en un tiempo determinado (esto depende del sistema operativo). Esto implica que el escaneo no se puede realizar a toda velocidad puesto que ciertos paquetes no se recibirían nunca y se considerarían abiertos puertos que no lo están.

En general existen muchos menos servicios que utilizan UDP que los que utilizan TCP y muchas veces no se les presta toda la atención que deberían.

MÁS ESCANEOS SIGILOSOS

Al igual que sucedía con las técnicas de detección de máquinas, nmap proporciona una amplia gama de soluciones de escaneo de puertos "sigilosos" que, en general, son más difíciles de detectar, o, cuanto menos, requieren que el administrador del sistema remoto tome ciertas precauciones. Cosa que no siempre ocurre.

Este tipo de escaneos se fundamentan en que el RFC que define TCP establece que la respuesta a cualquier paquete que no contenga los bits `SYN`, `RST` o `ACK` debe ser `RST` (la Regla 3). De esta forma, jugando con el resto de los flags TCP es posible averiguar si un puerto esta abierto o no de forma sigilosa.

Nmap proporciona tres tipos de escaneos de este tipo y una opción para que el usuario especifique cualquier combinación que desee. Veamos rápidamente cuales son estas tres opciones.

- `-sN` *Null Scan.* Con este flag enviaremos paquetes con todos los bits a cero.
- `-sF` *FIN Scan.* Este flag permite enviar paquetes con el bit TCP `FIN` activo.
- `-sX` *Xmas scan.* Este flag permite enviar paquetes con todos los bits activos. El nombre proviene de un símil con un árbol de navidad en el que todas las luces se encienden a la vez.

Los flags TCP que manipulan estos tipos de escaneo son: `FIN`, `PSH` y `URG`. Los interesados en saber que hace cada uno de ellos deberían leerse el RFC-793 donde se describe en detalle TCP.

Este tipo de escaneos son más difíciles de detectar, o cuanto menos requieren una configuración especial del sistema de seguridad instalado. La contrapartida es que no todos los sistemas siguen el RFC al pie de la letra y, en esos casos, el escaneo puede producir resultados erróneos o no decisorios.

Para que os hagáis una idea aquí tenéis el resultado de tcpdump, para tres escaneos consecutivos al puerto 443 con cada una de las opciones que acabamos de describir.

La Figura 11 nos muestra que en los tres casos, la salida de nmap clasifica el puerto como "open|filtered", no siendo capaz de determinar si el puerto está abierto o se encuentra tras un firewall.

Si os fijáis en la salida de tcpdump, podréis identificar (por las iniciales de los flags) que líneas se corresponden a cada escaneo... es decir, ver se ven, pero alguien tiene que tomar esos datos y decir que se trata de un escaneo de puertos.

Observad que nmap genera dos paquetes en cada escaneo. Esto ocurre porque el primer test no fue definitivo y por tanto se inicia un segundo test (sabríais decir la diferencia). La Figura 12 muestra qué obtendríamos si dirigimos nuestro escaneo a un puerto cerrado.

Ahora podemos ver los paquetes `RST` provenientes del sistema remoto y como, en este caso, nmap nos dice sin lugar a dudas que los puertos están cerrados. Observad también, como el segundo paquete ya no es necesario en este caso.

FIGURA 11 - CAPTURA DE ESCANEOS NULL, FIN, XMAS A PUERTO CERRADO.

```
13:16:59.166446 IP occams.53929 > razor.https: . win 4096
13:17:00.163292 IP occams.53930 > razor.https: . win 3072
13:17:12.916484 IP occams.53021 > razor.https: FP 2677994679:2677994679(0) win 2048 urg 0
13:17:13.917228 IP occams.53022 > razor.https: FP 2678060214:2678060214(0) win 4096 urg 0
13:17:22.289316 IP occams.63528 > razor.https: F 1455231218:1455231218(0) win 1024
13:17:23.290118 IP occams.63529 > razor.https: F 1455165683:1455165683(0) win 2048
```

FIGURA 12 - CAPTURA DE ESCANEOS NULL, FIN, XMAS A PUERTO ABIERTO.

```
13:31:58.832801 IP occams.52797 > razor.www: . win 1024
13:31:58.833375 IP razor.www > occams.52797: R 0:0(0) ack 3074120381 win 0
13:32:03.356435 IP occams.40594 > razor.www: F 1161376370:1161376370(0) win 3072
13:32:03.356950 IP razor.www > occams.40594: R 0:0(0) ack 1161376371 win 0
13:32:07.776070 IP occams.56879 > razor.www: FP 3957574167:3957574167(0) win 2048 urg 0
13:32:07.776575 IP razor.www > occams.56879: R 0:0(0) ack 3957574168 win 0
```

EXCEPCIONES A LAS REGLAS

Algunas secciones antes hemos introducido tres reglas generales con las que pretendíamos resumir el proceso de escaneo de puertos, sin embargo, como sucede con toda regla, siempre hay excepciones.

La principal excepción es el denominado escaneo TCP *Maimon scan*, que toma su nombre de Uriel Maimon, la persona que lo descubrió. La técnica es idéntica a los escaneos FIN, Null y Xmas que vimos en la sección anterior, pero en este caso los flags utilizados son `FIN` y `ACK`. Según la especificación de TCP, frente a un paquete de este tipo, el sistema remoto debería responder con un paquete RST, sin embargo, ciertos derivados BSD, tiran el paquete si el puerto al que va dirigido está abierto.

El escaneo TCP Maimon explota una característica especial de los sistemas BSD

El otro tipo de escaneo que queremos comentar no es realmente una excepción sino más bien una alternativa, que se basa en el análisis del campo "TCP Window" de los paquetes TCP.

Esta técnica es similar al TCP ACK scan que acabamos de describir, pero utiliza otras propiedades de las pilas TCP que permiten determinar si el puerto está abierto o no. Para ello, esta técnica analiza el campo "TCP Window" de los paquetes TCP. En ciertos sistemas, este campo es positivo si el puerto está abierto y cero si el puerto está cerrado.

Este tipo de escaneo es dependiente de la plataforma, es decir, no todos los sistemas responden de esta forma, y por tanto, el resultado obtenido no siempre es fiable. En general, si el resultado del escaneo es una mezcla de puertos abiertos y cerrados, probablemente el escaneo haya tenido éxito. Si todos los puertos aparecen como cerrados, puede ser que realmente lo estén o que este método no funcione en este caso.

Para terminar con esta técnica, comentar que el comportamiento del sistema remoto puede ser exactamente al contrario, es decir, si el escaneo informa de que todos los puertos están abiertos, excepto un puñado de ellos, probablemente los que componen ese puñado sean los que realmente están abiertos.

Nmap proporciona además un flag que nos permite enviar paquetes con la combinación de flags que deseemos. Este flag es `--scanflags` y los campos TCP se indican con las cadenas de texto: `URG ACK PSH RST SYN FIN`. Así la siguiente línea de comandos, activaría todos los flags:

```
# nmap --scanflags URGACKPSHRSTSYNFIN razor
```

Estos flags se pueden combinar con los tipos de escaneo que hemos comentado hasta el momento. El tipo de escaneo seleccionado (con el flags -s seguido del tipo elegido), va a determinar como se interpretarán los paquetes recibidos del sistema remoto. Recordad que según el tipo de escaneo, recibir un paquete `RST` o un tamaño de ventana positivo puede significar que el puerto esté abierto o cerrado, dependiendo de lo que hayamos enviado.

CORTAFUEGOS CON Y SIN CONEXIÓN

Como todos sabéis, un cortafuegos o firewall, permite controlar el tráfico que entra y sale de una red, para evitar ataques, normalmente desde el exterior. La configuración más habitual es que el cortafuegos, solo permita conexiones desde la red interna hacia el exterior, de forma que los usuarios de la red interna puedan navegar por internet o leer su correo. Si se desea proporcionar algún servicio, en general se suele utilizar una configuración más complicada, con dos cortafuegos y una DMZ.

La forma más sencilla de evitar conexiones desde el exterior, es tirar los paquetes `SYN`, sin embargo, los paquetes `ACK` no se pueden tirar ya que son necesarios para que las máquinas tras el cortafuegos puedan utilizar la red.

Los cortafuegos con estado o seguimiento de conexiones, son capaces de filtrar paquetes `ACK` "inapropiados". Para ello, mantienen una lista de todas las conexiones activas entre el exterior y la red interna. Si recibe un paquete `ACK` que no se pueda asociar con ninguna de las conexiones activas, ese paquete es descartado.

El escaneo ACK de nmap, está especialmente diseñado para detectar estas configuraciones.

ESCANEOS SOFISTICADOS

Como decía aquel conocido super-roedor... "No se vayan todavía, que aún hay más". Efectivamente, nmap es una mala bestia :).

No hace mucho tiempo, nmap incluyó un nuevo modo de escaneo denominado *Idlescan*. La peculiaridad de este escaneo es que es indetectable. El problema es que no se puede utilizar en cualquier situación, como sucede con las técnicas que hemos visto hasta el momento..

El escaneo IDLE es indetectable ya que la maquina del atacante no envía datos a la víctima

La técnica es indetectable en el sentido de que la máquina del atacante no envía ningún paquete al sistema remoto, y por tanto no existe una forma directa de relacionar ambas máquinas. Evidentemente, para conseguir esto, hay que usar una tercera máquina, a la que la documentación de nmap se refiere como "maquina zombie". Está máquina es la que recibirá la información por nosotros.

En el website de nmap, podéis encontrar un detallado artículo sobre el funcionamiento de esta técnica. Nosotros, como es habitual, solo os vamos a dar una breve descripción para que podáis seguir investigando por vuestra cuenta, que es lo realmente divertido.

El proceso se basa en la monitorización de los números de identificación de fragmento IP (IP ID). Este número se envía en las cabeceras IP y normalmente se incrementa con cada paquete enviado, de forma que si podemos leer este número para una determinada máquina, sabremos cuantos paquetes IP ha enviado esa máquina hasta el momento.

El IDLE SCAN utiliza los números de identificación de fragmento IP de las máquinas zombie

Con esta información, muchos de vosotros ya os estaréis imaginando como funciona este tipo de escaneo. Efectivamente, los pasos son bastante evidentes (lo que no es evidente es que esto se le ocurra a cualquiera :).

- Obtener el IP ID de la máquina zombie.
- Enviar un paquete `SYN` modificado con la dirección IP de la máquina zombie, a la máquina que se quiere escanear.
- Comprobar de nuevo el IP ID de la máquina zombie.

Un incremento de 1 en el ID significa que la máquina zombie no ha recibido ningún paquete a excepción de la prueba realizada desde la máquina atacante. Esto significa que el puerto remoto está cerrado, puesto que la máquina zombie habrá recibido un paquete `RST` o nada (en caso de existir un firewall).

Si el ID se ha incrementado en 2, el puerto estará abierto. La máquina víctima habrá respondido con un `SYN ACK` a la prueba `SYN` que envió el atacante. Ante este paquete, la máquina zombie enviará un paquete `RST` a la máquina víctima, puesto que, para ella, ninguna conexión está en marcha.

Si el ID se ha incrementado en más de 2, significa que el zombie no es una buena elección. Sí, eso todavía no lo hemos dicho, pero la máquina zombie debe ser una que no tenga apenas actividad de red, de forma que durante el escaneo, no envíe ningún paquete y la monitorización del IP ID sea efectiva.

Como os decíamos, en el sitio web de nmap podéis encontrar un extenso y detallado artículo sobre como funciona este tipo de escaneo (está traducido al español y portugués).

Para terminar, comentaros que existe una forma de escaneo adicional denominada *FTP bounce scan*, que explota las denominadas conexiones proxy ftp. La mayoría de los servidores FTP actuales tienen desactivada esta funcionalidad, porque ha dejado de ser práctica... pero siempre hay algún despistado por ahí. Los que queráis conocer más detalles sobre este tipo de escaneo, podéis consultar la página del manual de nmap o su sitio web.

ESCANEO DE PROTOCOLOS

Nmap proporciona un modo de escaneo de protocolos que se activa con el flag -sO. En este modo, en lugar de probar puertos de un determinado protocolo (UDP o TCP), nmap trata de comprobar que protocolos IP soporta una determinada máquina. El protocolo IP es un campo de la cabecera IP de los paquetes, así que nmap, simplemente envía paquetes con distintos valores en este campo. Como respuesta a estas pruebas, nmap espera mensajes ICMP, concretamente el mensaje "Protocol Unreachable" que significará que ese protocolo no está soportado. La recepción de otro tipo de mensaje ICMP hará que nmap marque el puerto como filtrado. En cualquier otro caso, nmap determinará que el protocolo está soportado. Como podéis observar el funcionamiento es muy similar al del escaneo UDP.

PONIÉNDOLO DIFÍCIL

Además de todas estas distintas formas de escaneo, para otros tantos escenarios, nmap proporciona una serie de flags adicionales con los que poner en apuros a los IDSs más sofisticados. Veamos rápidamente cuales son:

- `-f`: Fragmentado de paquetes. Utilizando esta opción nmap enviará sus paquetes de prueba como pequeños fragmentos, de forma que la cabecera TCP se envíe en varios paquetes IP, en lugar de en uno solo. Esto obliga a los firewall e IDSs a realizar el proceso de reensamblado de paquetes que, requiere CPU y memoria, y por eso muchas veces se desactiva. Alternativamente a esta opción es posible especificar la MTU (*Maximum Transfer Unit*) con la que especificar el tamaño máximo de cada paquete que se envía (ejecutad ifconfig para saber cual es la MTU para vuestro interfaz de red :).
- `-D`: Señuelos. Esta es una de las opciones más interesante para ofuscar un escaneo. Esta opción permite especificar una lista de máquinas separadas por comas. Por cada una de las máquinas especificadas, nmap producirá un paquete idéntico al de la prueba que está realizando, pero cambiando la dirección IP de origen por la de la máquina especificada con `-D`.

  De esta forma, desde el punto de vista del sistema remoto, parecerá que está siendo escaneado por todas esas máquinas simultáneamente, resultando bastante complicado determinar, cual de ellas es la que realmente está llevando a cabo el escaneo de puertos.

- `-S`: Spoofing de dirección origen. A lo largo de este texto hemos hablado en varias ocasiones de cambiar la dirección origen de los paquetes IP. Esta opción permite realizar precisamente eso. Si bien, su verdadera utilidad es la de indicarle a nmap cual es la dirección IP que debe utilizar cuando no es capaz de determinarla por si mismo (porque tengamos varios interfaces de red, por ejemplo), puede servir a otros usos maliciosos.

Relacionado con este último, el flag `--source-port` o `-g` que permite especificar el puerto de origen de los paquetes.

SACANDO EL MÁXIMO DE INFORMACIÓN

Para terminar con esta mala bestia, vamos a hablaros de una de las funcionalidades más potentes de nmap (la otra es el lenguaje de script que proporciona, pero este artículo ya es demasiado largo).

Nmap es capaz de realizar una batería de pruebas contra los puertos de una determinada máquina. Estas pruebas son un poco más complejas que los métodos de escaneo que hemos visto, y en general se basan en la ejecución de una serie de scripts y de la consulta de una base de datos bastante completa.

Nmap utiliza un sistema de "pruebas" para obtener información adicional sobre el sistema remoto

Con este proceso, nmap es capaz de darnos un montón de información sobre la aplicación asociada a un determinado puerto y sobre el sistema operativo de la máquina remota. Esta información se obtiene con los flags `-vS`, para obtener información sobre la aplicación asociada a un determinado puerto, y el flag `-O` para obtener información sobre el sistema operativo.

En la sección sobre spoofing ya vimos que tipo de información nos proporciona el flag `-O`. La Figura 13, muestra lo que obtenemos al utilizar el flag `-vS`... da miedo eh?

Bueno, la verdad es que no es para tanto. Si utilizamos nuestro querido netcat para una prueba rápida obtenemos:

```
# nc localhost 80
GET / HTTP/1.1
Host: localhost:8080

HTTP/1.1 200 OK
Date: Tue, 06 Jan 2009 20:54:31 GMT
Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.4 with Suhosin-Patch
Last-Modified: Tue, 30 Dec 2008 16:06:24 GMT
ETag: "3c1ca-2d-45f45c5dfc000"
Accept-Ranges: bytes
Content-Length: 45
Content-Type: text/html

<html><body><h1>It works!</h1></body></html>
```

El texto en color azul es lo que nosotros escribimos, y el texto en color rojo, es lo que el servidor nos cuenta "de porque sí".

Si utilizáis tcpdump, con un par de opciones que podéis encontrar en su página del manual, veréis que, entre otras cosas, nmap hace una petición de este tipo. Si tenéis un servidor ssh en vuestra máquina probad a repetir el experimento (el puerto ssh es el 22 ;).

MUCHAS MÁS OPCIONES

Nmap es una caja de sorpresas y os hemos dejado muchas cosas para que investiguéis. Esperamos que con lo que os hemos contado en este artículo tengáis las herramientas suficientes para utilizar a fondo nmap.

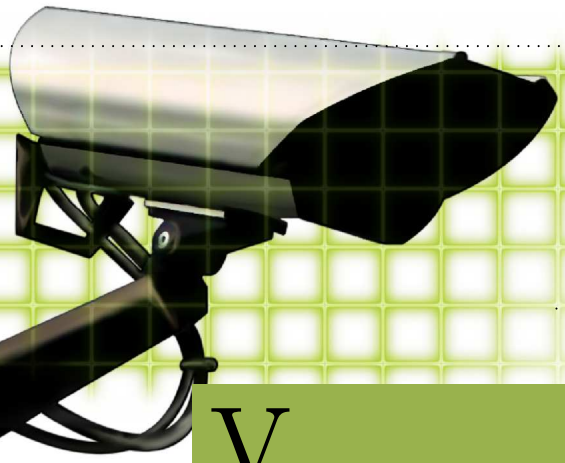
Solo tenéis que ejecutar "man nmap" en vuestra consola y empezar a leer. Hasta el próximo número. ■

FIGURA 13 - SALIDA DEL FLAG -vS CONTRA UN SERVIDOR WEB.

```
# nmap  -sV localhost -p 80

Starting Nmap 4.53 ( http://insecure.org ) at 2009-01-06 21:51 CET
Interesting ports on localhost (127.0.0.1):
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.2.8 ((Ubuntu) PHP/5.2.4-2ubuntu5.4 with Suhosin-Patch)

Service detection performed. Please report any incorrect results at http://insecure.org/nmap/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.160 seconds
```

RECURSOS

Sitio Web nmap
http://www.insecure.org
RFC 791 - Internet Protocol
http://www.faqs.org/rfcs/rfc791.html
RFC 792 - Internet Control MessageProtocol
http://www.faqs.org/rfcs/rfc792.html
RFC 793 - Transmission Control Protocol
http://www.faqs.org/rfcs/rfc793.html



Sistema de Videovigilancia Casero

Controla tu casa en dos patadas... y mucho más

por Er Boyer

Vale, para montar un sistema de videovigilancia casero mú rápido, podemos usar motion... eso sería lo más rápido, pero no sería tan divertido. En este número os vamos a contar como programar vuestros dispositivos de adquisición de vídeo, como codificar imágenes, y como mostrarlas en plan Aero en una ventana. Mola no?

Al final de este artículo habremos desarrollado un sistema de vídeo vigilancia muy rudimentario, es decir, no va a ser especialmente útil, pero os proporcionará el punto de partida para desarrollos más complejos, y sobre todo, para poder sacarle partido a vuestros dispositivos de adquisición de imagen, más allá de ver guarrerías decodificadas o chatear con el "mesenyer".

Nuestro sistema, nos permitirá capturar imágenes de varios dispositivos, almacenarlas comprimidas en el disco y visualizarlas de una forma güay... y como siempre, todo ello, mú rápido!

ARQUITECTURA

Un sistema de vídeo vigilancia básico debe ser capaz de realizar las siguientes tareas. A saber:

- Capturar imágenes de al menos una fuente de vídeo.
- Almacenar esas imágenes de forma eficiente (vamos, que no reventemos el disco duro).
- Visualizar las imágenes almacenadas y, por supuesto, el vídeo en tiempo real.

Con esto en mente, nuestro sistema estará compuesto de tres módulos. Un módulo con el que capturar imágenes, otro con el que codificar y decodificarlas y finalmente otro que nos permita visualizarlas. Obvio no?

Para conseguir esto, vamos a utilizar distintos sistemas de soporte.

Para la captura de vídeo utilizaremos el API V4L (Video 4 Linux) que proporciona nuestro querido kernel Linux. Este API, como veremos muy pronto, es un poco engorroso, por esa razón vamos a escribir un pequeño interfaz para hacer nuestra vida más fácil. En este artículo utilizaremos la versión 1 del API, que, aunque obsoleta, es la que más dispositivos soportan.

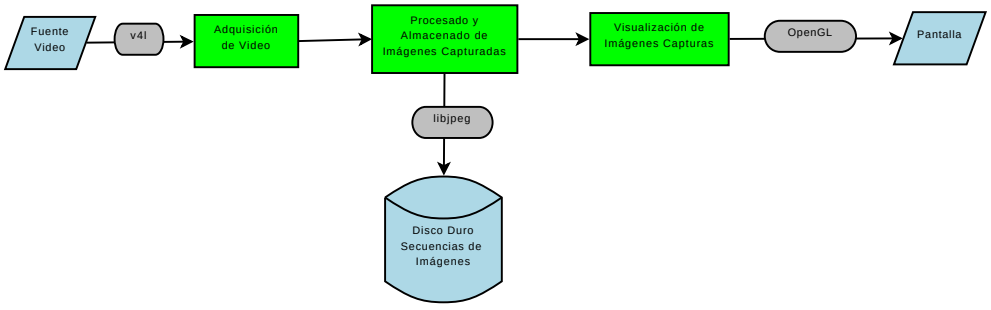
Para la codificación y decodificación de las imágenes utilizaremos la librería libjpeg. También os contaremos cómo utilizar un formato mucho más sencillo y que podéis incluir en vuestras aplicaciones "mú rápido".

Finalmente, para la visualización de las imágenes vamos a utilizar OpenGL. Esto si que es fácil, y además nos va a permitir visualizar nuestros vídeos en un entorno 3D bastante impactante. Os daremos las bases para poder utilizarlo... que quede bonito dependerá de vosotros :).

EL HARDWARE

Lo que vamos a necesitar seguro es algún tipo de hardware de captura de imágenes. Aquí tenemos varias opciones y las buenas noticias son que, prácticamente, todas ellas funcionan con el API V4L (Video for Linux).

Nosotros vamos a utilizar dos webcams Logitech Messenger que están bien soportadas en Linux y nos proporcionan, en un solo bloque, la cámara y el hardware de adquisición. Además solo cuestan unos 25 euros (julio 2008).



Arquitectura de nuestro sistema. Hardware, Software y APIs

Si disponéis de una tarjeta capturadora de TV BT848, necesitaréis una fuente de vídeo adicional, una cámara externa o, en su defecto, un DVD o magnetoscopio. Si utilizáis la salida de vídeo compuesto, el código que vamos a comentar en el artículo debería funcionar casi sin modificaciones. Si, lo que queréis es utilizar la entrada de RF (la de la antena), serán necesarias algunas modificaciones. Os daremos algunas indicaciones de como hacerlas pero, por cuestiones de espacio, no las vamos a incluir aquí.

Un par de webcams nos servirán para nuestro prototipo.

En nuestro caso concreto, el de las webcams Logitech Messenger, tenemos que hacer un par de comentarios. El primero es que parece que las cámaras (o más bien el driver) no funciona correctamente si se conectan a través de un HUB USB externo. Para poder utilizarlas deben estar conectadas directamente a los puertos USB del ordenador.

El segundo problema que nos encontramos con estas cámaras, fue que el driver no soporta más de un dispositivos del mismo tipo. Mardita sea!.

ARREGLANDO EL DRIVER

Aunque para lo que os vamos a contar en este artículo, podríamos utilizar una única cámara, a nosotros nos interesaba poder utilizar dos o más para poder hacer cosas más güays.... os suenan esas gafas rojas y azules?.

Así que nos bajamos el driver y le echamos un ojo, con la esperanza de poder modificarlo fácilmente para que nuestras dos cámaras funcionaran. Bendito software libre!.

Con la información de depuración del driver y mucha suerte, encontramos que el "problema" estaba en la función `gspca_init_transfert`.

Por cada dispositivo, esta función "reparte" lo que en los logs aparece como un "AlternateSet". Cuando la primera cámara es inicializada, la función le asigna todos los "recursos" (por llamarlos de alguna forma) y al intentar inicializar la segunda, se produce un error, ya que, aparentemente no hay "recursos" que asignarle (los tiene todos la primera cámara).

Los recursos no son otra cosa que el ancho de banda USB que se asigna a ese dispositivo. En la FAQ de motion, podréis encontrar algunos detalles más sobre como utilizar varias cámaras USB. Básicamente necesitaríais una segunda tarjeta USB. Como nosotros no la tenemos, hemos optado por este parche, como solución de compromiso.

Para ello, añadimos un nuevo parámetro al driver que llamamos `ndevices`. Las modificaciones en el fichero `gspca_core.c` fueron las siguientes:

```
static int ndevices = 1;

module_param(ndevices, int, 0644);
MODULE_PARM_DESC(ndevices, "Number of devices to"
                           " use simultaneously.");
```

Esta nueva variable la utilizamos en la función `gspca_init_transfert` para modificar el valor de la variable `nbalt`, de la siguiente forma:

```
intf = usb_ifnum_to_if(dev, spca50x->iface);
nbalt = intf->num_altsetting - 1;

/* Nuestra nueva línea */
nbalt = nbalt / ndevices;
```

Por supuesto esto es un hack "mú rápido" y probablemente haya mejores formas de hacerlo, pero para las pruebas que nosotros vamos a hacer es más que suficiente.

Ahora podemos cargar nuestra nueva versión del driver, de la siguiente forma:

```
# rmmod gspca
# insmod ./gspca.ko ndevices=2
```

Et voilà!.... tenemos nuestras dos webcams disponibles.

Para comprobar que todo funciona correctamente, podéis utilizar la aplicación spcaview descargable desde la página del driver gspca. Este programa es especialmente interesante ya que se trata de un ejemplo completo de como manejar un dispositivo soportado por este driver.

Como esta es la sección "mú rápido", nuestro interfaz V4L va a ser muy simplón, y estudiar esta aplicación os permitirá extenderlo si queréis hacer que vuestros programas funcionen en el caso general.

Para la captura de imágenes utilizaremos el API V4L

Si vuestro hardware utiliza otro driver, probad con alguna aplicación estándar como *camorama* o *cheese*.

CAPTURANDO VIDEO

Después de todos estos preparativos vamos al ajo :). Lo primero que vamos a hacer es nuestro módulo de captura de vídeo. En esta versión supersimplificada, nuestro módulo tendrá tres funciones. Aquí tenéis los prototipos.

```
int open_v4l (DISP d, char *d, int *w, int *h);
void start_v4l (DISP d);
char *capture_v4l_full (DISP d);
```

La primera de las funciones abre e inicializa el dispositivo de captura. La segunda le dice al dispositivo que comience a capturar video. Por último, la tercera, nos permite obtener la última imagen capturada por el dispositivo.

Para poder utilizar el API V4L, necesitamos los siguientes includes en nuestro código:

```
#include <sys/ioctl.h>
#include <fcntl.h>

#include <sys/mman.h>
#include <linux/videodev.h>
```

El primero es necesario para utilizar la llamada al sistema `ioctl`, con la que poder enviar mensajes al driver. El segundo solo es necesario para la llamada al sistema `open`, con la que acceder al dispositivo de captura. En seguida veremos como utilizarla.

El tercero lo necesitamos para el proceso de captura. Nosotros vamos a utilizar un mapeado de memoria en lugar del método `read`.

Finalmente, el último de los includes es el que define todas las estructuras de datos específicas de los dispositivos de video. La mayoría de las estructuras de datos que utiliza la función `open_v4l` están definidas aquí.

Para poder utilizar más de un dispositivo de captura de video, hemos definido una estructura muy sencilla en la que almacenar los datos específicos de cada uno de ellos. Esta es:

```
typedef struct disp_t {
  int                fd;
  struct video_mbuf  mbuf_info;
  struct video_mmap  mmap_info;
  /* Mapa de memorua de datos de video */
  char               *mm;
  char               *buffer;
  /* Información sobre la imagen */
  int                w, h, canal, bpp;
  double             brillo, contraste, color;
} *DISP;
```

El uso de cada uno de estos campos quedará claro en cuanto describamos las funciones de nuestro interfaz... esto es, ya mismo :)

ABRIENDO EL DISPOSITIVO

La función `open_v4l` es la que se encarga de la inicialización del dispositivo de captura de vídeo, y es la más larga de todas, por esa razón la iremos comentando por partes, para que sea más fácil seguirla.

Lo primero que nos encontramos es esto:

```
  struct video_capability dev_info;
  struct video_channel    channel_info;
  struct video_picture    picture_info;

  /* Abrimos el dispositivo de video */
  if ((d->fd = open (device, O_RDWR)) < 0)
    {
      fprintf (stderr, "No puedo abrir "
                       "dispositivo %s\n",
                       device);
      return (-1);
    }
```

La función comienza declarando tres variables locales que utilizaremos durante la inicialización del dispositivo, para, a continuación, abrir el dispositivo.

Dos comentarios aquí. El primero es que hemos eliminado todas las comprobaciones de error con la excepción de esta primera. Si el dispositivo se abre, muy probablemente todo vaya como la seda, pero si esto falla, nada funcionará.

El segundo es que los dispositivos de vídeo, tienen nombres como `/dev/video1` o `/dev/video2`. Tendremos tantos como dispositivos de captura hayamos instalado en el sistema. Si vuestro hardware de captura está correctamente instalado los dispositivos deberían estar ahí.

CONFIGURANDO CANALES

El siguiente paso consiste en configurar el canal del dispositivo que vamos a utilizar. Si estáis utilizando una webcam como nosotros, probablemente solo tengáis un canal. Pero si estáis utilizando una tarjeta de captura de televisión, normalmente tendréis al menos 2 canales. Uno para la entrada de video compuesto (o banda base) y otro para el sintonizador de televisión.

Video 4 Linux nos permite configurar los parámetros de captura de nuestra webcam

Si os veis obligados a utilizar la entrada del sintonizador de televisión, tendréis que configurar el canal correcto, y programar el sintonizador para seleccionar el canal (de la TV) que queráis capturar.

No vamos a explicar como hacer todo esto, pero si alguien está utilizando una tarjeta de captura TV, le recomendamos que utilice la librería libbgrab.

Volviendo al código...

```
/* Conseguimos las caracteristicas */
ioctl (d->fd, VIDIOCGCAP, &dev_info);

/* Nos quedamos con el ancho y alto máximos */
d->w = dev_info.maxwidth;
d->h = dev_info.maxheight;

*w = d->w;
*h = d->h;

/*Usamos canal 0. Entrada de video compuesto*/
channel_info.channel = d->canal;

/*Obtenemos información del canal seleccionado*/
ioctl (d->fd, VIDIOCGCHAN, &channel_info);

/*Activamos modo PAL*/
channel_info.norm = 0;

/*Actualizamos la información del canal*/
ioctl (d->fd, VIDIOCSCHAN, &channel_info);
```

Como adelantamos más arriba, la forma de "hablar" con el driver es a través de la llamada al sistema `ioctl`. Para la configuración del canal, utilizamos tres mensajes. `VIDIOCGCAP`, algo así como "VIDeo Input Output Chanel Get Capabilities"... vamos digo yo :), con lo que obtenemos las características del dispositivo, la información sobre sus canales. En nuestro ejemplo, estamos utilizando este mensaje para obtener el ancho y alto máximo de las imágenes que pueden ser capturadas utilizando este canal y almacenándolas en nuestra estructura para su uso posterior.

Estos valores los retornamos a la aplicación principal, ya que los necesitaremos para la visualización de las imágenes.

El siguiente mensaje es `VIDIOCGCHAN`, con el que obtenemos información sobre el canal de interés. El canal de interés se indica utilizando el campo `channel` de la estructura de datos `video_channel` (variable `channel_info`). La información sobre el canal se retorna sobre esa misma estructura de datos.

Los dispositivos de captura de imágenes pueden proporcionar varios canales

Finalmente, utilizamos el mensaje `VIDIOCSCHAN` (la s es de SET), para configurar el canal. Lo que estamos haciendo es coger la configuración del canal (`VIDIOCGCHAN`), modificando solamente la norma de video a PAL y reconfigurando el canal de nuevo (`VIDICSCHAN`).

Puede que dependiendo de vuestro hardware tengáis que modificar algún campo más para configurar apropiadamente el canal, o que no necesitéis modificar la norma de vídeo para vuestro dispositivo. Ahí tendréis que investigar un poco por vuestra cuenta.

Hemos intentado reducir al mínimo el código de inicialización saltándonos varias comprobaciones que serían necesarias en una aplicación "profesional". Si algo falla, coged el código de alguno de los programas incluido en vuestra distribución que funcione correctamente, y empezad a añadir las partes que faltan... eso es lo que nosotros hemos hecho :).

PARÁMETROS DE IMAGEN

Ahora que hemos seleccionado y configurado nuestro canal, tenemos que proporcionar al driver una serie de parámetros, para decirle como queremos las imágenes.

Aquí podéis ver como hacerlo.

```
/* Ahora los parámetros de imagen */
picture_info.brightness = (int)(65535*d->brillo);
picture_info.hue        = 0;
picture_info.colour     = (int)(65535*d->color);
picture_info.contrast   = (int)(65535*d->contraste);
picture_info.whiteness  = 0;
picture_info.depth      = 24;
picture_info.palette    = VIDEO_PALETTE_RGB24;

ioctl (d->fd, VIDIOCSPICT, &picture_info);
```

La estructura `video_picture` (variable `picture_info`), nos permite configurar el brillo, contraste y color de la imagen, entre otros parámetros. Dependiendo si la captura es en color o escala de grises ciertos parámetros no tienen efecto.

Todos estos valores se codifican como un entero entre 0 y 65535, y nosotros los proporcionamos como valores reales entre 0 y 1 representando un porcentaje.

Pero los datos importantes son los dos últimos (`depth` y `palette`). Estos campos nos permiten configurar la profundidad de color y el formato de la imagen que vamos a obtener. Nosotros estamos utilizando 24 bits por pixel (True Color) y un formato RGB de 24 bits, ya que esto va a hacer nuestro código de visualización mucho más sencillo.

Estos parámetros dependen del hardware de captura e idealmente, el programa, debe utilizar las `ioctl` apropiadas para preguntar al driver que formatos soporta y configurarlo apropiadamente. La configuración que estamos utilizando es bastante estándar y debería funcionar en la mayoría de los casos, pero quizás tengáis que modificar estos valores para que funcionen en vuestro hardware.

Obviamente el mensaje `VIDIOCSPICT` es el que nos permite pasar estos datos al driver (ya sabéis Set PICTure).

CONFIGURANDO LA TRANSFERENCIA DE DATOS

El último paso que nos falta para completar el proceso de inicialización es la configuración de transferencia de datos. Como adelantábamos un poco más arriba, hay dos formas de transferir las imágenes a nuestro programa. Una es utilizando la llamada al sistema `read` y otra es utilizando mapeado de memoria con la llamada al sistema `mmap`.

Nosotros utilizamos el mapeado de memoria, porque mola más y además podemos utilizar doble buffer para la captura. El código es el siguiente:

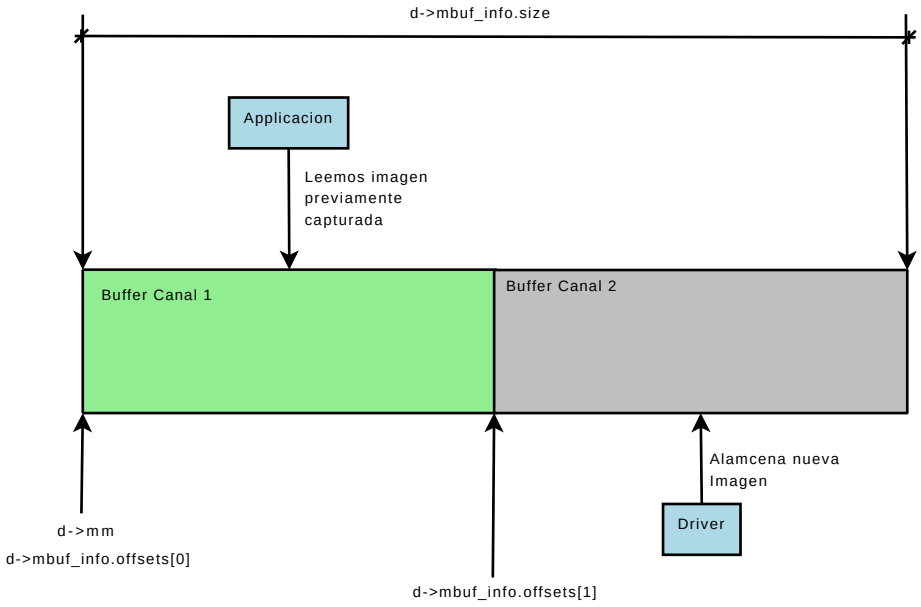
```
/* Obtenemos mapa de meoria */
ioctl (d->fd, VIDIOCGMBUF, &d->mbuf_info);

/* Vamos a trabajar con doble buffer */
if (d->mbuf_info.frames < 2)
  exit (1);
```

Lo primero que hacemos es preguntarle al driver los detalles para el mapeado de memoria, utilizando , una vez más, la llamada al sistema `ioctl`. En la estructura `video_mbuf` (uno de los campos de nuestra estructura `DISP`), obtendremos el resultado.

Nuestra aplicación captura las imágenes utilizando mapeado de memoria

¿Y cuales son los detalles para el mapeado de memoria? Os preguntaréis. Básicamente son los siguientes: El número de cuadros que el dispositivo es capaz de capturar, el tamaño de la zona de memoria en la que el driver va a poder los datos de la imagen y los desplazamientos, dentro de ese bloque de memoria, donde se almacenarán los distintos cuadros.

Todo esto tomará sentido muy pronto, en cuanto os mostremos una sencilla figura :).



En nuestro ejemplo, utilizamos doble buffer. Esta técnica consiste en utilizar dos bloques de memoria para adquirir los datos, de tal forma que cuando el driver está capturando un cuadro, lo está almacenando en uno de esos buffers y así, nosotros podemos acceder al otro (el cuadro anterior) sin que nadie nos moleste.

Utilizando doble buffer podemos capturar una imagen, mientras procesamos otra

Cuando el cuadro actual haya sido capturado, los buffers se intercambian y nuestra aplicación puede acceder al nuevo cuadro a la vez que el driver captura los datos del siguiente cuadro y los almacena en el otro buffer.

Por esa razón terminamos nuestra aplicación si el driver no es capaz de proporcionarnos al menos dos buffers. Si por alguna razón vuestro hardware/driver no soporta más que un buffer, podéis eliminar esa comprobación, pero tendréis que hacer unos pequeños cambios al resto de las funciones de este módulo.

Una vez obtenida esta información, utilizamos la llamada al sistema `mmap` para mapear la memoria, es decir, para permitir a nuestra aplicación acceder a los buffers que el driver está utilizando y, por tanto, acceder a los datos de las imágenes capturadas.

```
/* Mapeamos memoria para acceder a las imagenes*/
d->mm = (char *)mmap (0, d->mbuf_info.size,
                      PROT_READ | PROT_WRITE,
                      MAP_SHARED,
                      d->fd, 0);
```

Finalmente, reservamos un buffer adicional en el que almacenaremos una copia de la imagen capturada, e inicializamos el campo `mmap_info` con los datos que necesitaremos durante la captura real de las imágenes.

```
d->buffer = malloc (d->mbuf_info.size);

/* Estructura para el mapeo de memoria */
d->mmap_info.frame  = 0;
d->mmap_info.width  = d->w;
d->mmap_info.height = d->h;
d->mmap_info.format = picture_info.palette;
```

En este punto, nuestro hardware de captura está listo para empezar a "escupir" imágenes... pero, ¿cómo las obtenemos?... enseguida lo veremos.

INICIANDO LA CAPTURA

El proceso de captura lo hemos dividido en dos funciones, una que inicia el proceso y otra que obtiene los datos de la imagen.

La primera pinta tal que así:

```
void start_v4l (DISP d)
{
  d->mmap_info.frame = 0;
  ioctl (d->fd, VIDIOCMCAPTURE, &d->mmap_info);
  d->mmap_info.frame = 1;
  ioctl (d->fd, VIDIOCMCAPTURE, &d->mmap_info);
  d->mmap_info.frame = 0;
}
```

Esta función utiliza el mensaje `VIDIOCMCAPTURE` y espera como parámetro una estructura del tipo `video_mmap`. El campo `frame`, le dice al driver qué cuadro capturar, o dicho de otra forma, que parte del buffer utilizar.

Con lo que acabamos de comentar, debería estar claro lo que hace la función. Inicia la captura de los dos frames y nos devuelve el control mientras el hardware hace su trabajo.

A partir de ese momento, nuestra aplicación principal entrará en un bucle infinito, en que llamará continuamente a la función `capture_v4l` que os mostramos a continuación:

```
char *capture_v4l (DISP d)
{
  ioctl (d->fd, VIDIOCSYNC, &d->mmap_info.frame);

  memcpy (d->buffer, d->mm +
          d->mbuf_info.offsets[d->mmap_info.frame],
          d->mbuf_info.size / 2);

  ioctl (d->fd, VIDIOCMCAPTURE, &d->mmap_info);
  d->mmap_info.frame = 1 - d->mmap_info.frame;

  return d->buffer;
}
```

Lo primero que hace la función es enviar el mensaje `VIDIOCSYNC`. Este mensaje hace que nuestro programa espere hasta que el cuadro que se pasa como parámetro esté listo. Que significa que esté listo?. Pues que la parte del buffer asociada a ese cuadro, contenga una imagen completa.

Nuestro sencillo interfaz con V4L, nos permite capturar imágenes de forma mú rápida

Una vez que tenemos una imagen disponible (cuando `ioctl` retorna), copiamos inmediatamente la imagen a nuestro buffer temporal. En principio, esto no sería necesario, pero en las distintas pruebas que hemos hecho, cuando utilizamos más de un dispositivo los buffers se corrompen, si bien, esto puede ser debido a que el programa es muy tonto, o simplemente que nuestro USB no da para más :(. Aquí tenéis algo interesante para investigar ;).

Si solo estáis utilizando un dispositivo, podéis utilizar directamente el puntero de la zona de memoria mapeada. Esto sería:

```
d->mm + d->mbuf_info.offsets [d->mmap_info.frame]
```

Con lo que os ahorráis la copia de los datos, pero es necesaria una pequeña modificación del resto del código.

Finalmente, se inicia de nuevo la captura del frame que acabamos de copiar con el mensaje `VIDIOCMCAPTURE` y se actualiza el campo, para que en la próxima llamada a la función trabajemos sobre el otro buffer, que debería contener ya la siguiente imagen.

EL PROGRAMA PRINCIPAL

Ahora que ya tenemos nuestro interfaz de captura de imágenes, estamos en condiciones de escribir nuestra aplicación de captura de imágenes :).

El programa principal, además de utilizar nuestro flamante interface para V4L, mostrará en una ventana las imágenes capturadas. Para ello vamos a utilizar OpenGL, pero para el interfaz entre OpenGL y el sistema de ventanas vamos a utilizar GLUT (GL Utility Toolkit), que va a hacer nuestra vida mucho más fácil.... GLX y Xlib son un poco más engorrosos.

Aquí tenéis el programa principal.

```
DISP d[2];

int main (int argc, char *argv[])
{
  /* Creamos dispositivos */
  d[0] = create_dev();
  d[1] = create_dev();

  /* Inicializa GLUT */
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_RGBA | GLUT_DOUBLE
                                 | GLUT_DEPTH);
  glutInitWindowSize (WIN_WIDTH, WIN_HEIGHT);
  glutCreateWindow ("Sistema Video Vigilancia "
                    "- Occam's Razor");

  /* Inicializa OpenGL */
  gfx_init ();

  /* Inicializa dispositivos de captura.
     Terminamos ante cualquier error */
  open_v4l (d[0], "/dev/video1", &img_w, &img_h);
  start_v4l (d[0]);

  open_v4l (d[1], "/dev/video2", &img_w, &img_h);
  start_v4l (d[1]);

  /* Creamos las Texturas */
  init_texture (img_w, img_h);
  img_size = img_w * img_h * img_bpp;

  /* Configure GLUT callback handlers for events */
  glutDisplayFunc (gfx_pinta);
  glutKeyboardFunc (handle_keyboard);
  glutIdleFunc (handle_idle);

  /*Now we enter the main loop and process GLUT events*/
  glutMainLoop ();
  exit (0);
}
```

Si obviamos por un momento las llamadas a GLUT, el programa es muy sencillo. Hemos utilizado una función adicional (`create_dev`) que simplemente crea dinámicamente un objeto del tipo `DISP` (vamos, hace un `malloc`), y a continuación llamamos a las funciones para inicializar los dispositivos y comenzar la captura.

Lo que no vemos por ningún lado es la llamada a `capture_v4l`. Vale, para encontrar esa llamada tenemos que explicaros un poco como funciona GLUT.

LA APLICACIÓN GLUT

Como ya habréis intuido, todas las funciones de GLUT, empiezan precisamente por glut :). Las primeras cuatro llamadas que nos encontramos son las encargadas de crear la ventana en la que mostraremos nuestro video.

GLUT nos va a permitir utilizar OpenGL mú rápido

Una explicación detallada de GLUT se sale un poco de nuestro objetivo, así que no vamos a dar muchos detalles. Simplemente, las dos primeras llamadas inicializan la librería, y normalmente van a ser siempre así. Las dos siguientes, como os podéis imaginar, permiten definir el tamaño de la ventana, y crearla especificando un título.

El siguiente bloque de llamadas a GLUT lo encontramos hacia el final, y lo reproducimos de nuevo aquí por comodidad:

```
glutDisplayFunc (gfx_pinta);
glutKeyboardFunc (handle_keyboard);
glutIdleFunc (handle_idle);
```

Estas tres funciones nos permiten definir "call-backs". Concretamente, nos permiten decirle a GLUT qué función tiene que ejecutar cuando tenga que actualizar la ventana, cuando se pulse una tecla y cuando no tenga nada que hacer... sí, en esta última es en la que vamos a capturar nuestras imágenes.

Finalmente, la función `glutMainLoop` ejecuta un bucle infinito en el que va llamando, según convenga a cada una de los "call-backs" que hemos definido.

MANEJANDO EL TECLADO

Vamos a empezar por esta función, ya que es la más sencilla. El código es este:

```
void
handle_keyboard (unsigned char key, int x, int y)
{
  /* Process the input */
  switch (key)
    {
    case 'Q':
    case 27:
      exit (0);
      break;
      /* Control de la rotación */
    case 'W':
      xangle += 5;
      break;
    case 'S':
      xangle -= 5;
      break;
    case 'A':
      zangle += 5;
      break;
    case 'D':
      zangle -= 5;
      break;
    default:
      break;
    }

  /* Now refresh the display */
  gfx_pinta ();
}
```

Cada vez que el usuario pulsa una tecla, esta función se ejecuta, recibiendo como parámetro la tecla pulsada. Nuestro manejador, utiliza la tecla Q o ESC (ASCII 27) para terminar la aplicación. Las teclas A, S, D y W se utilizan para actualizar las variables globales xangle e yangle, que utilizaremos para rotar nuestros vídeos en 3D.

CAPTURANDO

La captura se realiza en la función `handle_idle`. Esta función se ejecuta en cada ciclo del bucle principal cuando no hay otra cosa que hacer, es decir, no hay que actualizar el contenido de la ventana, no se ha pulsado ninguna tecla ni se ha movido el ratón.

Esta función sería algo tal que así:

```
void handle_idle (void)
{
  char fname[1024];
  char *im, *im1;
  int  w, h, size, i, temp;

  im = capture_v4l_full (d[0]);
  im1 = capture_v4l_full (d[1]);

  /* Reordena componentes de color */
  for (i = 0; i < img_size; i+= img_bpp)
    {
      temp = *(im + i + 2);
      *(im + i + 2) = *(im + i + 0);
      *(im + i + 0) = temp;
    }

  for (i = 0; i < img_size; i+= img_bpp)
    {
      temp = *(im1 + i + 2);
      *(im1 + i + 2) = *(im1 + i + 0);
      *(im1 + i + 0) = temp;
    }

  /* Actualizamos nuestras texturas */
  change_texture (0, im);
  change_texture (1, im1);

  snprintf (fname, 1024, "video0-%05d.jpg", cnt);
  write_jpeg_file (fname, img);

  /* Redibujamos las nuevas imagenes */
  gfx_pinta ();
}
```

Lo primero que hacemos es obtener las últimas imágenes capturadas por nuestros dispositivos. A continuación tenemos que hacer un pequeño procesado. En nuestro caso, las componentes azul y roja (2 y 0) de cada pixel de la imagen están intercambiadas. Por alguna razón nuestra cámara nos devuelve la imagen en formato BGR en lugar de RGB.

Aunque podríamos dejar que OpenGL se encargara de esto, hemos preferido hacerlo a mano a modo de un sencillo procesado de imagen. Pensad que en un sistema de vídeo vigilancia real, una de las funcionalidades que se suelen requerir es la detección de movimiento, de forma que el sistema solo graba la secuencia de video cuando está pasando algo. Un montón de gigas de vídeo de una imagen estática no valen para nada :).

La captura de imágenes se realiza en el callback IDLE de GLUT

Ese tipo de procesado, o cualquier otra cosa que queramos hacer con las imágenes, implica manipular sus pixels... y eso es lo que hacen esos dos bucles tras las funciones de captura.

Una vez que tenemos nuestras imágenes en el formato que queremos, utilizamos la función `change_texture`, con la que modificaremos las texturas OpenGL que utilizamos para visualizar el video. Muy pronto veremos que hace esta función exactamente.

Finalmente, llamamos a una función `write_jpeg_file` que grabará las imágenes en el disco en formato JPEG.

La función termina ejecutando `gfx_pinta` que es la encargada de mostrar las imágenes en la ventana. Esta es nuestra siguiente víctima!

VISUALIZANDO LA ESCENA

La función `gfx_pinta` utiliza OpenGL para mostrar el vídeo en tiempo real en un espacio 3D (echadle un ojo a la figura). Si sabéis algo de OpenGL, veréis que la función es muy sencilla, sino, os comentaremos solamente un par de cosas para que podáis hacer algunos cambios.

```
void gfx_pinta (void)
{
  glClear (GL_COLOR_BUFFER_BIT |
           GL_DEPTH_BUFFER_BIT);

  /* Ponemos la prespectiva */
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective (45.0,
                  (double)WIN_WIDTH /
                  (double)WIN_HEIGHT,
                  0.2, 20);

  /* Pasamos a la matriz View Model */
  glMatrixMode (GL_MODELVIEW);
  glLoadIdentity();

  /* Posicionamos la cámara */
  gluLookAt (0, 4, 0,  /* Posición del Ojo */
             0, 1, 0,  /* Punto al que miramos*/
             0, 0, 1); /* Arriba */

  /* Rotamos la camara para efecto guay */
  glRotatef (zangle, 0, 0, 1);
  glRotatef (xangle, 1, 0, 0);

  glPushMatrix ();
  glTranslatef (-2.1, 0.0, 0.0);
  glScalef (scale, scale * ASPECT, scale);
  square_texture (0);

  glPopMatrix ();
  glTranslatef (0.1, 0.0, 0.0);
  glScalef (scale, scale * ASPECT, scale);
  square_texture (1);

  glutSwapBuffers ();
}
```

Lo primero que hace la función es limpiar la pantalla (bueno, y el z-buffer, pero eso ahora no nos interesa). A continuación define la matriz de proyección o, en lenguaje llano, la perspectiva de la escena, esto es, el campo de visión (que son 45 grados en nuestro ejemplo), el aspecto de la ventana, el resto de parámetros no nos interesan en este ejemplo.

A continuación se cambia a la matriz de modelo, que es la que tenemos que utilizar para pintar cosas en 3D, y la inicializamos (`glLoadIdentity`), para luego posicionar la cámara con `gluLookAt`. Los comentarios en el código son suficientemente claros no?.

En este punto está todo listo para que nuestros vídeos aparezcan en pantalla. Como queremos ese interfaz 3D súper güay de la muerte, lo primero que hacemos es rotar la escena tanto en el eje Z como en el X. Estos valores los podemos controlar con el teclado como ya hemos visto.

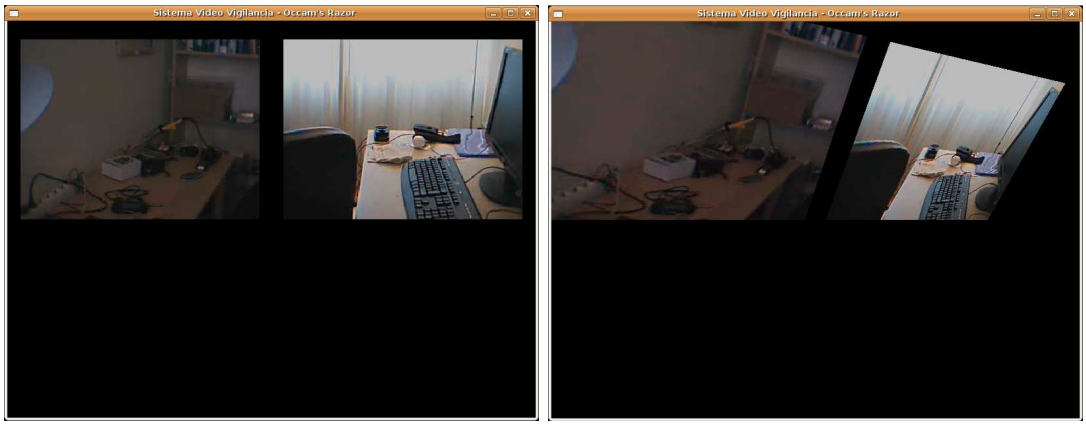
Esta transformación (la rotación definida con el teclado) la vamos a aplicar a los dos vídeos que mostraremos en pantalla, así que, una vez calculada, nos la guardamos para usarla con la segunda imagen. Esto es lo que hace la función `glPushMatrix`, almacena en una pila la transformación actual.

Movemos el primer vídeo un poco hacia la izquierda y los escalamos según el valor de la variable `scale`. Podéis ampliar la función de control de teclado para modificar este valor y hacer vuestros vídeos más grandes o pequeños en pantalla. Como podéis ver, la componente Y se escala de forma diferente para mantener el aspecto de la imagen, esto es, la relación entre el ancho y el alto, en función de las dimensiones de nuestra ventana. Esto es necesario debido a la forma en la que dibujamos los cuadros de nuestro vídeo. Lo veremos un poco más adelante.

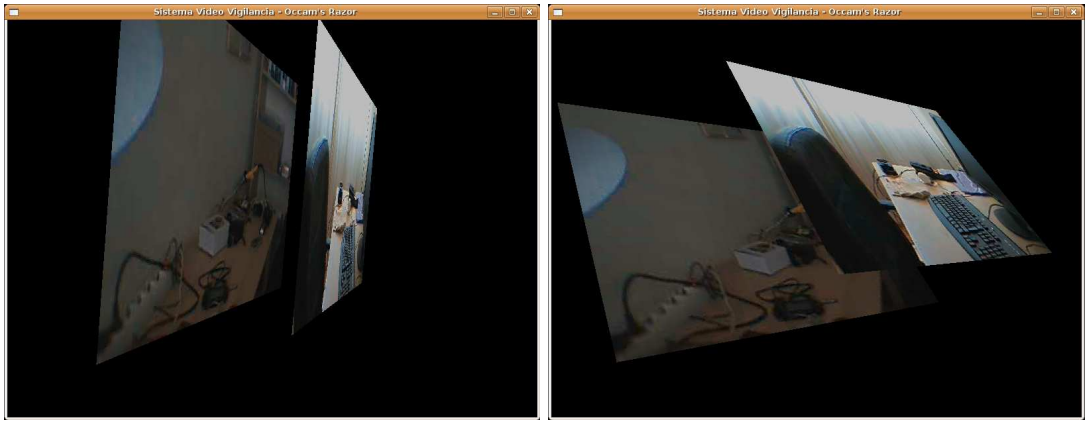
Finalmente dibujamos la imagen utilizando la función `square_texture`. Observad que antes de dibujar la segunda imagen, hemos recuperado nuestra transformación inicial (`glPopMatrix`), para, a continuación, posicionar la segunda imagen un poco a la izquierda.

Nosotros hemos colocado una imagen junto a la otra. Aunque visualmente no parece tan güay, es más práctico. Si queréis el estilo "Aero", tendréis que modificar la componente Z de las traslaciones y hacer que las imágenes se solapen modificando la componente X.

La función termina con una llamada a `glutSwapBuffers` que envía los datos a la tarjeta. Sí, OpenGL también utiliza doble buffer para la visualización. Realmente podéis activarlo o desactivarlo al inicializar la librería (habéis averiguado cual es el flag que hay que eliminar?).



Nuestra aplicación. Izquierda: GUI normal. Derecha GUI 3D



Jugando un poco podemos conseguir efectos más interesantes

GENERANDO LAS IMÁGENES

En la sección anterior hemos visto como componer la escena OpenGL, pero todavía no os hemos contado como visualizar realmente las imágenes.

Esto se puede hacer manipulando el frame buffer directamente, pero nosotros hemos elegido hacerlo utilizando texturas. El uso de texturas proporciona dos ventajas frente a escribir los datos directamente en el frame buffer: La primera es que la técnica es más rápida, y la segunda es que si queremos cambiar el tamaño de la imagen, OpenGL realizará la interpolación por nosotros.

Utilizaremos texturas para la visualización del video en directo

Así, el método consiste en lo siguiente. Primero generamos e inicializamos una textura por cada imagen que queremos mostrar. Utilizamos las imágenes capturadas para inicializar los datos de la textura y finalmente, dibujamos un cuadro sobre el que mapeamos esta textura.

Si habéis seguido los comentarios del programa principal, recordaréis que se nos han quedado tres funciones en el tintero. La primera se utiliza en la función main y se llama `init_texture`. La segunda se utiliza cuando capturamos nuestras imágenes y se llama, `change_texture`. Y finalmente, la última la acabamos de ver en la sección anterior y se llama `square_texture`.

INICIALIZANDO LAS TEXTURAS

La función `init_texture` se encarga de la inicialización de las texturas. La inicialización consiste en decirle a OpenGL que le de "nombre" a las texturas que vamos a utilizar y que inicialice sus atributos. Recordad que aunque nosotros vamos a utilizar la textura para visualizar una imagen, las texturas se utilizan sobre modelos 3D para darles mayor realismo. Cuando hacemos esto último todos estos parámetros toman mucho más sentido.

La función `init_texture` es la encargada de hacer esto. La hemos intentado reducir a la mínima expresión, y el resultado es este:

```
#define MAX_TEXTURE 4

static GLuint texid[MAX_TEXTURE];

void init_texture (int w, int h)
{
  int      i;

  /* Almacenamos las dimensiones para más tarde*/
  tex_w = w;
  tex_h = h;

  /* Genera "Nombres" OpenGL para las texturas */
  glGenTextures (MAX_TEXTURE, &texid);

  for (i = 0; i < MAX_TEXTURE; i++)
    {
      glBindTexture (GL_TEXTURE_2D, texid[i]);

      /* Define el tipo de interpolación */
      /* Para ampliar la textura*/
      glTexParameteri (GL_TEXTURE_2D,
                       GL_TEXTURE_MAG_FILTER,
                       GL_NEAREST);

      /* Para disminuir la textura*/
      glTexParameteri (GL_TEXTURE_2D,
                       GL_TEXTURE_MIN_FILTER,
                       GL_NEAREST);

      /* Inicializa los datos de la textura */
      glTexImage2D (GL_TEXTURE_2D, 0, GL_RGBA8,
                    w, h, 0, GL_RGB,
                    GL_UNSIGNED_BYTE, NULL);
    }

}
```

Nuestro más-que-básico gestor de texturas es capaz de manejar un máximo de cuatro de ellas y almacena sus identificadores en una variable global del módulo llamada `texid`.

Lo primero que hacemos es decirle a OpenGL que necesitamos cuatro texturas y que nos almacene los identificadores en la matriz `texid`. A continuación, tenemos que inicializar cada una de ellas.

Aquí podemos configurar un montón de parámetros de la textura, pero nosotros solo hemos dejado uno de ellos. Obviamente la forma de configurar los parámetros de las texturas es utilizando la función `glTexParameteri`... está claro no? :).

Los dos parámetros que hemos usado le dicen a OpenGL que filtro utilizar cuando tiene que ampliar la textura o hacerla más pequeña. Nosotros hemos utilizado el filtro "del vecino más cercano", que en principio es el más rápido, pero podéis utilizar interpolación lineal, mejorando la calidad del zoom de vuestras aplicaciones, simplemente cambiando este parámetro de `GL_NEAREST` a `GL_LINEAR`.

Finalmente, la función `glTexImage2D` es la que nos permite definir el formato de los datos de imagen que vamos a pasar a OpenGL. Como podéis ver los datos se corresponden con los que hemos utilizado todo el tiempo, ancho, alto y formato (RGB 24bits). El último parámetro es un puntero a los datos de imagen a almacenar en la textura. Nosotros haremos esto en otra función y por eso pasamos el valor NULL.

OpenGL nos permite suavizar las imágenes por Hardware con un simple parámetro

Algunos comentarios dependientes de la implementación concreta de OpenGL que estéis usando: El primero es que muchas implementaciones viejas necesitan que los dimensiones de las texturas (ancho y alto) sean potencias de dos. Si este es vuestro caso (obtendríais una imagen blanca en lugar de la imagen de la cámara), en la función de inicialización tendréis que aproximar las dimensiones de vuestras imágenes a los valores correctos.

El otro comentario, es que algunas implementaciones de OpenGL soportan el formato de texturas GL_BGR. Si recordáis, nuestro dispositivo V4L, nos devolvía los valores en este formato, y tuvimos que añadir un pequeño bucle para ordenarlos antes de enviárselos a OpenGL.... Pues quizás os lo podáis ahorrar.

CAMBIANDO LOS DATOS DE LA TEXTURA

Cada vez que capturamos una imagen, tenemos que actualizar nuestra textura. Esto lo hacemos con la función `change_texture` que podéis ver a continuación.

```
void change_texture (int id, char *in_data)
{
  glBindTexture (GL_TEXTURE_2D, texid[id]);

  glTexSubImage2D (GL_TEXTURE_2D, 0, 0, 0,
  tex_w, tex_h, GL_RGB, GL_UNSIGNED_BYTE, in_data);

}
```

Esta es sencilla no?. La función `glBindTexture` le dice a OpenGL que textura queremos utilizar. Sí, es la misma que utilizamos para configurar la textura tras su creación. El índice que pasamos como primer parámetro nos permite obtener el identificador OpenGL de la textura que nos interesa.

Una vez seleccionada la textura, utilizamos la función `glTexSubImage2D`, que es exactamente igual a la función `glTexImage2D` que utilizamos en la inicialización, pero ésta nos permite actualizar solo una parte de la textura.

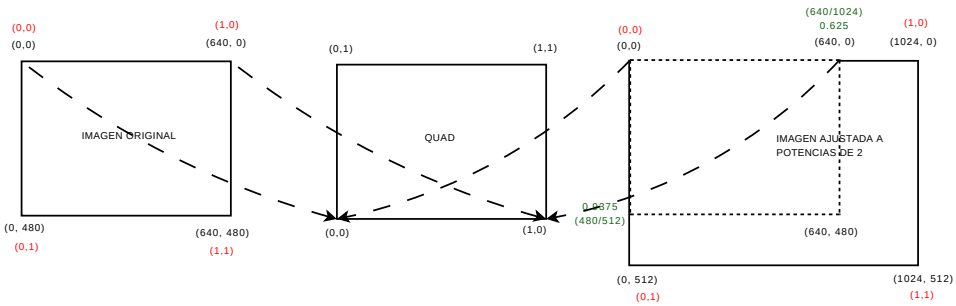
Esta función es en principio más rápida, y además funcionará si nuestra implementación OpenGL tiene la limitación de las potencias de 2, ya que nos permitirá actualizar la parte de la textura que realmente utilizamos.

PINTANDO LAS IMÁGENES

Ya estamos en condiciones de pintar la imagen. Como comentamos, lo que vamos a hacer es pintar un cuadro sobre el que mapearemos una textura. Esta es la forma de hacerlo.

```
void square_texture (int id)
{

  glBindTexture (GL_TEXTURE_2D, texid[id]);

  glBegin (GL_QUADS);
  glTexCoord2f (0.0, 0.0); glVertex2f (0.0, 1.0);
  glTexCoord2f (0.0, 1.0); glVertex2f (0.0, 0.0);
  glTexCoord2f (1.0, 1.0); glVertex2f (1.0, 0.0);
  glTexCoord2f (1.0, 0.0); glVertex2f (1.0, 1.0);
  glEnd ();

}
```

Lo primero que hace la función, es seleccionar la textura que vamos a mapear, con la habitual llamada a `glBindTexture`. A continuación, dibujaremos el cuadrado, para lo que tendremos que decirle a OpenGL cuales serán las coordenadas de sus vértices, y que "punto" de la textura queremos asociar a cada uno de ellos.



Transformaciones de coordenadas para mapeado de texturas

La función `glTexCoord2f`, nos permite seleccionar un punto dentro de la textura. El 0,0 será el punto inicial de nuestra imagen, y el 1,1 se corresponderá con la esquina inferior derecha. Si os fijáis, las coordenadas de la textura y del vértice del cuadro no se corresponden. Esto sucede porque el sistema de coordenadas OpenGL asigna el origen a la esquina inferior izquierda y la imagen que nos devuelve nuestro dispositivo de captura, está definida respecto a la esquina superior izquierda.

Para utilizar nuestra imagen como textura es necesario realizar algunas transformaciones

Observad que si habéis modificado el programa para utilizar texturas con dimensiones múltiplos de potencias de 2, la parte de la textura que contiene la imagen no se corresponde con el tamaño de la textura (hay una parte negra a la derecha). En ese caso, tendréis que sustituir el valor 1.0 en `glTexCoord2f`, por el valor apropiado, que dependerá del tamaño de vuestra imagen.

Finalmente, ¿recordáis el escalado especial de la coordenada Y en la función que dibuja la escena?. Como podéis ver aquí, lo que estamos dibujando es un cuadrado. Si en lugar de un cuadrado, dibujáramos el rectángulo con el aspecto correcto, no sería necesario escalar la coordenada Y de forma especial... La mejor opción dependerá de cada aplicación.

INICIALIZACIÓN DE OPENGL

Para terminar con todo lo relacionado con la visualización, tenemos que comentar una última función, `gfx_init`. Esta función se encarga de inicializar OpenGL, y en general contiene comandos OpenGL que solamente es necesario ejecutar una vez.

En nuestro caso la función es muy sencilla.

```
void gfx_init (void)
{
  glClearColor (0, 0, 0, 0);
  glEnable (GL_DEPTH_TEST);
  glEnable (GL_TEXTURE_2D);
}
```

El primer comando define el color de fondo a utilizar cuando borremos los buffers de color (la imagen en la ventana), utilizando la función `glClear` (la primera llamada en `gfx_pinta`).

Las otras dos funciones permiten activar, respectivamente, el uso del z-buffer y el de las texturas 2D. Tened en cuenta que antes de ejecutar ninguna de las funciones relacionadas con la manipulación de las texturas que acabamos de ver, necesitamos ejecutar esta función para activar el uso de texturas 2D. Si no lo hacemos obtendremos una imagen vacía.

El uso de z-buffer es necesario si las imágenes se van a solapar de alguna manera en la escena. Si no activamos el z-buffer, las imágenes se visualizarán en el orden en el que las pintemos en nuestro programa, independientemente de su posición real en el espacio 3D.

Si no queréis hacer filigranas con las texturas y vuestras imágenes van a aparecer en una posición fija en la pantalla (lo usual en un sistema de vídeo vigilancia), podéis desactivar el z-buffer, ahorrando memoria y trabajo a vuestra tarjeta gráfica. En este mismo caso, probablemente os interese más utilizar una proyección ortográfica que la perspectiva que estamos utilizando... solo estamos dando pistas...

VOLCANDO IMÁGENES

El último elemento que nos queda es la función para volcar las imágenes capturadas al disco. Desde el punto de vista del almacenaje nos interesa utilizar un formato comprimido, pero antes os vamos a comentar como utilizar el formato Netpbm, que es tremendamente sencillo y si lo utilizamos junto con zlib, nos puede solucionar el problema mú rápido.

Aquí tenéis la función para grabar vuestras imágenes:

```
int write_image (char *fname, char*data)
{
  FILE *f;

  f = fopen (fname, "wb");
  fprintf (f, "P6\n#Occam's Razor "
              "Video Vigilancia\n"
              "%d %d\n%d\n",
           img_w, img_h, 255);
  fwrite (data, img_size, 1, f);
  fclose (f);
}
```

Más rápido no se puede XD. El formato ppm, utiliza una sencilla cabecera ASCII, en la que se indican las dimensiones de la imagen y su profundidad de color (el valor 255 indica 8 bits por componente). A continuación escribimos los datos y listo.

El formato PPM es muy conveniente para volcar imágenes de forma sencilla

El problema de PPM es que los datos no se comprimen en absoluto. Cada imagen captura por nuestra aplicación ocupa unos 900Kb. La misma imagen en formato jpg ocupa unos 19Kb, y por eso os vamos a contar como usar este formato.

En la sección Ratas de Biblioteca de el número 1 de Occam's Razor, os contábamos como utilizar zlib para almacenar ficheros. Alguien quiere modificar la función de arriba y ver cuanto puede comprimir nuestras imágenes zlib?. Esperamos vuestros resultados ;)

USANDO JPEG

Al principio del artículo os decíamos que íbamos a utilizar libjpeg para almacenar las imagenes, y como lo prometido es deuda, aquí tenéis la función para almacenar una imagen en formato JPEG en el disco:

```
#include <jpeglib.h>

int color_space = JCS_RGB;
/* JCS_GRAYSCALE para nivels de gris */

int
write_jpeg_file(char *filename, char *buffer,
                int w, int h, int bpp )
{
  struct jpeg_compress_struct cinfo;
  struct jpeg_error_mgr jerr;

  JSAMPROW row_pointer[1];
  FILE *outfile = fopen( filename, "wb" );

  if ( !outfile )
    {
      printf("Error opening output jpeg "
             "file %s\n!", filename );
      return -1;
    }

  cinfo.err = jpeg_std_error( &jerr );
  jpeg_create_compress(&cinfo);
  jpeg_stdio_dest(&cinfo, outfile);

  cinfo.image_width = w;
  cinfo.image_height = h;
  cinfo.input_components = bpp;
  cinfo.in_color_space = color_space;

  jpeg_set_defaults( &cinfo );

  jpeg_start_compress( &cinfo, TRUE );

  while(cinfo.next_scanline<cinfo.image_height)
    {
      row_pointer[0]=&buffer[cinfo.next_scanline*
                             cinfo.image_width*
                             cinfo.input_components];
      jpeg_write_scanlines( &cinfo, row_pointer, 1);
    }
  /* similar to read file, clean up after
     we're done compressing */
  jpeg_finish_compress( &cinfo );
  jpeg_destroy_compress( &cinfo );
  fclose( outfile );
  return 1;
}
```

Bueno, esta función la podéis encontrar en los ejemplos que se distribuyen con el código fuente de libjpeg. En el mismo fichero podéis encontrar la función para leer un fichero jpeg del disco.

El código es autoexplicativo, y además, así es como funciona la librería, no hay mucho que rascar.

libjpeg nos permite grabar imágenes en formato JPEG de forma sencilla

Con esta función podremos almacenar las imágenes capturadas en formato jpeg. Esto es más o menos equivalente a almacenar la secuencia de video en formato MJPEG, lo que no está nada mal.

A JUGAR!

Bueno, este artículo ha sido un poco largo, pero hemos tratado un montón de cosas distintas que nos dan mucho juego. Aquí van algunas propuestas:

- Añadir soporte de red a la aplicación. Con lo que os hemos contado en los números anteriores de Occam's Razor, en esta misma sección, eso debería estar *chupao*!
- Captura de vídeos panorámicos. Utilizando varias cámaras, podemos modificar nuestro programa para poder generar vídeos panorámicos, utilizando un par de webcams :).
- Vigilando en estéreo?. Tenéis unas gafas de esas rojas y azules?. Sabéis lo que es un anaglifo? (buscad Anaglifo en la wikipedia :)). Sabéis que con el código de este artículo, podéis generar un anaglifo modificando una sola línea del código!!!! (bueno, y colocando las cámaras correctamente).
- Grabar la secuencia de vídeo en formato AVI, MPEG o lo que os apetezca (pista... ffmpeg).

Y cualquier otra cosa que se os ocurra. No os olvidéis de enviarnos vuestros experimentos!!. Esta vez hay cosas de sobra para jugar!. Hasta el próximo número. ■

RECURSOS

Driver GSPCA y Spcaview
http://mxhaard.free.fr/download.html

FAQ de motion con mucha información interesante
http://www.lavrsen.dk/twiki/bin/view/Motion/FrequentlyAskedQuestions

Tutoriales y ejemplos OpenGL
http://www.opengl.org/code/

Formato PPM
http://netpbm.sourceforge.net/doc/ppm.html

Libjpeg a través de la wikipedia
http://en.wikipedia.org/wiki/Libjpeg

Para los que quieran jugar
Anaglifos
http://es.wikipedia.org/wiki/Anaglifo
Página web de FFMPEG
http://en.wikipedia.org/wiki/Libjpeg

Cuando la dirección de la revista me propuso escribir este artículo, me di cuenta de que sin pretenderlo me he convertido en una especie de historiador tecnológico aficionado. Aprovecho para recomendaros (si no los habéis leído ya) mis anteriores artículos "Mi Historia de las Telecomunicaciones" y "El telégrafo de Gauss", ya sabéis que nunca puedo resistirme a hacer un poco de autobombo. En fin, ya que este número va dedicado a las tecnologías de seguridad resulta muy propio un artículo sobre la historia de la criptografía. La criptografía es la ciencia (o la técnica) de "poner en clave" mensajes de forma que un espía que logre interceptarlos no pueda entenderlos pero aquellos que conozcan la clave (o claves) adecuada(s) logren fácilmente recuperar la información inicial. Como veremos, la criptografía existe desde tiempos muy antiguos, aunque ha sido la llegada de los ordenadores la que ha logrado crear sistemas realmente potentes y seguros.

Si me permitís repetirme un poco volveremos a la definición. El término *criptografía* procede de la unión de dos palabras griegas: krypto (ocultar) y graphos (escribir). Se trata por tanto de "escritura oculta"... es decir: la criptografía es un conjunto de técnicas que permiten poner un mensaje cualquiera en un formato ininteligible para todos, excepto para aquellos que poseen una información adicional (clave). Por supuesto, la clave es una información necesaria para realizar el proceso de cifrado (encriptado o puesta en clave) y también para el proceso de descifrado. Según el algoritmo empleado, la clave puede ser un número, una letra, un conjunto de bits o, incluso, cosas más raras como una tabla de símbolos.

Una suposición fundamental en todos los estudios criptográficos es que la seguridad se debe basar en el secreto de la clave, nunca en el secreto del algoritmo. Los algoritmos de cifrado deben ser públicos. Un sistema que se base en el secreto del algoritmo quedará comprometido si se descubre éste. Sin embargo, si se descubre una clave, podremos seguir operando el sistema cambiándola.

Para terminar la introducción veremos un poco de terminología sobre el cifrado, a saber:

- **Mensaje llano**: es el mensaje inicial, antes de ser cifrado, por tanto todavía legible.
- **Criptograma**: mensaje una vez cifrado.
- **Criptoanálisis**: son las operaciones que realiza un espía para intentar averiguar el mensaje llano.

  Normalmente, se supone que el espía conoce el algoritmo pero no la clave. El criptoanálisis es diferente si el espía sólo dispone de un criptograma, de varios o dispone de texto seleccionado (criptogramas para los que conoce el texto llano). Esta última es la situación más favorable para el ataque. Pensad que hay un criptoanálisis que nunca falla: probar todas las claves posibles (método de la fuerza bruta). Un buen algoritmo de cifrado debe estar pensado para que el único ataque posible sea la fuerza bruta y, además, para que este ataque no sea viable. Para evitar la fuerza bruta, debe haber muchas claves posibles (longitud de clave de muchos bits) de forma que comprobar todas consuma una cantidad de tiempo excesiva (del orden de siglos).

Criptografía significa literalmente escritura oculta

Como sabréis (creo que es una de las cosas que enseña esta revista), la mente humana es infinita y hasta ha habido estudios matemáticos sobre cómo crear un algoritmo matemáticamente seguro ante la fuerza bruta. Como no, se debe al creador de la teoría de la información (Claude E. Shannon). ¿Cómo es posible esto? -os preguntaréis. Si se puede descifrar sabiendo la clave, cualquiera puede ponerse a probar claves: **CIERTO**.

El problema "no resuelto" es: ¿Cómo hace el criptoanalista para saber qué clave es la buena, si no conoce el texto en llano? Nuestro malvado espía se tiene que basar en tomar una decisión binaria ante cada prueba: ¿El mensaje descifrado tiene sentido o no? Si un algoritmo es atacado por fuerza bruta y obtenemos varios mensajes coherentes probando claves distintas, no sabremos cuál de ellas es. Realmente, si el espía puede reducir la incógnita sobre la clave, en el próximo criptograma que analice tendrá menos claves que probar. Por otra parte, la coherencia del mensaje es algo totalmente dependiente del tipo de mensaje. Si sabemos que son imágenes JPEG deben cumplir un formato y además salir algo con sentido al representarlas. ¿Algo con sentido? algunas imágenes microscópicas pueden no parecer con mucho sentido

CIFRADOS DE SUSTITUCIÓN

La técnica de cifrado más antigua es la de sustitución. Como su nombre indica, consiste en sustituir cada símbolo (letra) del alfabeto con el que representamos nuestros mensajes por otro símbolo de otro alfabeto (o del mismo).

El primer método sistemático de cifrado del que se tiene noticia fue utilizado por los militares romanos y se conoce todavía hoy como "Cifrado de César". Se le llama así porque se encontraron documentos cifrados con esta técnica por Julio César y por su heredero Augusto; sin embargo, no está demostrado que César fuera el primero en emplearlo.

El cifrado de César fué el primer sistema de cifrado sistemático que se conoce.

El cifrado de César se basa en una sustitución de las letras del alfabeto latino (el nuestro) por ellas mismas según una clave que no es más que un desplazamiento. Esto es: si la clave es +1 cada letra se sustituye por la siguiente del alfabeto, en vez de CÉSAR (ignorando el acento) escribiríamos DFTBS. Nótese que en aquella época el cifrado se realizaba de "memoria". Lógicamente, el desplazamiento es circular: después de la Z viene la A. También son posibles los desplazamientos negativos aunque -1 sería equivalente a 25 (si hay 26 letras). Nótese que para N letras en el alfabeto hay N-1 posibles claves y un desplazamiento d es equivalente al d+N. El número tan reducido de claves se debe a que estamos conservando el orden de las letras. Si consideramos sustituir el alfabeto latino por él mismo sin ninguna restricción el número de claves posibles será N!

El cryptoanálisis de la sustitución es muy simple.

Ya es un tópico pero es este momento es inevitable recordar que en el nombre de la computadora de la película 2001: "HAL" había un mensaje cifrado. ¿Alguien no lo sabía? Pistas: el método empleado ya lo hemos comentado en este artículo y el mensaje llano tiene algo que ver con la informática.

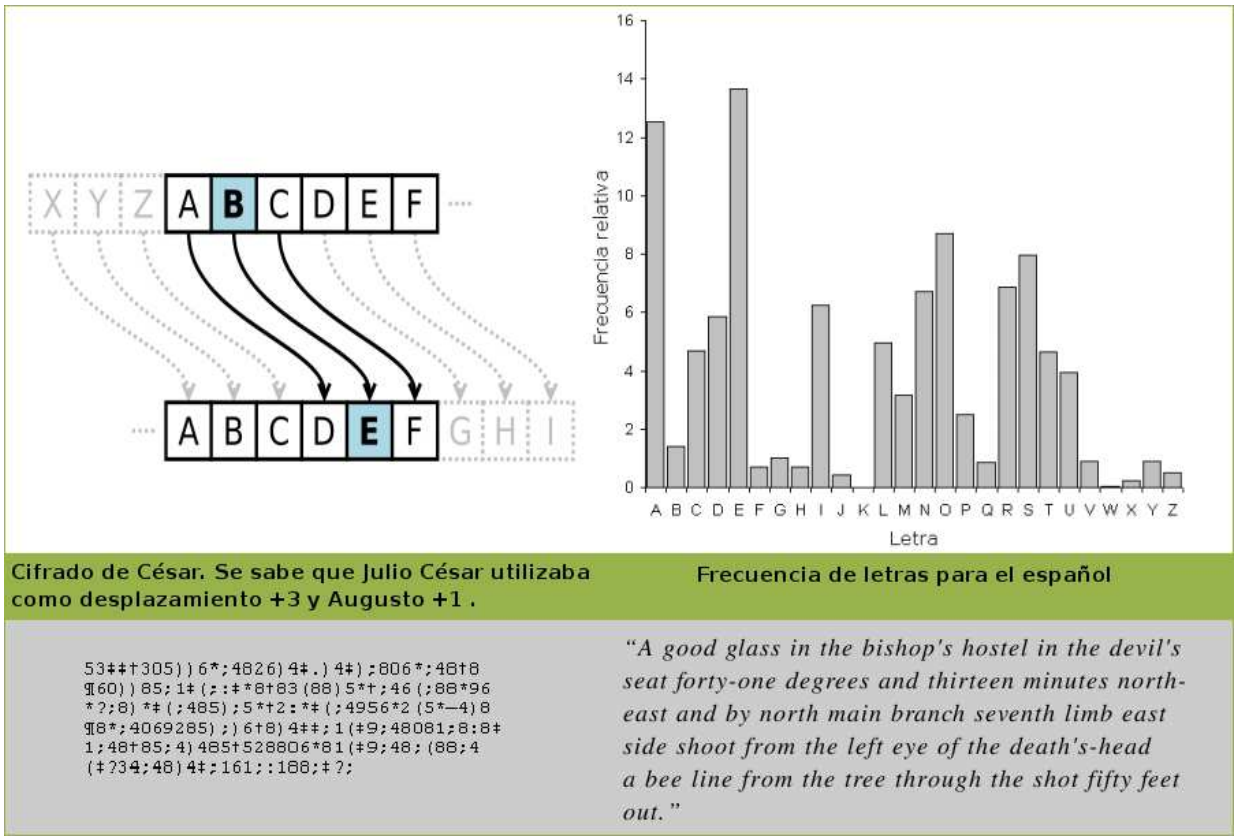
Otro tipo de cifrados de sustitución son los que inventan nuevos alfabetos donde cada símbolo representa una letra. La clave pasa por conocer la correspondencia entre letras y símbolos. Realmente estos métodos no son más seguros que una sustitución de las letras estándar por ellas mismas. Sigue habiendo N! ordenaciones.

El criptoanálisis de la sustitución es muy simple y la manera más gratificante de aprenderlo sería leer "El Escarabajo de Oro". En este relato (nunca he sabido si es un cuento largo o una novela corta) los protagonistas descifran un mensaje pirata (con alfabeto de símbolos inventados mezclados con números y signos de puntuación). La técnica que usan es probabilística: ya que la E es la letra más usada del inglés debe ser la que corresponde con el símbolo más repetido. Conociendo la frecuencia de las letras en el idioma podemos hacer suposiciones de este tipo sobre dos o tres caracteres e intentar descubrir alguno más. Para eso ayudan mucho las partículas de cada idioma, en el ejemplo del escarabajo de oro logran descifrar la T y la E y ven muchas veces la palabra T4E, con lo que "4" debe ser la H (recuérdese que era un mensaje en inglés). ¿Quién es el autor de este relato? Se trata de un novelista norteamericano genial aunque de vida algo atormentada: Edgar Allan Poe.

Por supuesto que si sabemos que es un cifrado César la fuerza bruta nos puede ayudar (el número de claves es ridículamente bajo). Ojo... no hay constancia de que ningún enemigo lograra, en su época, descifrar los mensajes cifrados por César por lo que confirmamos (por si no lo sabíamos) que el valor de una tecnología es siempre relativo a su época (también hay que decir que la mayoría de esos enemigos no sabían leer).

La criptografía como disciplina matemática

Realmente la criptografía puede verse como una parte de las matemáticas dedicada a crear funciones inestables. Un algoritmo criptográfico es una función que opera con dos variables: el mensaje llano y la clave para obtener un resultado: el criptograma. Esa función debe ser biyectiva, esto es inversible ya que debe existir una función inversa que a partir del criptograma y la clave permita obtener de nuevo el mensaje llano. Para que una función "cifre" debe ser inestable, esto es: al modificar muy poco el mensaje o la clave el criptograma debe variar mucho... eso asegura que es difícil de invertir sin conocer la clave ya que no podemos hacer aproximaciones sucesivas.



Criptograma de "El Escarabajo de Oro" y su texto llano una vez descifrado. Nótese que se eliminan los espacios para lograr una mayor confusión del lector. .

Otro método de sustitución muy utilizado hasta el siglo XIX (y muy seguro en aquella época) es el método de Vigenère. Realmente, se trata de una mejora del método de César. La idea es muy simple... definamos M claves para el método de César (M números entre 1 y 25 para un alfabeto de 26 letras). Si aplicamos el cifrado de César con la primera clave a la primera letra, el mismo método a la segunda letra con la segunda clave... tendremos un cifrado César con clave variable. A partir de M aplicaciones volvemos a la primera clave.

El acierto de este método es que una misma letra será cifrada de forma diferente aunque aparezca muchas veces. Si cada clave César, se representa por una letra (donde la A es el desplazamiento 0 y la Z el 26) la clave completa (M letras) se podrá escribir como una palabra. No hay longitud mínima ni máxima. De hecho, la longitud de la clave es el gran secreto del método.

El cifrado de Vigenère realiza una sustitución polialfabética.

Este método se consideró invulnerable hasta que el militar prusiano Friedrich Kasiski publicó un método de criptoanálisis válido en 1.863. El método de Kasiski se basa en buscar palabras repetidas en el criptograma. Lo más probable es que sean debidas a que entre ellas el número de caracteres es múltiplo de M. Buscando todas las repeticiones podemos determinar M (como el máximo común divisor de las distancias entre repeticiones). Hecho eso, hay que descifrar M cifrados César.

El cifrado de Vigenère es un ejemplo de sustitución polialfabética. Esto es: se hace sustitución con una secuencia de claves diferentes. El problema del criptoanálisis realmente estriba en predecir la secuencia.

Hacia el 1920 se empezaron a fabricar máquinas capaces de realizar cifrados polialfabéticos. Se trataba de máquinas electro mecánicas, donde se utilizaba el giro de una serie de discos que contenían contactos eléctricos. Esas máquinas tenían el aspecto de máquinas de escribir en las que se tecleaba el texto llano y se obtenía el criptograma. A veces, el resultado se obtenía impreso, y en otras versiones se tenía una bombilla por cada letra del alfabeto y un operador debía copiar "al dictado".

La máquina de cifrado electromecánico más famosa de la historia es la alemana "Enigma". Las primeras versiones datan de 1919 y, ya desde entonces, existían versiones comerciales que se compraban y vendían libremente. Algunas de esas primeras máquinas fueron utilizadas por españoles e italianos en la guerra civil española y sus mensajes fueron descifrados por criptoanalistas del ejército británico.

En esa época, el ejército alemán comenzó a utilizar una "versión militar" de enigma muy superior a la comercial. Los primeros esfuerzos británicos por romper el cifrado fueron inútiles (en tiempo de paz, pero temiendo ya el inicio de un conflicto). El primer avance en el criptoanálisis de enigma vino del matemático polaco Marian Rejewski que descubrió ciertas regularidades en el comportamiento de enigma debidas a la repetición de palabras en el texto llano. Sin embargo, el número de cálculos a realizar era excesivo para su capacidad de trabajo y no pudieron descifrar el código. El descubrimiento de Rejewski se realizó antes de que el ejército nazi invadiera Polonia. Cuando la sospecha de invasión fue inminente, el gobierno polaco decidió traspasar a la inteligencia británica todos sus conocimientos.

La máquina alemana Enigma, es el dispositivo cifrado electromecánico más famosa de la historia.

El gobierno británico montó un centro de criptoanálisis en Bletchley Park (80 Km al norte de Londres) cuyo principal objetivo era romper enigma. El equipo de Bletchley Park, dirigido por Alan Turing, consiguió romper el código de enigma construyendo la "bomba": una máquina electromecánica de cálculo que ya había sido propuesta por el equipo de Rejewski y que se puede considerar uno de los primeros ordenadores de la historia.

Máquina Enigma.

La existencia del centro de criptoanálisis de Bletchley Park y los trabajos realizados allí fueron estrictamente secretos hasta la década de los 60.

A pesar del gran trabajo de Turing y su equipo quedaron algunos mensajes sin descifrar. Los últimos fueron traducidos en ¡¡¡2006!!! por el llamado "proyecto-M4". Como ejemplo, tenemos el siguiente:

"nczwvusxpnyminhzxmqxsfwxwlkjahshnmcoccakuqp
mkcsmhkseinjusblkiosxckubhmllxcsjusrrdvkohulxwc
cbgvliyxeoahxrhkkfvdrewezlxobafgyujqukgrtvukam
eurbveksuhhvoyhabcjwmaklfklmyfvnrizrvvrtkofdan
jmolbgffleoprgtflvrhowopbekvwmuqfmpwparmfhag
kxiibg"

"Señal de radio 1132/19. Contenido: Forzados a sumergirnos durante ataque, cargas de profundidad. Última localización enemiga: 8:30h, cuadrícula AJ 9863, 220 grados, 8 millas náuticas. [Estoy] siguiendo [al enemigo]. [El barómetro] cae 14 milibares. NNO 4, visibilidad 10."

CIFRADOS POR PERMUTACIÓN

Los métodos de permutación se basan en cambiar el orden de los símbolos en vez de sustituir estos. Si agrupamos los símbolos de M en M y los trasponemos, habrá M! transposiciones posibles (M! claves).

Fijaos que los métodos de sustitución son "orientados a símbolos" mientras que éstos son orientados a bloques (hay que agrupar un bloque de símbolos que se cifra entero). También hay que decidir qué se hace cuando al final del mensaje tengamos un bloque incompleto (algo que ocurrirá casi siempre). Las soluciones pasan por completar el bloque (con el alfabeto empezando en la A o, mejor, con texto aleatorio). Al descifrarlo (sabiendo la clave) tendremos algunos caracteres extra pero los descartaremos porque "no tienen sentido".

Criptoanálisis: si logramos averiguar el tamaño del bloque (tal vez viendo que todos los mensajes tienen longitud múltiplo de M... que será el máximo común divisor de las longitudes), podremos dividir el criptograma en bloques permutados... Si para alguno de ellos, vemos una hipótesis con sentido (una permutación que da lugar a una palabra o frase válidas) la podemos probar para el resto.

Las permutaciones nunca o casi nunca se usan como método único sino que se combinan con sustituciones u otras operaciones.

PERMUTACIONES Y SUSTITUCIONES: DES

A mediados del siglo XX, el conocido investigador Claude Shannon llegó a afirmar que una combinación de muchas permutaciones alternadas con sustituciones podría dar lugar a cifrados muy seguros... Esa idea fue aprovechada algunos años después cuando se dispuso de la tecnología necesaria para automatizar ese tipo de procesos.

El algoritmo DES (Data Encryption Standard) fue creado en los años 70 y, como su nombre indica, pretendía ser "la solución definitiva" al problema de cifrado. Por supuesto que las soluciones definitivas no existen pero DES fue una solución válida durante muchos años y fue el primer cifrado que se aplicó masivamente en aplicaciones informáticas.

DES nació por iniciativa del NBS (National Bureau of Standards) u oficina de estándares del ministerio de comercio americano (por tanto, hablamos de estándares civiles). Ahora el NBS se llama NIST (National Institute of Standards and Technology, www.nist.gov). Para decidir un estándar de cifrado se convocó un concurso en el que fue elegida la solución presentada por IBM (que provenía de mejorar un algoritmo anterior denominado Lucifer). Desde entonces existe una polémica no resuelta porque se dice que el gobierno (concretamente la agencia de seguridad nacional, NSA) obligó a modificar el algoritmo para que fuese menos seguro. De todas formas, hasta el año 1992 no se publicó una forma de hacer un ataque más eficiente que la fuerza bruta y requiere tener el criptograma correspondiente a ¡¡¡247!!! textos planos determinados... ALGO TOTALMENTE IRREAL. En 1998 se creó un ordenador especial (conocido como Deep Crack) para romper el DES. Fue capaz de romper el cifrado por fuerza bruta en 56 horas.

El DES es un algoritmo de cifrado por bloques.

El DES es un cifrado por bloques. Divide el texto llano en bloques de 64 bits a los que aplica una clave de 56 bits y obtiene un criptograma de 64 bits. Para lograrlo, realiza una complicada combinación de permutaciones y sustituciones. En la edición antigua del excelente libro de A. S. Tanembaum: "Redes de Ordenadores" (o en su edición inglesa, "Computer Networks"), se describía detalladamente la operación del algoritmo e incluso se incluía el código (en pascal) necesario para programarlo. Hablo de la edición antigua porque fue la que leí... imagino que la edición actual seguirá teniendo esa información. Mientras dudaba en ir o no a la biblioteca a comprobarlo, busqué en la mayor biblioteca de la historia y encontré esto: Free Encription/Cryptographic Libraries. Si la página no miente, ahí tenéis código libre C++ para DES y para muchos algoritmos más (muchos de ellos mencionados en este artículo).

La debilidad de DES fue siempre la longitud de la clave. Aunque se creó en un momento en el que era inabordable un ataque por fuerza bruta, los grandes avances en potencia de cálculo llegaron a hacerlo posible en algunos años (aproximadamente 30). Probablemente, ese sea un problema de todos los algoritmos... la viabilidad o no de la fuerza bruta cambia con el tiempo.

Decíamos que en un buen algoritmo las modificaciones pequeñas de la clave o del llano deben afectar mucho al criptograma. DES cumple esa condición a la perfección. Se ha comprobado que la modificación de 1 bit en el llano o en la clave provoca que los bits del criptograma cambien con probabilidad $\frac{1}{2}$, esto es: en media cambiarán la mitad... que es el cambio más impredecible que puede haber.

DES ya no se utiliza en la práctica, aunque lo que sí podemos encontrar todavía es una variante llamada triple DES. Triple DES (TDES o 3DES) consiste en aplicar DES tres veces consecutivas a los 64 bits de entrada, si tenemos tres claves diferentes la longitud de la clave global será de 168 bits. Se han publicado ataques que consiguen reducir el número de claves a probar, de forma que se demostró que triple DES ofrece la seguridad de una clave más corta (aunque haya 2.168 claves llega con probar 2.112). Por eso, podéis oír o leer que en el triple DES la tercera clave suele ser igual a la primera (2 claves diferentes pero DES se aplica tres veces). TDES es un algoritmo que va desapareciendo porque es lento frente a otros más modernos (DES ya era lento y aquí hay que ejecutarlo 3 veces) pero, en su momento, sirvió para que DES volviera a ser seguro y se pudieran reutilizar los sistemas software (y hardware) que implementaban DES.

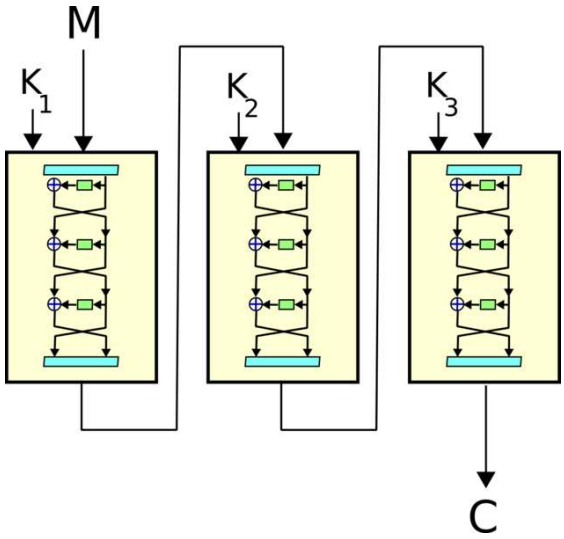


Diagrama de bloques de triple DES.

DES apenas se utiliza en la actualidad. Sin embargo, su variante triple DES todavía se puede encontrar por ahí.

Todos los algoritmos que comentamos hasta ahora son simétricos. Esto es: la clave de cifrado es la misma que la de descifrado. Eso crea un problema para la seguridad en redes: ¿Cómo se distribuyen las claves? Si se hace de forma offline (un señor con un diskette o algo) será poco práctico. Eso era lo que se hacía hasta los años 90 y el boom de Internet. Si se distribuye la clave llana el intruso sólo tendrá que interceptar ESE mensaje. A partir de ese momento, ya podemos usar el mejor método jamás inventado que estaremos haciendo el tonto.

Para evitar esos problemas nació el cifrado asimétrico, también llamado muchas veces "cifrado de clave pública".

CRIPTOGRAFÍA ASIMÉTRICA. RSA

¿En qué consiste eso? Supongamos que se logra crear un método donde la clave K1 sirve para cifrar y la K2 para descifrar. Por supuesto K1 y K2 estarán relacionadas pero debería ser muy difícil obtener K1 a partir de K2 o al revés... eso debería ser tan difícil como descifrar los criptogramas.

Si yo quiero que me envíen mensajes cifrados puedo ¡¡¡publicar K1 sin más!!!. Si sólo yo conozco K2, sólo yo podré descifrarlos, ¿NO?

La cosa se puede complicar más... Imaginemos que creo otras dos claves F1 y F2 y ahora hago pública F2. ¿Para qué puede servir? Pongamos que tomo un texto llano que quiero enviar y le añado algún código de redundancia (CRC 16 ó 32). Si lo cifro con F1 y lo envío todos podrán leerlo (F2 es pública) y, además, todos podrán comprobar el CRC. ¿Eso qué demuestra? Demuestra que quién lo generó conocía F1. Como sólo yo conozco F1 acaba de nacer la "firma digital" (sólo yo puedo generar esos mensajes, aunque todos pueden verlos y comprobar la firma).

El algoritmo más conocido para criptografía asimétrica es el RSA.

El algoritmo más conocido para criptografía asimétrica es el RSA (inventado por Rivest, Shamir y Addleman). El algoritmo se basa en los llamados cuerpos y anillos finitos. Si definimos el conjunto de números enteros de 0 a M-1, y hacemos sumas y productos módulo M, tendremos:

- Un cuerpo si M es primo.
- Un anillo si no lo es.

Si K1 y K2 son números del conjunto (enteros entre 0 y M-1) y además K1*K2 = 1 (mod M), tendremos:

- Para un número LL (texto llano), el número $C = LL^{K1}$ es muy diferente a LL. Podríamos llamar a C "texto cifrado".
- Además: $C^{K2} = (LL^{K1})^{K2} = LL^{(K1*K2)} = LL^1 = LL$ (todo módulo M, claro). Esto es: ciframos conociendo $K1$ y desciframos conociendo $K2$.

Repasemos, para que esto realmente sea cifrado de clave pública, todos deben conocer M y K1 pero K2 debe permanecer secreto. Además fijaos que si publicamos K2 y guardamos K1 el efecto será el mismo... pero los criptogramas se obtendrán como $C = LL^{K2}$.

Puede parecer un poco raro que esto realmente sea un cifrado seguro... para que lo sea deberíamos cumplir:

- $M = (p-1)*(q-1)$ donde p y q son números primos grandes (como de 100 cifras) y distintos.
- K1 no debe compartir factores primos con M y K2 tampoco... de hecho existe un algoritmo para calcular K1 y K2 (algoritmo extendido de Euclides).

¿De dónde viene la seguridad? De que los números primos son una de las grandes cuestiones no resueltas por las matemáticas. El único algoritmo fiable para saber si un número es primo o no es ponerse a dividir, existen tests de primalidad pero no son 100 % fiables. Lo que es peor, la única forma de descomponer un número en factores es ponerse a dividir... Dicho en una frase: con los primos sólo conocemos un método, "la fuerza bruta". Basta que los primos sean grandes para ponerlo muy difícil.

La seguridad del método RSA se basa en que si p y q son dos primos muy grandes, $M = (p-1)*(q-1)$ es una operación muy fácil de realizar pero muy difícil de invertir (hay que factorizar M y eso sólo se puede hacer por fuerza bruta). Si se conocen p y q, se puede calcular K1 a partir de K2 o al revés, esto es: el cifrado habrá caído.

El algoritmo RSA es lento ya que tiene que operar con precisión sobre números enormes. Eso hace que el cifrado asimétrico casi nunca sea el único método utilizado en una comunicación segura. Más bien es un método de distribución de claves para utilizar después algoritmos simétricos (protocolo Diffie-Hellman).

Buscando información para el artículo me "encontré" con el número primo de Mersenne. Ese número es el primo más grande conocido (por lo menos el más grande publicado hasta la fecha de edición de ese artículo). Vale:

$$2^{13466917} - 1$$

¿Os imagináis que se publique el "mayor número par" o el "mayor múltiplo de 5"? ¿Y que encima le pongan el nombre del matemático que lo encontró? Yo siempre he pensado que los números primos son una especie de "error de la naturaleza"...

En 1984 fue publicado un sistema de prestaciones equivalentes al RSA y libre de patentes. Lo diseñó el matemático egipcio Taher Elgamal trabajando en Stanford. El cifrado de Elgamal (a veces, llamado El Gamal) se basa en otro problema de la matemática discreta: el logaritmo discreto. El problema es el siguiente: en un espacio discreto módulo M, dados los números x y a (enteros entre 0 y M-1), se trata de calcular otro entero y tal que $x = a^y$ (mod M). Se dice que **y** es el logaritmo discreto de x en base a (módulo M). Dada la naturaleza discreta del problema, no existe otra solución que probar con todos los enteros entre 0 y M-1. Basta que M sea un número primo grande para que el problema del logaritmo discreto sea imposible de resolver en tiempo razonable.

En el método de Elgamal la clave privada es K1 y la pública $K2 = g^{K1} (\text{mod M})$. Si fuéramos capaces de resolver el logaritmo discreto podríamos calcular K1 a partir de K2 y romper el cifrado. El parámetro g recibe el nombre de generador y es un número elegido aleatoriamente entre 0 y M-1.

Las fórmulas de cifrado y descifrado son:

1. **Cifrado**:
   - Elegir b aleatorio entre 2 y M-2.
   - $C1 = gb$ (mod M).
   - $C2 = (K2)^b LL$ (mod M).
   - $C = (C1, C2)$ (mod M).
2. **Descifrado**:
   - Haciendo $C1^{(M-1-K2)} C2$ (mod M), se obtiene de nuevo LL.

El método de Elgamal está libre de patentes.

MÉTODOS MÁS ACTUALES: IDEA, BLOWFISH, AES...

Una vez recorridos los que creo que son los hitos principales del cifrado desearía presentar brevemente los métodos actuales:

- **IDEA**: su nombre viene de "International Data Encryption Algorithm". Es un algoritmo publicado en 1991 que se propuso como estándar europeo (una iniciativa similar al DES norteamericano). Igual que DES cifra por bloques de 64 bits pero la clave es de 128. El algoritmo IDEA se basa en realizar repetidas veces una serie de operaciones binarias: O-exclusiva (XOR) bit a bit, Suma módulo 216, multiplicación módulo 216+1. La filosofía es similar al DES (reiteración de operaciones simples) pero es mucho más moderno y la clave es mucho más larga con lo que es mucho más seguro. IDEA se utiliza en PGPv2.0 y hasta la fecha no se ha publicado ningún ataque válido.
- **Blowfish**: este método fue diseñado por Bruce Schneier en 1993. De nuevo es un cifrador de bloques de 64 bits. La longitud de la clave es variable y va desde 0 a 448 bits. De nuevo, la filosofía es muy similar al DES ya que se basa en varias rondas de operaciones XOR y sustituciones realizadas por tablas de entradas/salidas llamadas cajas-S (S-boxes, un concepto que ya existía en el DES, se sabe que de su diseño depende enormemente la seguridad del algoritmo). Hoy día se usan más otros métodos que trabajan con bloques mayores como AES y TwoFish (basado en Blowfish). No se conocen ataques efectivos.
- **AES**: el Advanced Encryption Standard es el nuevo estándar criptográfico del NIST (desde el 26 de mayo de 2002 ha sustituido al DES). Se espera que sea el más utilizado en un futuro próximo. El tamaño de bloque es de 128 bits y la clave puede ser de 128, 192 ó 256 bits. De nuevo, se trata de la aplicación repetida de sustituciones, permutaciones y operaciones XOR. Nótese que estas operaciones deben estar muy estudiadas para dar lugar a un algoritmo seguro. Si intentamos crear un algoritmo de este tipo de cualquier manera, seguramente crearemos un cifrador predecible (y por tanto atacable).

FUNCIONES HASH: MD5, SHA

He dejado los algoritmos HASH justo para el final porque realmente no son algoritmos de cifrado sino de "reducción criptográfica". Estos métodos convierten cualquier texto en un número de N bits. Además, funcionan como una función de un único sentido, esto es: no se conoce ninguna forma de recuperar el texto a partir del "texto reducido" de N bits. Por último, y de ahí viene su nombre, actúan como funciones hash, esto es: la probabilidad de que dos textos diferentes produzcan los mismos N bits es despreciable. ¿Y esto para qué puede servir? Pues, por ejemplo, para almacenar textos que no se pueden guardar en llano pero que tampoco hay que descifrar. Se usan, por ejemplo, para guardar passwords. El sistema aplica obligatoriamente la función hash cuando se introduce una clave y compara las reducciones. Aunque un espía lea las reducciones nunca podrá obtener las claves originales. Hay dos algoritmos hash importantes hoy día:

- **MD5**: MD viene de "Message Digest" o resumen del mensaje. El 5, evidentemente, es el número de versión. MD5 convierte cualquier texto en un número de 128 bits. Este método MD5 fue diseñado en 1991 por el profesor del MIT Ronald Rivest (la R del RSA). MD5 es el método utilizado en Linux para almacenar las passwords de usuarios. También se usa para calcular resúmenes a modo de CRC's de archivos completos para asegurar su autenticidad. Por último, comentar que el señor Rivest también es autor de varios métodos de cifrado simétrico llamados RC2, RC4, RC5 y RC6 (RC1 y RC3 resultaron ser inseguros y nunca llegaron a ser publicados).
- **SHA**: Secure Hash Algorithm (SHA) es un conjunto de funciones HASH criptográficas publicadas por el NIST norteamericano y cuya autoría se atribuye a la agencia NSA (National Security Agency). La primera (SHA-0) se publicó en 1993 y después surgieron cinco versiones más. SHA-0 y SHA-1 producen una salida resumen de 160 bits. Las versiones SHA-224, SHA-256, SHA-384, y SHA-512 producen salidas de la longitud que indica su nombre, aunque a todos estos métodos se les llama conjuntamente SHA-2.

ESTEGANOGRAFÍA, WATERMARKING

Quería hablar un poco de esteganografía... que no es cifrado sino que se traduciría como "el arte de escribir oculto". Otra forma de lograr el secreto es meter un mensaje donde nadie lo puede ver... para eso hay muchas técnicas sencillas pero ingeniosas:

- Suponiendo que en inglés "which" y "that" son equivalentes, pongamos que "which" es un 1 y "that" un cero, podemos intercambiar un texto que realmente contiene un mensaje... Necesitamos una fuente de texto casi inagotable para sustituir las palabras clave hasta conseguir meter todos los bits. Las hay: la Biblia, las obras completas de Shakespeare...
- Un método similar al anterior pero válido en todas las lenguas es usar el espaciado entre palabras: dos espacios es un uno y un espacio es un cero.
- El más divertido (aunque igual de simple y de "atacable" que los anteriores) es meter el mensaje en los bits de menor peso de un fichero multimedia (audio o imagen). Imaginad una imagen en escala de grises donde cada punto es un número de 8 bits. Modificar el de menor peso (para que sea un bit del mensaje que queremos transmitir) no debería notarse. Para el que no se lo crea, copio abajo dos funciones de matlab. La primera función introduce el mensaje en la imagen sin que se note mientras que la segunda lo recupera. ¿Cuántos Kbytes caben en una imagen 512x512?

```
function ImOut = MensajeSecreto(im0,mensaje)
%Ojo:no funciona si la imagen no es formato uint8
im00 = im2uint8(im0);    %Pasar a uint8
%Pasar el mensaje a binario (ascii)
msgbin = uint8(mensaje);
%255 = caracter fin de mensaje
msgbin = [msgbin 255];
%Introducir el mensaje en la imagen
%usando los bits de menor peso
cont = 1; %Pixel actual de la imagen
for i=1:length(msgbin)
    aux = msgbin(i); %Valor a introducir
    for j=0:7,
        bit = (bitand(aux,2^j)>0); %Bit j-esimo
        if (bit)
            %Encender el bit de menor peso
            %(si no lo estaba ya)
            im00(cont) = bitor(im00(cont),1);
        else
            %Apagar el bit de menor peso
            %(si no lo estaba ya)
            im00(cont) = bitand(im00(cont),254);
        end
        cont = cont+1; %Pasar al siguiente
    end
end
%Dar salida y acabar.
ImOut = im00;
function mensaje = LeerMensaje(im0)
%Ojo:no funciona si la imagen no es formato uint8
im00 = im2uint8(im0); %Pasar a uint8
msgbin = []; %Mensaje inicialmente en blanco
%Ir leyendo el mensaje
cont = 1; %Pixel actual de la imagen
aux = -1; %Caracter anterior
while (aux~=255)
    if (aux~=-1)
        %Anhadir el caracter anterior al mensaje
        %(si procede)
        msgbin = [msgbin aux];
    end
    aux = 0;
    for j=0:7,
        %Bit de menor peso de este pixel
        bit = (bitand(im00(cont),1)>0);
        if (bit)
            %Encender el bit j-esimo
            aux = bitor(aux,2^j);
        end
        cont = cont+1; %Pasar al siguiente pixel
    end
end
%Pasar a texto y acabar.
mensaje = char(msgbin);
```

Este último ejemplo nos introduce en un tema relacionado con la esteganografía y el cifrado: el watermarking o "marcado al agua". La marca al agua de los billetes es un dibujo que se puede ver al trasluz que sirve para autentificar al billete. ¿Podemos añadir a las imágenes y/o al audio una marca imperceptible al ojo (o al oído) que marque quién produjo la imagen y cuando. Eso serviría para saber que esta o aquella película ha sido pirateada... si la marca indica a quién se le prestó el original alguien puede tener problemas :). Las buenas marcas al agua resisten los ataques... el pirata puede que se dedique a hacer operaciones al fichero para intentar hacer desaparecer la marca. Deberíamos asegurar que mientras la imagen/audio no sean estropeados excesivamente, la marca sigue ahí. ¿Qué puede hacer un pirata? Filtrar paso-bajo sin que se note mucho, pasar a analógico y digitalizar, con las imágenes hay muchas más posibilidades: reescalar, girar 30º grados a la derecha y 30.1º a la izquierda, aplicar distorsiones no lineales... (si usamos sólo el bit de menor peso, el pirata los pondrá todos a cero y se quedará tan ancho). El watermarking robusto es un problema muy complejo y sólo puede resolverse si se considera como un problema de transmisión digital en un canal especial (una imagen o un fichero de audio). Con las técnicas adecuadas se crean mensajes que "se adaptan a la imagen" ya que en algunas partes se puede sumar un mensaje mayor que 1 bit sin que se note y en otras no. ■

No puedo dejar de mencionar que sobre watermarking se han hecho trabajos muy importantes en nuestro centro, por ejemplo: **"Statistical Analysis of Watermarking Schemes for Copyright Protection of Images"**, autores: *Juan Ramón, Hernández y Fernando Pérez-González*. Se publicó en 1999 en los proceedings del **IEEE**.

Ahora tocaría hablar de criptografía cuántica... un método que se supone totalmente inviolable. Lo siento pero... ya me ha salido muy largo y no sé si va a caber en la revista. Lo de arriba es una excusa... ese tema me parece extra-terrestre y no me atrevo a hablar de él.
Si queréis saber cosas sobre eso ya sabéis... en Internet está todo (bueno, también hay libros). Y, de nuevo, gracias a la Wikipedia por la gran cantidad de información que nos regala.

Sockets RAW

Descubre como trabajan las herramientas de los hackers

por Huakin Paquete

Alguna vez os habéis preguntado como funcionan esas herramientas de red que usáis a diario?. No, no hablamos del explorador o el Filezilla, sino de las herramientas del sistema como ping, traceroute o nmap?... ah, que no las usáis a diario?. En ese caso, este artículo probablemente no sea para vosotros.

La mayoría de las herramientas de red a nivel de sistema necesitan poder acceder al hardware de red a muy bajo nivel.

No es necesario llegar a manejar el chip de la tarjeta de red directamente (eso es cosa de los drivers en el kernel), pero prácticamente todas ellas requieren poder construir paquetes especiales y capturarlos tal y como llegan, sin que nadie los haya manipulado.

El kernel Linux proporciona distintas posibilidades para ello, siendo los sockets RAW una de las más utilizadas y veteranas. Los sockets RAW, como su propio nombre indican son sockets (iguales que los otros), cuya única diferencia es que cuando recibimos o enviamos datos a través de ellos, esos datos se proporcionan o se obtienen "en bruto" ("raw" en inglés).

Los socket RAW nos permiten enviar y recibir datos "en bruto" a través de la red

HERRAMIENTAS ICMP

Vamos a comenzar con una de las herramientas más sencillas y más útiles que nos podemos encontrar en el sistema: `ping`. Esta herramienta nos permite saber si una determinada máquina conectada a la red está funcionando o no.

La herramienta `ping` simplemente envía paquetes ICMP `ECHO`, y espera recibir una respuesta ICMP `ECHOREPLY`. Los paquetes ICMP `ECHO` permiten enviar algunos datos extra, y son estos datos los que `ping` utiliza para estimar el tiempo de respuesta de la máquina remota. El valor *time* de la salida del programa.

Para implementar nuestra versión reducida de `ping`, necesitamos realizar dos operaciones. La primera es poder enviar paquetes ICMP, y la segunda, obviamente, es poder recibir paquetes ICMP.

Comenzaremos por la segunda parte, ya que es más sencilla y nos permitirá introducir los nuevos conceptos de una forma progresiva.

CREANDO NUESTRO SOCKET

Como siempre, lo primero que necesitamos son algunos ficheros de cabecera. Aquí tenéis la lista de los que necesitamos.

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

#include <sys/socket.h>
#include <netinet/in.h>

#include <linux/ip.h>
#include <linux/icmp.h>
```

Hemos agrupado los ficheros de cabecera en tres grupos. En el primero encontramos los de siempre, cabeceras estándar para utilizar funciones como `printf` o `exit`. El segundo bloque es el que contiene las definiciones que necesitaremos para crear nuestro socket RAW. Finalmente, el último grupo contiene la definición de las cabeceras de los paquetes para los protocolos IP e ICMP. Enseguida veremos como utilizarlos.

Con estas definiciones ya estamos en condiciones de escribir una función para poder crear nuestro socket RAW.

```
int crea_socket (int proto) {
 int s;

 if ((s = socket (AF_INET, SOCK_RAW, proto))<0)
  {
    perror ("socket:");
    exit (1);
  }
 return s;
}
```

Hemos decidido pasar como parámetro el protocolo, de forma que podamos reutilizar esta función en el resto de los ejemplos de este artículo. Cuando lleguemos a la función main de nuestro programa veremos para que sirve ese parámetro.

Como podéis observar, hemos seleccionado la familia de direcciones de internet (`AF_INET`) e indicado que el socket sea de tipo `SOCK_RAW`. Sin grandes sorpresas hasta aquí.

Hemos mantenido la comprobación de errores para la llamada al sistema `socket` ya que los socket RAW necesitan permisos especiales. Si no ejecutamos nuestro programa como root, o utilizando `sudo`, la llamada al sistema fallará y los resultados del programa serán erróneos.

LEYENDO PAQUETES

Lo siguiente que necesitamos es una estructura de datos para poder acceder cómodamente a los distintos campos de los paquetes que capturemos. Esta es la que nosotros hemos elegido.

```
typedef struct {
  struct iphdr   ip;
  struct icmphdr icmp;
  char   buffer[1024]; /* Datos */
} PKT;
```

Esta estructura representa un paquete de datos ICMP, tal y como lo leeremos desde nuestro socket RAW. Lo primero que encontraremos será la cabecera IP del paquete. A continuación, la cabecera del mensaje ICMP, seguida de un bloque de datos que variará en función del tipo de mensaje ICMP recibido.

En principio, el tamaño de buffer debería reservarse dinámicamente según la información en los campos de tamaño de las cabeceras del paquete. Para mantener los ejemplos de código sencillos vamos a utilizar un buffer de 1Kb que será más que suficiente para nuestras pruebas.

CAPTURANDO PAQUETES ICMP

Con nuestras definiciones previas, estamos en condiciones de escribir nuestro primer capturador de paquetes ICMP. Podéis ver la función `main` a continuación:

```
int main (int argc, char *argv[]) {
int   s;
PKT   pkt;

s = crea_socket  (IPPROTO_ICMP);

while (1)
 {
   read(s, &pkt, sizeof (PKT));
   printf ("Code: %d.%d ",
           pkt.icmp.type, pkt.icmp.code);
   printf ("(%s)\n",
           inet_ntoa(pkt.ip.saddr));
  }
return 0;
}
```

Sencillo no?. Simplemente creamos nuestro socket RAW, indicando que queremos utilizar el protocolo ICMP, leemos datos del socket en nuestra estructura especial, y accedemos a la información del paquete utilizando esta estructura.

Para probar nuestro ejemplo, lanzamos como usuario root (o utilizando sudo, lo que más rabia os de), nuestro capturador de paquetes (que hemos llamado `icmplog`. Desde otro terminal, hacemos un `ping` a cualquier máquina que sepamos que responderá a nuestra llamada (vuestro modem ADSL, vuestro DNS, google o ping.com :).

Esto es lo que obtendremos:

```
occams@razor $ sudo ./icmplog
Code: 0.0 (192.168.100.1)
Code: 0.0 (192.168.100.1)
...
```

Los mensajes de tipo 0 y código 0 son efectivamente mensajes `ECHOREPLY`, que el sistema remoto responde a nuestro ping (el que estamos ejecutando en la otra consola). Obviamente `192.168.100.1` es la máquina contra la que lanzamos el ping.

PÁ ENTRETENERSE

Si miráis la salida del programa ping, veréis que proporciona más datos que nuestra pequeña aplicación. Algunos de los datos que muestra se encuentran en la cabecera ICMP, otros en la sección de datos -tendréis que ver el código fuente de ping- y otros requieren conocer qué paquete se envió en primer lugar. Los que andéis aburridos podéis extender este ejemplo para que se parezca más a ping

DETECTANDO ESCANEOS SIGILOSOS

Antes de continuar con la versión completa de nuestro sencillo `ping` vamos a hacer un corto inciso, para, convertir nuestro pequeño programa en un detector de escaneos sigilosos como los que realiza `nmap`.

Capturar paquetes IP con un socket RAW es muy sencillo

Partiendo de nuestro ejemplo anterior, vamos a realizar unas sencillas modificaciones para transformar nuestro programa en un rudimentario detector de escaneos de tipo `FIN`. Estos escaneos se caracterizan por que el flag `FIN` de la cabecera TCP está puesto a 1.

Nosotros vamos a informar de todos los paquetes de este tipo que recibamos, si bien puede que algunos sean paquete legítimos.

Lo primero que tenemos que hacer es añadir un nuevo fichero de cabecera a nuestro programa

```
#include <linux/tcp.h>
```

También tenemos que hacer una pequeña modificación a nuestra estructura de datos. Ahora vamos a capturar paquete TCP y, en este caso, lo que nos encontramos a continuación de la cabecera IP, es una cabecera TCP, no una cabecera ICMP. Teniendo esto en cuenta, nuestra estructura de datos sería algo como esto:

```
typedef struct {
  struct iphdr   ip;
  struct tcphdr  tcp;
  char   buffer[1024]; /* Datos */
} PKT_TCP;
```

Con estos cambios, nuestra función `main` para detectar posibles escaneos FIN quedaría de la siguiente forma:

```
int main (int argc, char *argv[]) {
int     s;
PKT_TCP pkt;

s = crea_socket  (IPPROTO_TCP);

while (1)
 {
  read(s, &pkt, sizeof (PKT_TCP));
  if (pkt.tcp.fin)
    printf ("Posible FIN scan al puerto %d "
            "desde (%s)\n",
            ntohs (pkt.tcp.dest),
            inet_ntoa(pkt.ip.saddr));
  }
return 0;
}
```

Como podéis observar, cuando creamos nuestro nuevo socket RAW estamos indicando que queremos capturar paquetes TCP, en lugar de ICMP como hicimos en nuestro ejemplo anterior.

Una vez hecho esto, nuestra nueva estructura de datos nos permite acceder directamente a la información en la cabecera TCP. Por una parte comprobamos el valor del flag `FIN`, y por otra obtenemos la información relativa al puerto al que va dirigido el paquete.

Podemos detectar un escaneo FIN en unas pocas líneas de código

Observad que tenemos que utilizar la función `ntohs` para convertir del formato de red al formato de máquina (`ntohs` significa *Network TO Host Short*). Sí, los datos de las cabeceras se transforman a un formato especial para ser transmitidos por la red, de forma que cualquier máquina pueda acceder a ellos independientemente de su arquitectura (*little* o *big endian*).

COMPROBANDO EL DETECTOR

Vamos a comprobar que tal funciona nuestro nuevo detector de escaneos `FIN`. Para ello, iniciamos nuestra aplicación en una consola, y desde otro terminal ejecutamos el siguiente comando:

```
# nmap -sF localhost -p 2000
```

Nuestro flamante detector nos informará diligentemente del intento de ataque:

```
occams@razor:~$ sudo ./finscan
[sudo] password for occams:
Posible FIN scan al puerto 2000 desde (127.0.0.1)
```

Pero si recordáis, os comentamos que el hecho de recibir un paquete `FIN` no significa que se esté llevando a cabo un escaneo de puertos. Para comprobar como se generan estos paquetes vamos a realizar un pequeño experimento.

Abrid dos consolas y en una de ellas ejecutad *Netcat* en modo servidor escuchando en el puerto 2000. En la otra ejecutad *Netcat* en modo cliente conectándose a `localhost` y al puerto 2000.

```
CONSOLA Servidor: nc -l -p 2000
CONSOLA Cliente : nc localhost 2000
```

Ahora, pulsad CTRL+C en la parte cliente y comprobad que obtenéis en el detector de escaneos. Repetid el proceso, pero ahora pulsando CTRL+C en la consola donde se ejecuta el servidor.

Lo que nuestro detector mostrará será algo como esto:

```
Posible FIN scan al puerto 60784 desde (127.0.0.1)
Posible FIN scan al puerto 2000 desde (127.0.0.1)
```

Efectivamente, el flag `FIN` se utiliza para cerrar las conexiones, de forma que, para detectar si se trata de un escaneo real, tendríamos que saber si el paquete `FIN` que hemos capturado no se corresponde con el cierre legítimo de una conexión activa. Es decir, necesitamos saber que conexiones están abiertas... o quizás podamos averiguarlo a partir de los otros campos de las cabeceras IP y TCP?... algo más para entretenerse :).

ENVIANDO PAQUETES ICMP

Después de este "no tan corto paréntesis", volvamos a nuestra pequeña utilidad `ping`.

Como os decíamos en una sección anterior, para poder implementar algo parecido a `ping`, necesitamos poder enviar paquetes ICMP. No existe una manera directa de hacer esto, de forma similar a como se transmiten datos utilizando TCP o UDP. Necesitamos un socket RAW para poder enviar estos paquetes.

Para este ejemplo necesitamos los mismos ficheros de cabecera que para el primer ejemplo que introdujimos (`icmplog`). También reutilizaremos nuestra función `crea_socket`, pero para poder transmitir nuestro paquete ICMP necesitaremos algo más de código.

Lo primero que necesitamos es una estructura de datos adicional. Si bien, podríamos utilizar nuestra estructura `PKT`, ya que un paquete es un paquete, por cuestiones prácticas nos interesa definir una estructura como esta:

```
typedef struct {
  struct icmphdr icmp;
  char   buffer[1024]; /* Data */
} PKT_TX;
```

Como podéis observar, hemos eliminado la cabecera IP de nuestra estructura de transmisión. La razón: Vamos a dejar que la pila TCP/IP de nuestro sistema rellene los campos de la cabecera IP. Más adelante veremos como rellenarlos nosotros mismos, pero para implementar nuestro `ping`, es mucho más conveniente dejar al sistema hacerlo.

Además de esta estructura de datos, vamos a añadir una función para construir paquetes ICMP `ECHO`, con un determinado bloque de datos. Podéis ver esta función a continuación.

```
int
hello_pkt2 (char *pkt_icmp, char *str)
{
 struct icmphdr *icmp =(struct icmphdr*)pkt_icmp;
 PKT_TX         *pkt = (PKT_TX*) pkt_icmp;
 int             len;

 len = sprintf (pkt->buffer, "%s", str);
 len += sizeof (struct icmphdr);

 /* Construye cabecera ICMP */
 icmp->type = ICMP_ECHO;
 icmp->code = 0;
 icmp->un.echo.id = htons(123);
 icmp->un.echo.sequence = htons(5);
 icmp->checksum = 0;
 icmp->checksum = cksum ((char*)icmp, len);
 return len;
}
```

Esta función recibe como parámetro, un puntero a nuestra estructura de paquete en memoria, y una cadena de caracteres que se enviará en el bloque de datos.

Construir un paquete ICMP ECHO es tan sencillo como rellenar una estructura de datos

Con esta información, la función construye un paquete ICMP `ECHO` con identificador 123 y número de secuencia 5. Estos valores serán los que utilicemos para saber si el paquete que capturamos es el que nos interesa.

CALCULANDO EL CHECKSUM

El último elemento que necesitamos para construir un paquete ICMP válido es el cálculo de la suma de comprobación de los datos. El protocolo ICMP especifica que este valor se debe calcular utilizando todo el paquete (cabecera + datos), siendo el valor del campo `checksum` de la cabecera igual a 0, para este cálculo.

Nosotros hemos tomado prestada la rutina de *checksum* de la implementación de `ping` que podéis encontrar en cualquier distribución linux. Esta función tiene la siguiente pinta.

```
/* Calculo de checksum */
u_short
cksum (u_char *addr, int len)
{
 register int sum = 0;
 u_short answer = 0;
 u_short *wp;

 for (wp =(u_short*)addr; len>1; wp++,len-=2)
   sum += *wp;

 /* Take in an odd byte if present */
 if (len == 1)
   {
     *(u_char *)&answer = *(u_char*)wp;
     sum += answer;
   }

 /* add high 16 to low 16 */
 sum = (sum >> 16) + (sum & 0xffff);
 /* add carry */
 sum += (sum >> 16);
 /* truncate to 16 bits */
 answer = ~sum;
 return answer;
}
```

Básicamente lo que hace la función es sumar todos los bytes que componen el paquete (de ahí lo de suma de comprobación), y finalmente reducirlo a un valor de 16 bits que es el espacio disponible en la cabecera.

Con estas funciones de soporte ya podemos escribir nuestra versión super-simplificada de ping.

EL PING MÁS TONTO DEL MUNDO

Nuestro programa principal es ahora un poco más complicado... pero no mucho más.

```
int
main (int argc, char *argv[])
{
  int     s, data_size;
  PKT     pkt;
  PKT_TX  echo;
  char    tmp[1024];
  struct sockaddr_in dest;

  s = crea_socket  (IPPROTO_ICMP);

  /* Construimos paquete */
  dest.sin_family = AF_INET;
  inet_aton (argv[1], &dest.sin_addr);
  data_size = hello_pkt2 ((char*) &echo,
                          "Hello World!!");
  /* Enviamos el paquete */
  sendto (s, &echo, data_size, 0,
          (struct sockaddr*) &dest,
          sizeof (struct sockaddr_in));

  /* Empezamos la captura */
  while (1)
    {
      read(s, &pkt, sizeof (PKT));
      printf ("Code: %d.%d (%d)",
              pkt.icmp.type, pkt.icmp.code,
              ntohs(pkt.icmp.un.echo.id));
      printf ("(%s)", inet_ntoa(pkt.ip.saddr));
      if ((pkt.icmp.type == ICMP_ECHOREPLY) &&
          (ntohs(pkt.icmp.un.echo.id) == 123))
        {
          printf ("\nEste es el nuestro :)!!\n");
          break;
        }
      printf ("\n");
    }
  close (s);

  return 0;
}
```

Al igual que en nuestro anterior ejemplo, comenzamos creando nuestro socket RAW, pero en este caso, antes de ponernos a capturar paquetes, tenemos que enviar nuestro paquete `ECHO`.

Para ello utilizamos la llamada al sistema `sendto`, que nos permite especificar la dirección de destino del paquete. Recordad que, si no se especifica lo contrario, la pila TCP/IP de nuestro ordenador rellenará las cabeceras IP. Sin embargo, lo que la pila es capaz de rellenar tienen un límite. Hasta que los ordenadores sean capaces de leer nuestras mentes, pues tendremos que decirle a donde queremos enviar el paquete :).

La llamada al sistema `sendto` espera esta información en su tercer parámetro como un puntero a una variable de tipo `struct sockaddr`.

Así que lo primero que hace el programa es rellenar una estructura de ese tipo. A continuación creamos el paquete `ECHO` con la función que ya teníamos lista, y lo enviamos con la anteriormente mencionada función `sendto`

Lo que sigue ya os tendría que resultar familiar. Es el mismo bucle que escribimos para nuestro capturador de paquetes ICMP, pero en este caso, mostramos un mensaje especial cuando recibimos una respuesta `ECHO_REPLY` al paquete que acabamos de enviar. Además comprobamos que el paquete contiene el identificador de paquete que habíamos introducido.

PROBANDO NUESTRO PING

Ha llegado el momento de probar nuestro `ping`. Para ello, como en nuestros ejemplos anteriores, escogemos una máquina que responda a los *pings* y ejecutamos nuestro programa.

```
occams@razor$ sudo ./pingc1 192.168.100.1
[sudo] password for occams:
Code: 0.0 (123)(192.168.100.1)
Este es el nuestro :)!!
```

Si enviamos nuestros paquetes contra una máquina que no existe (o no responde a los paquetes ECHO), nuestro programa quedará esperando indefinidamente. Sí, obviamente tenemos que utilizar un temporizador para determinar si un paquete se ha perdido... eso os lo dejamos como ejercicio :).

Pero, ¿qué ha pasado con nuestro "Hola Mundo!"?... Ejecutemos de nuevo nuestro programa, pero esta vez vamos a lanzar `tcpdump` en un terminal. Esto es lo que obtendríamos:

```
occams@razor$ sudo tcpdump -A 'icmp'
(...)
15:28:12.219162 IP occams.local > target: ICMP echo request,
                id 123, seq 5, length 21
                E..)..@.@.....o.........{..Hello World!!
15:28:12.219604 IP target > occmas.local: ICMP echo reply,
                id 123, seq 5, length 21
                E..)....@.........o.....{..Hello World!!.3hV.
(...)
```

Sí, ahí está. Ha ido hasta la máquina destino y a vuelto. Así que podemos utilizar paquetes ICMP también para transmitir datos. Un ejemplo de este uso es el programa ICMP shell, que permite acceso *shell* remoto a una máquina a través de ICMP. En concreto esta aplicación permite utilizar otros mensajes ICMP, no solo los mensajes `ECHO`, para este intercambio de datos.

EL PING DE LA MUERTE

El sistema operativo Windows 95, tenía un bug en su pila TCP/IP que hacía que el sistema se estrellase si recibía un paquete ICMP ECHO con un bloque de datos de más de 64Kb. Este paquete, así como la aplicación que lo mandaba -que ahora ya sabéis hacer- se conocía como el Ping de la Muerte (*PoD Ping of Death*)

Más información...

ESCANEO DE MEDIA CONEXIÓN

Ahora que ya tenemos dominado ICMP, vamos con TCP. Para ilustrar este ejemplo vamos a implementar el clásico escaneo de media conexión, en el que el tradicional *"saludo a tres bandas"* de TCP se queda en sólo dos.

Una conexión TCP, se inicia con el envío de un paquete `SYN` (en seguida veremos que es esto). El sistema remoto responde con un paquete `SYN ACK`, o paquete de asentimiento de conexión. Para que la conexión se considere establecida el sistema que inicia la comunicación debe enviar un paquete `ACK` de vuelta.

Básicamente esto es lo que hace la llamada al sistema `connect`... nosotros vamos a implementar nuestro propio `connect`.

En este ir y venir de paquetes, ambos sistemas intercambian números de serie que serán utilizados durante la comunicación. Estos números de serie son los que hacen que el spoofing de una conexión TCP no sea práctico, al menos lo que se conoce como *blind spoofing*.

Los socket RAW nos permiten control absoluto sobre los flags TCP

En un escaneo de media conexión no se envía el último paquete `ACK`, con lo cual la conexión no se puede considerar completa y, dependiendo del sistema de detección de intrusos instalado, puede que el escaneo no se detecte. Esto no es muy probable que pase hoy en día, pero desde un punto de vista didáctico nos va a permitir introducir un montón de conceptos.

MODIFICANDO NUESTRO EJEMPLO

Para poder implementar nuestro escáner de media conexión tendremos que hacer algunas modificaciones a nuestro programa. Lo primero que necesitamos es un nuevo fichero de cabecera, y una mínima modificación de nuestras estructuras de datos.

```
#include <linux/tcp.h>

typedef struct {
  struct tcphdr  tcp;
} PKT;

typedef struct {
  struct iphdr   ip;
  struct tcphdr  tcp;
} PKT_RX;
```

Como podéis comprobar, simplemente hemos substituido la cabecera ICMP de nuestro ejemplo anterior por la nueva cabecera TCP (para eso necesitamos el fichero .h adicional). También hemos eliminado el bloque de datos que no vamos a necesitar para nuestro escáner. Volveremos sobre estas estructuras un poco más tarde.

La siguiente parte que tenemos que modificar es el programa principal, que quedaría de esta guisa.

```
int
main (int argc, char *argv[]) {
 int     s, data_size;
 PKT_RX pkt;
 PKT     syn;
 char    tmp[1024];
 struct sockaddr_in dest;

 s = crea_socket   (IPPROTO_TCP);

 /* Construimos paquete */
 dest.sin_family = AF_INET;
 inet_aton (argv[1], &dest.sin_addr);
 dest.sin_port = htons(atoi(argv[2]));

 memset (&syn, 0, sizeof(PKT));
 data_size = tcp_pkt (&syn, argv[1],atoi(argv[2]),
                      argv[3], atoi(argv[4]));

 /* Enviamos el paquete */
 sendto (s, &syn, data_size, 0,
          (struct sockaddr*) &dest,
          sizeof (struct sockaddr_in));

 /* Empezamos la captura */
 while (1)
 {
   read(s, &pkt, sizeof (PKT_RX));
   printf ("Paquete desde %s: %d ACK: %d SYN: %d\n",
    inet_ntoa(*((struct in_addr*)&pkt.ip.saddr)),
    ntohs(pkt.tcp.source),
    pkt.tcp.ack, pkt.tcp.syn);

   if (!memcmp (&pkt.ip.saddr,
                &dest.sin_addr,
                sizeof (struct in_addr))) break;
   }
  close (s);

  return 0;
}
```

El primer cambio que observamos es que ahora nuestro socket RAW va a utilizar un protocolo diferente. En lugar del anterior `IPPROTO_ICMP`, ahora nos encontramos un interesante `IPPRPTO_TCP`, que enviamos como parámetro a nuestra función de creación del socket.

El segundo cambio importante que nos encontramos es que, ahora, nuestro programa va a recibir cuatro parámetros. Los dos primeros identificarán la dirección ip y puerto de la máquina remota, y los dos segundos identificarán la dirección ip y puerto de la máquina local... Esto último debería pareceros interesante ;).

En tercer lugar, obviamente, tenemos que utilizar una función distinta para crear nuestro paquete. En esta ocasión le hemos dado el original nombre de `tcp_pkt`.

No hay grandes diferencias entre la construcción de un paquete ICMP y de un paquete TCP.

Para terminar, hemos mejorado un poco nuestro bucle principal. Ahora mostramos información específica del protocolo TCP (como por ejemplo los valores de los flags `SYN` y `ACK`) y hemos añadido un test sencillo para terminar la aplicación.

Como enseguida comprobaréis, el test para terminar la aplicación no es completo. El test completo os lo dejamos a vosotros para que os entretengáis y así el listado nos queda más corto y centrado en el tema que nos ocupa.

CONSTRUYENDO UN PAQUETE TCP

Para que nuestro escáner esté completo, nos falta la función que construye el paquete TCP. Como dijimos, necesitamos enviar un paquete `SYN`, es decir, un paquete con el flag `SYN` activado en la cabecera TCP. Tranquilos, esto es muchísimo más fácil de lo que parece. Aquí tenéis la función `tcp_pkt`.

```
int
tcp_pkt (PKT *p, char *dir_dest, int puertod,
         char *dir_orig, int puertoo)
{
 p->tcp.source = htons (puertoo);
 p->tcp.dest = htons (puertod);
 p->tcp.doff = sizeof (struct tcphdr)  / 4;

 p->tcp.syn = 1;

 p->tcp.check = cksum ((unsigned char*)
                        &p->tcp,
                        sizeof(struct tcphdr));
 return (sizeof(struct tcphdr));
}
```

La función simplemente rellena la cabecera TCP con la mínima información necesaria para poder enviar el paquete. El puerto origen y destino, el flag `SYN` (ya os dije que era fácil), el *checksum*, que calculamos con la misma función que utilizamos para nuestro `ping`, y el campo `doff` que indica el desplazamiento dentro del paquete desde la cabecera tcp a la zona de datos en palabras de 32 bits (de ahí lo de dividir por 4).

Dicho de otra forma, contiene el número de palabras de 32bits que ocupa la cabecera TCP. Sí, ya, pero como el nombre del campo es doff (*Data Offset*)... Podéis probar que sucede si no inicializáis correctamente este campo.

PROBANDO NUESTRO FLAMANTE ESCÁNER

Compilamos nuestro programa que hemos llamado `syn-scan-bogus` y lo ejecutamos.

```
occams@razor$ sudo ./synscan-bogus 192.168.100.1 443 \
> 192.168.100.100 5000
```

Nada... Mi gozo en un pozo.

Veamos que nos dice `tcpdump`

```
occams@razor:~$ sudo tcpdump -n
tcpdump: verbose output suppressed, use -v or -vv for full
protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
20:04:07.617606 IP 192.168.100.100.5000 > 192.168.1.100.443:
                S 0:0(0) win 0
```

Bueno, el paquete sale, con las direcciones IP y puertos correctos y el flag `SYN` activo. Seguiremos el consejo de `tcpdump`, y activamos el modo *verbose*.

```
occams@razor:~$ sudo tcpdump -nv
20:07:08.439772 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF],
                proto TCP (6), length 40) 192.168.100.100.5000 >
                192.168.1.1.443: S, cksum 0x9ac4
                (incorrect (-> 0x16e9), 0:0(0) win 0
```

Ajá, nuestro *checksum* es incorrecto y el sistema remoto simplemente tira el paquete, por eso no recibimos ninguna respuesta.

Ni cortos ni perezosos comprobamos el RFC 793, el que define TCP. Una búsqueda rápida de la cadena *checksum* en el documento nos lleva a una detallada sección que describe como calcularlo.

CALCULANDO EL CHECKSUM TCP

Leyendo detenidamente el RFC, podemos comprobar que nuestra función para calcular la suma de comprobación del paquete es correcta, pero tenemos que añadir lo que el RFC llama una pseudo-cabecera. Vale, modifiquemos nuestro escáner.

Lo primero que tenemos que hacer es modificar nuestras estructuras de datos. Añadimos una estructura para trabajar cómodamente con esta pseudo-cabecera, y modificamos `PKT` para que la incluya. Notad que no estamos enviando datos en nuestro paquete, así que simplemente podemos añadir la pseudo-cabecera a continuación del la cabecera TCP. Las estructuras quedarían de esta forma.

```
typedef  struct pseudo_header {
  unsigned long src;
  unsigned long dst;
  unsigned char zero;
  unsigned char proto;
  unsigned short length;
} TCP_PHDR;

typedef struct {
  struct tcphdr  tcp;
  TCP_PHDR       tcp_phdr;
} PKT;
```

Ahora tenemos que actualizar nuestra función `tcp_pkt` para que tenga en cuenta la pseudo-cabecera a la hora de calcular la suma de comprobación. Así es como quedaría.

```
int
tcp_pkt (PKT *p, char *dir_dest, int puertod,
         char *dir_orig, int puertoo) {
 p->tcp.source = htons (puertoo);
 p->tcp.dest = htons (puertod);
 p->tcp.doff = sizeof (struct tcphdr)  / 4;
 p->tcp.syn = 1;

 p->tcp_phdr.src = inet_addr (dir_orig);
 p->tcp_phdr.dst = inet_addr (dir_dest);
 p->tcp_phdr.zero = 0;
 p->tcp_phdr.proto = 6;
 p->tcp_phdr.length=htons(sizeof(struct tcphdr));


 p->tcp.check = cksum ((unsigned char*)&p->tcp,
                       sizeof(struct tcphdr) +
                       sizeof (TCP_PHDR));

 return (sizeof(struct tcphdr));
}
```

Observad que la pseudo-cabecera solo se utiliza para calcular la suma de comprobación y no se enviará junto con el paquete, de ahí que el valor que retorna la función es simplemente el tamaño de la cabecera TCP, como en nuestro primer ejemplo.

SEGUNDO INTENTO

Con esta última modificación del programa, repetimos nuestra prueba.

```
occams@razor$ sudo ./synscan-tcp 192.168.100.1 443 \
> 192.168.100.100 5000
[sudo] password for occams:
Paquete desde 192.168.100.1:443 ACK:1 SYN:1

occams@razor$
```

Mucho mejor esta vez. Bueno, ya lo sabíamos por eso le cambiamos el nombre al programa :). Simplemente para comprobar veamos que dice esta vez `tcpdump`

```
occams@razor:~$ sudo tcpdump -nv
20:40:36.509596 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF],
                proto TCP (6), length 40) 192.168.100.100.5000 >
                192.168.100.1.443: S, cksum 0x16df (correct),
                0:0(0) win 0
20:40:36.510083 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF],
                proto TCP (6), length 44) 192.168.100.1.443 >
                192.168.100.100.5000: S, cksum 0xb8ea (correct),
                2921431349:2921431349(0) ack 1 win 5840 <mss 1460>
```

Ahora nuestros *checksums* son correctos, y recibimos la respuesta esperada desde la máquina remota, un paquete con los flags `SYN` y `ACK` activados.

CONTROL ABSOLUTO

Si observamos con detenimiento la información que nos proporciona `tcpdump`, veremos que hay un montón de datos en los paquetes que enviamos que a priori no podemos modificar. Datos como las direcciones IP o el TTL, se encuentran en la cabecera IP, la cual, hasta el momento, hemos dejado que la manejara el sistema operativo.

Pero eso se acabó. Vamos a modificar nuestro escáner para poder controlar absolutamente toda la información que nos muestra `tcpdump`. Para ello, tenemos que hacer algunas modificaciones adicionales en nuestro pequeño programa.

Utilizando la syscall setsockopts podremos acceder a la cabecera IP

El primer cambio es que ya no necesitamos dos estructuras de datos para enviar y para recibir los paquetes. Ahora vamos a manejar la cabecera IP tanto en transmisión como en recepción, así que solo necesitaremos la siguiente definición.

```
typedef struct {
  struct iphdr   ip;
  struct tcphdr  tcp;
  TCP_PHDR       tcp_phdr;
} PKT;
```

Esta nueva estructura substituye a las anteriores `PKT` y `PKT_RX`. Mantenemos el nombre de la primera, porque mola y para tener que hacer menos cambios al programa.

Además, necesitamos decirle al kernel que no maneje las cabeceras por nosotros. Esto lo conseguimos activando la opción `IP_HDRINCL` (*IP Header Included*... obviamente) de nuestro socket RAW. Esto lo hacemos en una nueva función que os mostramos a continuación.

```
int
cfg_socket (int s)
{
  int   val = 1;

  if (setsockopt (s, IPPROTO_IP, IP_HDRINCL,
                  &val, sizeof (val)) < 0)
    perror ("setsockopt (IPHDRINCL):");

  return s;
}
```

Como hicimos en el ejemplo anterior, antes de meternos en los detalles de la generación de los paquetes, vamos a ver como quedaría nuestra función main.

Solamente vamos a reproducir la primera parte de la función, en la que se envía el paquete, ya que el resto no necesita ningún cambio. De hecho, en esta primera parte solo vamos a modificar dos líneas.

```
int main (int argc, char *argv[]) {
  int     s, data_size;
  PKT     pkt;
  PKT     syn;
  char    tmp[1024];
  struct sockaddr_in dest;

  s = crea_socket   (IPPROTO_RAW);
  cfg_socket (s);

  /* Construimos paquete */
  dest.sin_family = AF_INET;
  inet_aton (argv[1], &dest.sin_addr);
  dest.sin_port = htons(atoi(argv[2]));

  memset (&syn, 0, sizeof(PKT));
  ip_pkt (&syn, argv[1],  argv[3]);
  tcp_pkt (&syn, argv[1], atoi(argv[2]),
           argv[3], atoi(argv[4]));
...
```

Lo primero que podemos ver es que ahora estamos utilizando como protocolo `IPPROTO_RAW` y a continuación estamos llamando a nuestra nueva función para decirle al kernel que nosotros proporcionamos las cabeceras IP (`cfg_socket`).

En este caso concreto, podríamos haber mantenido el protocolo `IPPROTO_TCP`, pero `IPPROTO_RAW` es formalmente más correcto.

El segundo cambio que observamos es la llamada a la función `ip_pkt`. Sí, efectivamente, esa es la función que rellena la cabecera IP. Como en la capa IP no existen los puertos, no es necesario que se los enviemos.

Sin grandes sorpresas hasta aquí. Le decimos al kernel que nosotros manejamos las cabeceras IP, y añadimos una función para manejarlas.

GENERANDO CABECERAS IP

Como acabamos de contaros, la función `ip_pkt` es la encargada de rellenar la cabecera IP de nuestro paquete. Nosotros hemos incluido en el código valores constantes para la mayoría de los campos para mantener el programa pequeño.

Vosotros podéis modificar la función o añadir funciones adicionales para configurar esos valores. Aquí está la función.

```
int
ip_pkt (PKT *p, char *dir_dest, int puertod,
        char *dir_orig, int puertoo)
{
  p->ip.ihl = 5;
  p->ip.version = 4;
  p->ip.tos = 0;
  p->ip.tot_len = sizeof (struct iphdr) +
                  sizeof (struct tcphdr);
  p->ip.id = htons (1234);
  p->ip.frag_off = 0;
  p->ip.ttl = 255;
  p->ip.protocol = 6;
  p->ip.check = 0;
  p->ip.saddr = inet_addr (dir_orig);
  p->ip.daddr = inet_addr (dir_dest);

}
```

Más fácil no se puede. No vamos a explicar en detalle cada uno de los campos de la cabecera, para ellos os remitimos al RFC 791. Pero un par de ellos si merecen algún comentario.

El campo TTL *Time To Live* indica el número de saltos máximo que el paquete puede dar a través de internet, antes de que algún router lo descarte. Este campo es interesante por dos cosas. A saber.

La primera es que la utilidad `traceroute` utiliza este campo y la captura de mensajes ICMP para determinar la ruta de un paquete. La utilidad va incrementando el campo de uno en uno, de forma que cada vez el paquete es descartado por un nodo más lejano en la ruta. Cuando esto sucede, el nodo que lo descarta envía un mensaje ICMP para comunicar la situación. Estos mensajes son procesados por `traceroute` para mostrarnos la ruta final del paquete.

La segunda es que el campo TTL es uno de los utilizados por los sistemas de identificación pasiva de sistema operativo, de tal forma que modificándolo el IDS se puede confundir un poco.

PASSIVE FINGERPRINTING

Los sistemas pasivos de extracción de huellas dactilares (sí, en inglés es mucho más corto), analizan los paquetes que una determinada máquina recibe e intentan determinar el tipo de sistema que lo envió. Para ellos utilizan campos como TTL, que por ejemplo, en los sistemas Linux suele tener un valor de 64, y en los sistemas Windows 255, aunque eso depende de la versión. Es muy fácil modificar este comportamiento, por lo que los sistemas pasivos no son muy fiables. Quizás el más conocido de estos sistemas es p0f.

El sistema de detección remota que proporciona nmap, por el contrario, es un sistema activo, puesto que envía paquetes especiales al sistema remoto con el fin de extraer información adicional sobre él. Por supuesto, estos sistemas son mucho más escandalosos..

IP SPOOFING Y OTRAS MALDADES

Los otros dos campos que nos interesan son, por supuesto, las direcciones IP de origen y destino. Y sí, finalmente ahí estamos, esta es la forma de hacer *IP spoofing*. Podemos poner cualquier valor en el campo de la dirección origen y eso es lo que saldrá por el cable.

Con el programa que hemos escrito, solo tenéis que cambiar el tercer parámetro para enviar un paquete `SYN` con una dirección falsa. El problema de hacer esto es que la respuesta del sistema remoto se enviará a esa dirección y por lo tanto no la podréis ver.

Aplicar técnicas de Blind IP Spoofing hoy en día es prácticamente imposible

Hay dos formas de sacar partido a esto del *spoofing*. La primera, es la que proporciona `nmap` con el flag -D (del inglés *Decoy*, señuelo). Esta opción le indica a `nmap` que envíe varios paquetes al sistema remoto que desea escanear. Todos ellos excepto uno tienen direcciones falsas, de forma que el sistema remoto tiene que comprobar todas las direcciones de los paquetes que recibe y, en general, va a ser bastante complicado determinar desde cual se está produciendo el ataque.

La otra forma de sacarle partido al spoofing es en el secuestro de conexiones, que no deja de ser un caso particular de un ataque *MIM* (*Man in the Middle*. Hoy en día es muy difícil hacer esto, a no ser que se den una serie de condiciones, tan específicas, que la técnica no resulta práctica. Estas condiciones son:

- El atacante tiene que poder capturar el tráfico de la comunicación que desea interceptar. Eso reduce la técnica a las redes de área local, o al control de un router que no es algo sencillo.
- El atacante tiene que poder bloquear la conexión de red de la máquina que pretende suplantar, ya sea con un ataque DoS o utilizando un exploit.

En estas circunstancias, es posible el secuestro de una conexión activa. El resumen es que es necesario conocer los números de serie que van en cada paquete TCP y esto solo se puede hacer de dos formas. O viéndolos directamente o adivinándolos, y esta última opción hace muchos años que dejo de ser viable.

Sí, hace años, los generadores de números pseudoaleatorios utilizados para generar esos números de serie no eran muy buenos (probablemente por cuestiones de eficiencia), y había formas de llegar a predecir esos números de una manera relativamente sencilla.

Sabemos que no hemos hablado sobre estos números de serie, pero en los RFCs que hemos comentado a lo largo del texto podréis encontrar una descripción detallada de como se utilizan.

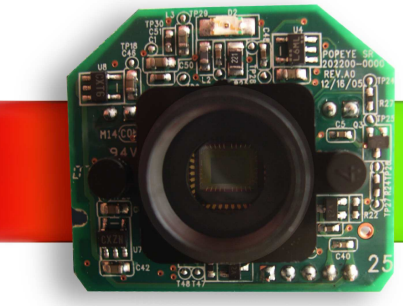
Finalmente comentar que el IP spoofing se suele utilizar en ataques DoS no solo para ocultar el origen del ataque, sino para, como en el caso de *SMURF*, producir efectos de *flooding*... Pero eso ya es tema para otro artículo.

A PARTIR DE AQUÍ...

En este artículo hemos explorado los fundamentos de los socket RAW y como utilizarlos para capturar tráfico o generar paquetes especiales. Con lo que os hemos contado tenéis las herramientas para construir la mayoría de las utilidades del sistema relacionadas con la red y jugar con la pila TCP/IP "casi" al más bajo nivel.

¿Qué podéis hacer ahora?... Bueno, echadle un ojo al fichero `/etc/protocols`, donde encontraréis una lista completa de protocolos con los que podéis jugar. También deberíais comprobar la página del manual `man 7 raw` y si queréis ir más allá `man 7 packet`...

Como siempre, esperamos saber de vuestros experimentos!! ■



Tu propia webcam IR

Explorando el Infrarrojo cercano

por Chinao

Buscando cosas güays por internet, nos encontramos una interesante página en la que cuentan como "tunear" tu webcam para convertirla en una cámara de infrarrojos (los recursos al final del artículo). Nos pareció interesante, así que ni cortos ni perezosos nos pusimos a ello.

Antes de que comencéis a, quizás, destruir vuestra webcam, tenemos que decir dos cosas. La primera es que no nos hacemos responsables de cualquier daño ocasionado por seguir los pasos que se describen en este artículo. Es decir, si os cargáis la cámara (la verdad que es difícil, pero podría llegar a pasar) es vuestro problema.

La segunda es que antes de destrozarla, comprobéis que realmente va a servir de algo. Para ello no tenéis más que conectar la cámara al ordenador, lanzar vuestro programa preferido para utilizarla, y coger cualquier mando a distancia que tengáis por casa y que funcione con infrarrojos.

Estos mandos suelen tener lo que parece un LED (y que efectivamente lo es :), en el extremo con el que intentáis intimidar a vuestro equipo electrónico doméstico. Normalmente, cuando pulsáis cualquier botón, el LED parece no hacer nada, pero si apuntáis a vuestra webcam y al pulsar el botón veis un punto brillante donde debería estar el LED... entonces sí, es hora de "cacharrear".

RADIACIÓN ELECTROMAGNÉTICA

La mayoría de los sensores utilizados por las cámaras digitales (CCD o CMOS), por sus propias características, son sensibles a lo que se conoce como infrarrojo cercano.

¿Y que es eso del infrarrojo?... mejor aún, ¿qué es eso del infrarrojo cercano?. Vale, pues, si hablamos con propiedad, de lo que tenemos que hablar es de radiación infrarroja. La radiación infrarroja, es una radiación electromagnética, cuya longitud de onda se encuentra entre 750nm y 1mm.

Sí, ya, o no os he contado nada nuevo, o os habéis quedado como estábais. Bien, vamos a explicarlo un poco más. La radiación electromagnética que todos conocemos es la luz visible. Lo bueno de ésta es que la podemos ver, como su propio nombre indica :). Esta radiación se propaga, es decir, se mueve y llega hasta nuestros ojos como una onda (una onda electromagnética concretamente). Dependiendo de la frecuencia de esa onda (lo rápido o lento) que varíe, veremos un color u otro.

Así, las ondas que tienen menor longitud de onda, es decir, que necesitan menos "espacio" para pasar de una cresta a la siguiente, son las que tienen mayor frecuencia (cambian más rápido, o en menos "espacio" si lo preferís).

Cuanto menor sea lo longitud de onda, más "azulado" será el color que percibimos, hasta que llega un punto, en el que nuestro ojo ya no es capaz de percibirlo. Los fotones de esa frecuencia que llegan a nuestra retina no pueden excitar las células que en ella se encuentra y por lo tanto ninguna señal llega al cerebro... estamos ciegos para esa longitud de onda!.

La radiación infrarroja tiene una longitud de onda se encuentra entre 750nm y 1mm.

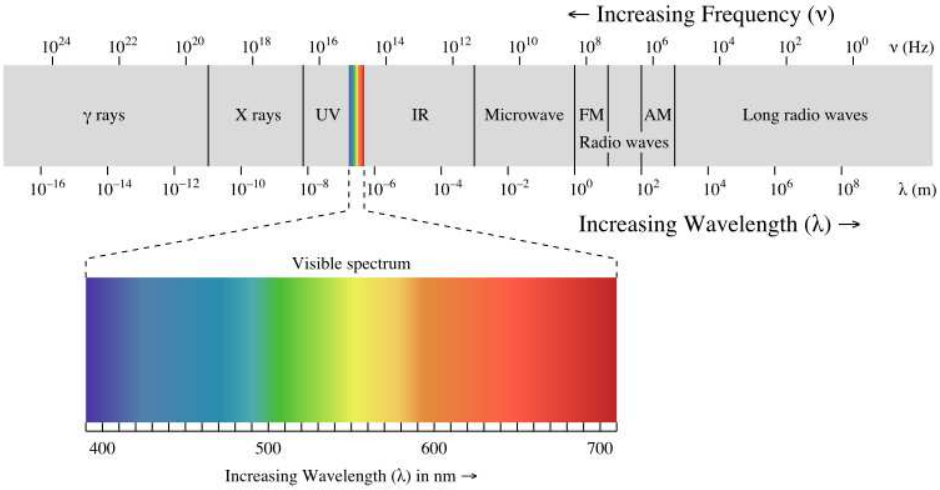
Como os decíamos según disminuimos la longitud de onda, los colores se hacen más azulados, y en el punto en el que dejamos de percibirlo, lo que estaremos recibiendo es radiación ultravioleta.

Si por el contrario, aumentamos la longitud de onda (y por tanto disminuimos la frecuencia), los colores se hacen más rojizos, hasta que llega un punto en el que ya no los podemos percibir. Estamos hablando de radiación infrarroja.

Por qué lo del infra y el ultra... muchos ya lo habréis imaginado, estos nombres hacen referencia a la frecuencia y no a la longitud de onda. Así que lo que está por debajo del rojo visible (menor frecuencia) se denomina infrarrojo, mientras que lo que está por encima del violeta visible se denomina ultravioleta :)

EL INFRARROJO CERCANO

Sí, las radiaciones infrarrojas tienen frecuencias inferiores a las del rojo visible, pero no todas las frecuencias inferiores a esta son radiación infrarroja.



Espectro Electromagnético (fuente Wikipedia)

Si seguimos bajando la frecuencia nos encontramos las microondas y más abajo las frecuencias utilizadas por la radio y la televisión. Sí amigos, tenéis dispositivos de alta frecuencia en vuestras cabezas XD.

Bien, la cuestión es que la región del espectro (esto es, todas las frecuencias posibles) que se considera radiación infrarroja, está dividida en tres grandes grupos, que se llaman, respectivamente, IR-A, IR-B e IR-C.

Con lo que nosotros vamos a jugar es con el IR-A, también conocido como infrarrojo cercano... os imagináis por qué no?. Efectivamente, porque es el que está más cerca del espectro visible. Sí, los más interesantes son los otros, los que se usan para guiar misiles y para obtener esas imágenes térmicas tan chulas que salen en las películas... pero efectivamente, hace falta algo más caro que una webcam de 25 euros para jugar con esas cosas.

Así que ahora que tenemos una idea general de que es lo qué estamos haciendo... pues vamos a ponernos manos a la obra.

DESMONTANDO LA CÁMARA

Nuestra webcam es una Logitech Messenger. Sí, aprovechamos todo aquí en Occam's... Que!, ¿todavía no os habéis leído el artículo sobre video vigilancia?... ¿a qué estáis esperando?

Primer consejo, perded un poco de tiempo intentando descubrir que tenéis que desmontar. Los tornillos suelen estar ocultos, porque son antiestéticos y, por la ley de Murphy, nosotros empezamos quitando justo los que no necesitábamos quitar XD.

Para poder desmontar esta webcam, tenemos que retirar una pequeña tapa de goma que se encuentra en uno de los laterales de la esfera que contiene la lente. En el agujero que cubre encontraremos un pequeño tornillo que debemos retirar para poder acceder al interior de la cámara.

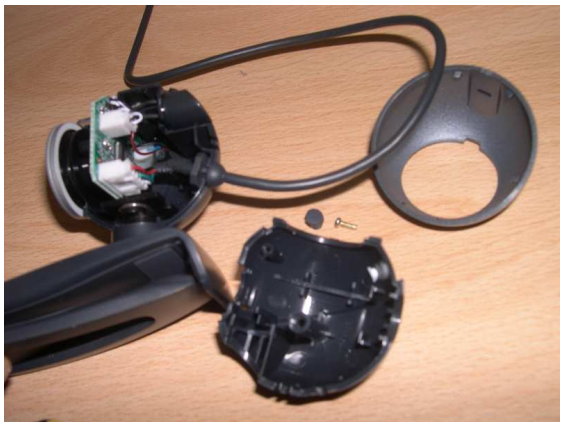
Accediendo al tornillo.

Por otra parte, con mucho cuidado, retiramos la carcasa gris azulada, de la parte frontal, ayudándonos de un destornillador o cualquier cosa que nos permita hacer palanca.

Quitando parte frontal.

Ahora ya podemos acceder a la cámara propiamente dicha, presionando el clip que une las dos mitades de la esfera.

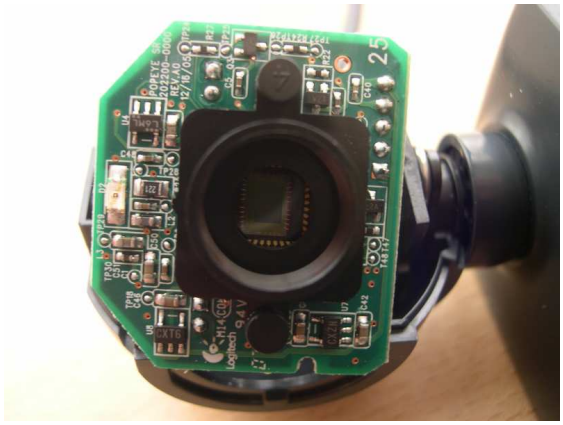
Retiramos una de las mitades, la guía de luz para poder ver el LED de actividad desde fuera (un pedazo de plástico transparente), y ya tenemos nuestra cámara "al desnudo" :).



Ya casi estamos.

El siguiente paso es desmontar la lente. En esta cámara (y al parecer en la mayoría de las webcams), la lente se enrosca directamente sobre el sensor que captura la imagen. Así que la giramos hasta que se separe del circuito.

Ahora tenemos, por una parte, la placa de circuito impreso, con el sensor de imagen y la rosca para la lente, y la lente por la otra.



Sensor de la webcam.

La lente que acabamos de desenrroscar.

FILTRANDO LA LUZ

Ooops!, no habíamos dicho nada sobre esto. Bueno, lo decimos ahora. Puesto que los sensores son sensibles al IR cercano, las cámaras digitales traen incorporado un filtro IR, ya que de lo contrario... bueno, eso lo podréis comprobar vosotros mismos ;).

El objetivo de todo esto es sustituir ese filtro IR, por un filtro de luz visible, de tal forma que la cámara solo capte el IR cercano y no la luz normal.

Como podéis ver en la imagen, el filtro IR se encuentra (siempre para nuestro modelo), en la parte de la lente, sujeto a ella con una especie de pasta negra. Con un cutter eliminamos parte de esta pasta y quitamos el filtro.

Filtro IR separado de la lente.

Efectivamente, ese cristalito rosado es el filtro IR. Que os esperabais, los filtros de luz son así. Tiene que ser transparente, para que pase la luz visible, y, pues eso, se llama infraRROJO por algo :).

El filtro IR es un cristal rosado entre la lente y el sensor.

La gran cuestión ahora es como hacer un filtro de luz visible. La respuesta la sacamos de Internet, yo no tenía ni idea la verdad, lo único que tenía claro, era que tenía que ser algo oscuro.

Pues nuestro filtro lo haremos con un negativo fotográfico velado... sí, ¿quién diría que algún día eso no sería tan fácil de conseguir eh?. Ojo, el negativo tiene que estar revelado y tendréis que escoger una de esas partes oscuras que normalmente quedan al principio o final del rollo.

Para nuestro modelo de cámara, utilizamos un perforador de hojas que tiene aproximadamente el mismo diámetro que el orificio de la lente. Acoplamos la lente de nuevo y a ver que pasa.

Nuestro filtro de luz visible.

Nosotros probamos la cámara antes de volver a montarla, pero vosotros haced lo que queráis :).

A JUGAR

Bien, pues después de todo este lío, vamos a ver que nos permite ver nuestra nueva cámara IR. En las páginas web de los recursos podéis ver varios ejemplos. Nosotros pondremos algunos más aquí.

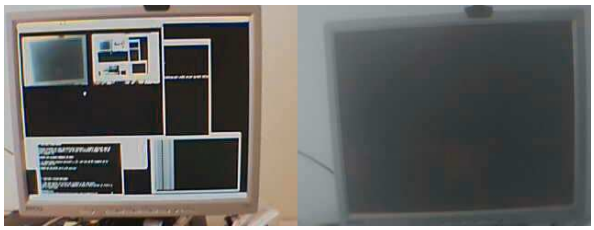
Lo primero que vamos a comprobar es que nuestro filtro de color funciona. Para ello, simplemente vamos a capturar una imagen multicolor y lo que esperaríamos obtener sería la misma imagen pero sin colores... que no en blanco y negro. Como podéis apreciar en la imagen, hay una sutil diferencia.

Parece funcionar bastante bien pero, normalmente, donde los filtros suelen tener problemas es en las denominadas frecuencias de corte. Para nuestro caso específico, esa frecuencia de corte se encuentra entre el rojo visible y el infrarrojo cercano.

En la siguiente imagen utilizamos un puntero láser de esos de color rojo para comprobar que lo que acabamos de decir tiene un cierto sentido :P.

Como podéis ver en la imagen de la derecha, aún cuando el color ha sido eliminado de la escena, la luz roja proveniente del puntero láser no es eliminada por nuestro filtro hecho con negativos fotográficos usados... bueno, nadie esperaría que fuera perfecto.

Las pantallas TFT no se ven... pero la televisión es otra historia. De la TV no tenemos imágenes, pero las podéis obtener vosotros mismos muy fácilmente.



VISIÓN DE RAYOS X?

Si buscáis más información sobre todo esto de las cámaras IR, os encontraréis entradas sobre "milagrosos ingenios para mirar a través de las cosas". Sí, es que suenan así de raros.

Lo que si es cierto es que al poder "capturar" la radiación infrarroja, y eliminar las componentes del espectro visible, ciertos detalles de las imágenes se hacen más evidentes.

Por ejemplo, si miramos a este lápiz USB que utiliza una carcasa azul transparente, apenas podemos ver su interior, pero si filtramos el color (el azul en este caso), la superficie se vuelva prácticamente transparente, y podemos ver los chips en su interior.

Lo mismo sucede con estas gafas de sol. Para ser sincero, las gafas que aparecen en la imagen son un poco cutres de estas de plástico, así que no podemos confirmar como funcionaría con unas gafas de las buenas :). Observad el ojo de Don... no nos la volverá a jugar en las partidas de póker.

La razón de la popularidad de estas páginas es que supuestamente permiten ver la ropa interior. Lamentablemente nadie se ha ofrecido voluntario para enseñar sus vergüenzas en aras de la ciencia.

COSAS RARAS

Como os adelantábamos hace un momento, cuando eliminamos la radiación visible de nuestras imágenes, los distintos materiales de nuestras ropas de muestran de diferentes formas.

En general, el color desaparece, ya que normalmente se consigue utilizando un tinte que la única propiedad del material que modifica es precisamente el color.

Aquí podéis ver como convertimos una moderna e informal chaqueta de rayas, en un elegante traje blanco :).

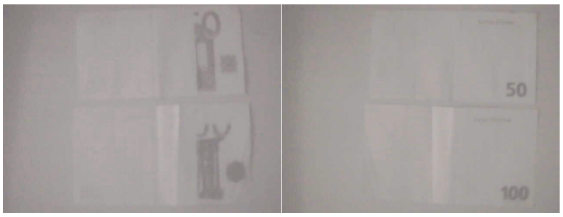
Lo mismo sucede con las plantas que en general se verán de color blanco, como podéis observar en la siguiente imagen. Nuestro filtro de color elimina el característico color verde.

Lo curioso de esto es que podemos hacer bonitas estampas invernales en pleno agosto :).

Otra de las características de nuestra nueva cámara es que nos permite observar algunas de las medidas de seguridad incluidas en los billetes. En la siguiente imagen podéis observar un billete de 5 euros y otros de 10.

Como en la mayoría de los ejemplos anteriores, gran parte del color desaparece, pero la zona entorno al holograma, está impresa con una tinta especial (algo más que simplemente color). En el caso particular de los billetes, resulta más efectivo e interesante utilizar luz ultravioleta o luz "negra" como suele ser más conocida.

En la siguiente imagen podemos ver un billete de 50 y otro de 100 por ambos lados. Nótese como el número de serie y la cantidad se imprimen con una tinta especial.



Finalmente, aquí podéis ver mi cartera que esta un poco para ser retirada. Parece que las zonas mas desgastadas se resaltan en nuestra camara IR.

LUZ INVISIBLE EN LA OSCURIDAD

Y por supuesto, nuestro mando a distancia de la tele se convierte en una linterna que no da luz XDDDDD. Y es precisamente esta característica la que resulta más interesante desde el punto de vista de las aplicaciones de nuestra webcam IR.

Unas imágenes curiosas ¿eh?... Pero muchos os preguntaréis, y ¿para qué sirve esto?. Bueno, además de para quedarte con los colegas. La pregunta exacta que probablemente se os esté viniendo a la cabeza es: ¿Qué hace una gente tan discreta como los tíos de Occam's, con algo que solo sirve para chulear?

Buena pregunta. Efectivamente, disponer de una cámara IR nos permite abordar algunos proyectos muy interesantes, sobre todo desde el punto de vista de los denominado MMI (*Man Machine Interface* o Interface Hombre Máquina).

La primera y la más obvia de estas aplicaciones es lo que los ingleses denominan tracking y que nosotros podemos llamar seguimiento.

TRACKING

Como os decíamos, las aplicaciones más interesantes y directas de este tipo de cámaras es el de los interfaces MMI. La ventaja más importante... pues que la radiación infrarroja es invisible para el ojo humano.

Esto se traduce en que nuestra cámara va a capturar unas imágenes totalmente oscuras con un pedazo de punto blanco allá donde haya una fuente de luz infrarroja. Como podéis imaginar, esto hace prácticamente trivial el localizar esas fuentes de luz IR en la imagen.

Para utilizar estas técnicas, tenemos dos opciones fundamentales. Hacer que la "cosa" que queremos seguir emita luz IR, o que la refleje.

En general si tenemos que seguir muchos puntos es mejor utilizar elementos que reflejen la luz IR, de lo contrario habría que llenar el objeto a seguir (pensad por ejemplo en un sistema de captura de movimiento de personas para un juego) de cables y baterías.

Si por el contrario, solo necesitamos un par de puntos de referencia, por ejemplo para un dispositivo apuntador en un sistema de realidad virtual, un par de LEDs y unas pilas harán el trabajo.

El wiimote utiliza una barra sensora con luz IR para calcular la posición del jugador

Este tipo de soluciones son utilizadas, por ejemplo, por el mando de la consola Wii (el denominado wiimote), como parte de su sistema de posicionamiento. El mando incluye un acelerómetro y una cámara infrarroja los cuales, junto con una barra sensora permiten calcular la posición y orientación del jugador.

La barra sensora está compuesta de dos grupos de LEDs infrarrojos, separados una distancia conocida. La cámara en el mando detecta estos dos grupos y el procesador interno hace las matemáticas :).

Con un puñado de LEDs infrarrojos, algunas resistencias y nuestra webcam IR podemos explorar este tipo de sistemas.

PANTALLAS TÁCTILES

Otra aplicación en la que podemos utilizar nuestra flamante webcam IR es la construcción de una pantalla táctil multi-toque :)... si, como la Microsoft Surface y similares.

Este tipo de pantallas se pueden construir de distintas formas, pero una de ellas, bastante asequible para el aficionado al cacharreo es la denominada FTIR que son las siglas de *Frustrated Total Internal Reflection*. Podríamos traducirlo por Reflexión Interna Total Frustrada, y enseguida veremos que el nombre tiene cierto sentido.

Estas pantallas utilizan una superficie acrílica (un plástico), en cuyo interior se inserta luz infrarroja, de forma que no es visible por el ojo humano. La luz insertada rebota en el interior del material pasando de un extremo al otro.

Si presionamos la superficie con, por ejemplo, nuestros dedos, la deformación que producimos en la superficie, interrumpe el rebote de la luz IR en su interior y produce un efecto de dispersión... es decir, que se produce un "puntazo IR"... Ahora lo de reflexión interna frustrada tiene más sentido ¿no?

Todo este proceso es completamente transparente al ojo humano ya que la luz utilizada es IR. Pero para una cámara IR, esos "puntazos" que se forman al presionar la superficie se pueden capturar y procesar muy fácilmente.

Otra técnica bastante utilizada en la denominada Iluminación Difusa (*Diffuse Ilummination*). Existen dos formas de utilizar esta técnica, pero la que nos interesa a nosotros es la denominada *Rear Diffuse Illumination*.

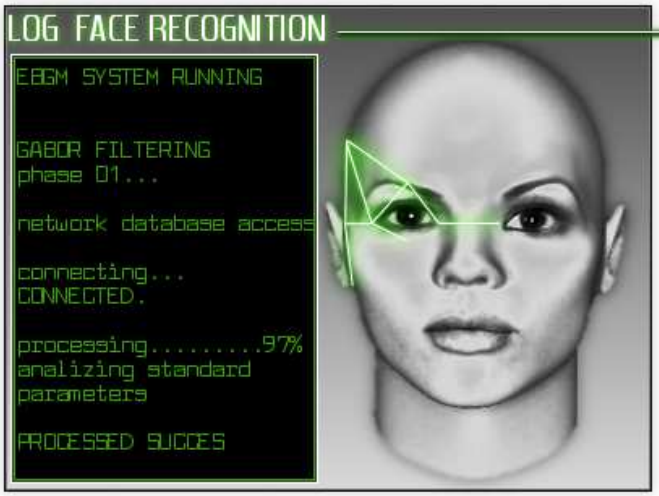
El sistema se basa en colocar un difusor sobre la superficie táctil e iluminarlo con luz infrarroja. Cuan el usuario pulsa sobre la superficie su dedo reflejara más luz IR que la que refleja el difusor, de tal suerte que la cámara pueda detectar el punto correcto. El principal problema de esta técnica es el conseguir una iluminación uniforme en toda la superficie.

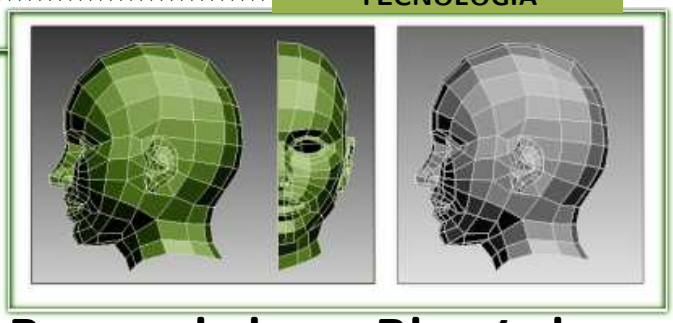
El hecho de que sea multi-toque o no, va a depender del software que procese las imágenes y lo que ese software haga con los datos que obtenga.

Esto es todo. Esperamos que os haya gustado esta nueva sección. ■

RECURSOS

How to make a webcam work in infra red
http://www.hoagieshouse.com/IR/
Pantallas Multitouch FTIR
http://en.wikipedia.org/wiki/Total_internal_reflection#Frustrated_total_internal_reflection
http://en.wikipedia.org/wiki/Multi-touch
NUI Group Community Book - "Multi-Touch Technologies"
http://nuigroup.com/log/nuigroup_book_1/





Reconocimiento Biométrico a Través de la Cara

por Fernando Martín Rodríguez (fmartin@tsc.uvigo.es)

Siendo éste un número dedicado a las tecnologías de seguridad no podía faltar un artículo de la serie biométrica. Advertido queda que seguramente será el último... Ante lo que seguramente estará pensando la dirección de la revista digo que no, no me siento capaz de escribir sobre el reconocimiento a través de voz. Hoy toca el reconocimiento a través de imágenes de la cara.

Fijaos que este es el método que más empleamos los humanos para reconocer personas. En el 99 % de los casos reconocemos a alguien al ver su cara, también reconocemos personas por la voz pero casi siempre vemos las caras antes de oír nada. Ya puestos, comentar que el reconocimiento humano es muy fiable porque utiliza múltiples datos, esto es: vemos la cara pero también la silueta (la estatura, complexión...), oímos la voz y también nos fijamos en cosas muy difíciles de medir para una máquina: los gestos, la forma de andar.... La suma de todos esos factores (una suma ponderada que hace el cerebro sin que lo notemos) da un reconocimiento muy fiable. Realmente ese método (la redundancia) es el que usa el cerebro humano para casi todas sus tareas de reconocimiento.

INTRODUCCIÓN

Para mantener la tradición, empezaremos recordando las características que un rasgo debe cumplir para dar lugar a un buen sistema biométrico y, ya de paso, veremos cómo se cumplen para las imágenes faciales. A saber:

- Universalidad: todas las personas deben poseer el rasgo. Bueno, eso es cierto... por lo menos para personas vivas.
- Unicidad: no debe haber dos personas con el rasgo seleccionado igual. Aquí ya empezamos a no cumplir... aparte de gemelos idénticos también hay pares de personas parecidas (más probables en gente con parentesco pero no necesariamente).
- Permanencia: debe ser invariantes con el tiempo. ¡¡La cagamos!! No hay rasgo biométrico más variable que la cara: barba, gafas, pelo largo o corto... Aparte de las diferencias (notables) que puede llegar a haber en la captura de imagen debidas a la postura frente a la cámara (pose), el gesto (sonrisa, enfado, bostezo ...). Pensad que en caso de usuarios malintencionados podremos tener casos de disfraz. Todo lo anterior, unido a que el número de imágenes que podemos tener almacenadas de la cara de alguien es limitado hace que éste sea uno de los rasgos biométricos menos fiables que se conocen.
- Cuantificación: admiten medidas cuantitativas. Sí que las admiten... si no, no sería posible ni intentarlo y no estaríamos escribiendo esto.

El reconocimiento de caras es el método más utilizado por los humanos para reconocer personas

Aun a riesgo de repetirme, voy a copiar otra vez el cuadro comparativo de los diferentes sistemas biométricos. Recordad que TFA significa "Tasa de Falsa Aceptación" y es el porcentaje de veces que el sistema declara como iguales a dos individuos que no lo son, mientras que TFR es "Tasa de Falso Rechazo" y es el porcentaje de veces que dos individuos iguales son erróneamente considerados diferentes. Fijaos que aunque dichas tasas pueden llegar a ser bajas (en experimentos muy controlados), la memoria y el tiempo requeridos son altos y la facilidad de engañar al sistema también.

Indicador biométrico	TFR	TFA	Memoria (bytes)	Tiempo respuesta	Variabilidad intra-clase	Variabilidad inter-clase	Implantación en mercado	Facilidad de engaño
Huellas	3 %	$1:1^6$	100-1000	Depende del sistema	Media	Alta	52 %	Media
Retina	Bajas	aprox. 0 %	Poca	Bastante bajo	Muy baja	Muy alta	<1 %	Baja
Iris	Bajas	aprox. 0 %	1024 bits	Medio-bajo	Muy baja	Muy alta	7,3 %	Baja
Cara	Bajas	Bajas	>10,000	Alto(1:N) Bajo (1:1)	Muy alta	Alta	11,4 %	Muy alta
Voz	Bajas	Bajas	>1000	Alto (1:N) Bajo (1:1)	Alta	Media-alta	4,1 %	Muy alta

Tabla 1: Principales características de algunos indicadores biométricos.

El conocido gurú de la seguridad Bruce Schneier (inventor del algoritmo de cifrado Blowfish) escribió que era inviable el uso de un sistema de reconocimiento de caras en una aplicación masiva como, por ejemplo, detectar sospechosos en un aeropuerto. Con una TFA típica del 1 % significaría "parar" a una de cada 100 personas que pasen por delante de una cámara (y obligarles a identificarse utilizando un método más fiable, por ejemplo: colocando el índice en un lector automático de huellas). Si en Barajas (el aeropuerto de Madrid) hay casi 150.000 pasajeros al día (ese es el dato que encontré en Internet) tendríamos que identificar a 1500 personas/día. ¡¡¡Ojo!!! que no estamos contando a los que van a esperar a alguien... Además, ser identificado por la policía para que resulte ser un error es algo bastante desagradable y que puede dar lugar a denuncias, quejas... ¿Qué ocurriría en el aeropuerto de Chicago (más de 250.000 pasajeros/día)?

El reconocimiento de caras sí es viable en grupos cerrados de pocos usuarios (control de acceso a un recinto o a un sistema informático...) y es en esos casos cuando se obtienen buenos resultados en TFR y TFA.

PREPROCESADO

Como en todo problema de reconocimiento de imágenes, primero debemos extraer la zona de interés. En el reconocimiento de caras se suele buscar una cara frente a la cámara, con los ojos en posición conocida y de un tamaño fijo. Además, deberíamos normalizar el brillo de la imagen de forma que no haya variaciones grandes de iluminación. Llegar a tener esa imagen a partir de lo que ha capturado la cámara es lo que llamamos preprocesado y es muy importante hacerlo bien para que el reconocimiento funcione. En imágenes capturadas "de cualquier manera", esto es: las imágenes de una cámara de seguridad que apunta a un pasillo o a una cola de personas puede no ser posible hacer un preprocesado correcto.

Comenzaremos preprocesando las imágenes

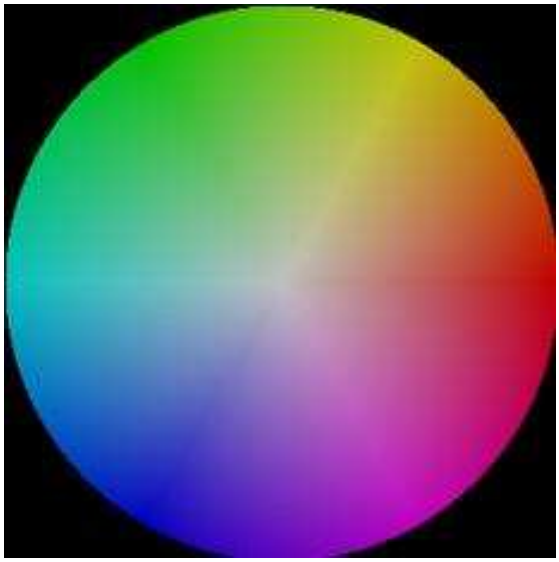
A continuación describiremos dos técnicas muy populares para localizar caras. Además, no son excluyentes, pudiendo haber sistemas que combinen ambas.

LOCALIZACIÓN POR COLOR

Una de las técnicas más empleadas para localizar caras es la detección de aquellos puntos de la imagen que tienen un color similar al de la piel humana. Este método nos dará una serie de regiones con alta probabilidad de contener caras. Cada una de esas regiones (blobs) deberá ser procesada posteriormente para determinar si realmente es una cara o no. Una aplicación es, por ejemplo, buscar caras en una imagen compleja e impredecible (como las imágenes de una cámara de vigilancia).

Para entender bien la localización por color debemos saber que la representación habitual de los colores (R, G, B) no es la única posible y se puede transformar a otras representaciones (espacios de color), todas ellas basadas en tres componentes. Una representación muy interesante es la HSV que divide el color en tres números:

- El tinte (Hue, H): es un ángulo y define la posición del color en un círculo que empieza en el rojo (ángulo 0) y que recorre todo el arco iris. El tinte es lo que coloquialmente llamamos "color": rojo, naranja, verde ... Otra cosa es que rojo sea intenso, brillante ...
- La saturación (Saturation, S): se mide entre 0 y 1 y es la "pureza" del color. Si compramos pintura roja debería ser rojo puro (saturación 1 o rojo al 100 %). Si deseamos "rebajar" ese rojo, podemos mezclar la pintura roja con blanca e iremos logrando diferentes "niveles" de rojo, si tenemos 50 % de rojo y 50 % de blanco la saturación será 0.5.
- El brillo (llamado Value, V; a veces se le llama intensidad y se habla de sistema HIS): el brillo es la "potencia" del color o la iluminación que proporciona. El color blanco es el más brillante y el negro tiene brillo cero. Hace años se usaban monitores verdes porque es un color que da gran sensación de brillo para poco gasto en energía, las luces azules para estudiar usan la idea contraria (es un color que ilumina poco pero cansa menos la vista).



Círculo de Color. El ángulo polar determina el tinte, la distancia al centro la saturación.

Para la piel humana....

- El tinte es ¡¡¡naranja!!! ¿Qué ocurre si subimos mucho el color del televisor? No nos vemos naranjas porque la saturación es baja.
- El color varía mucho para los diferentes tipos de piel. La raza amarilla es la más "saturada". La raza negra tiene brillo muy bajo (y eso puede introducir errores de redondeo en el cálculo del tinte).
- Para hacer un reconocedor serio debemos utilizar reconocimiento de patrones. Esto es: tomar muchas fotos de individuos con diferentes tipos de piel y, de alguna manera, comparar la imagen que tengamos a la entrada con las previas (conjunto de entrenamiento).
- El sistema de color HSV es el más intuitivo para este problema pero se utilizan otros.

Para ampliar un poco estos conocimientos podéis leer el sencillo artículo: "Detección de Caras en Imágenes Capturadas por Cámaras Web" (autores: Patricia Conde Pardo y yo mismo). Sí, sí, lo he vuelto a hacer: "me acabo de hacer autobombo". Lo podéis descargar de http://webs.uvigo.es/gpi-rv/research-papers.html.

EL ALGORITMO DE VIOLA

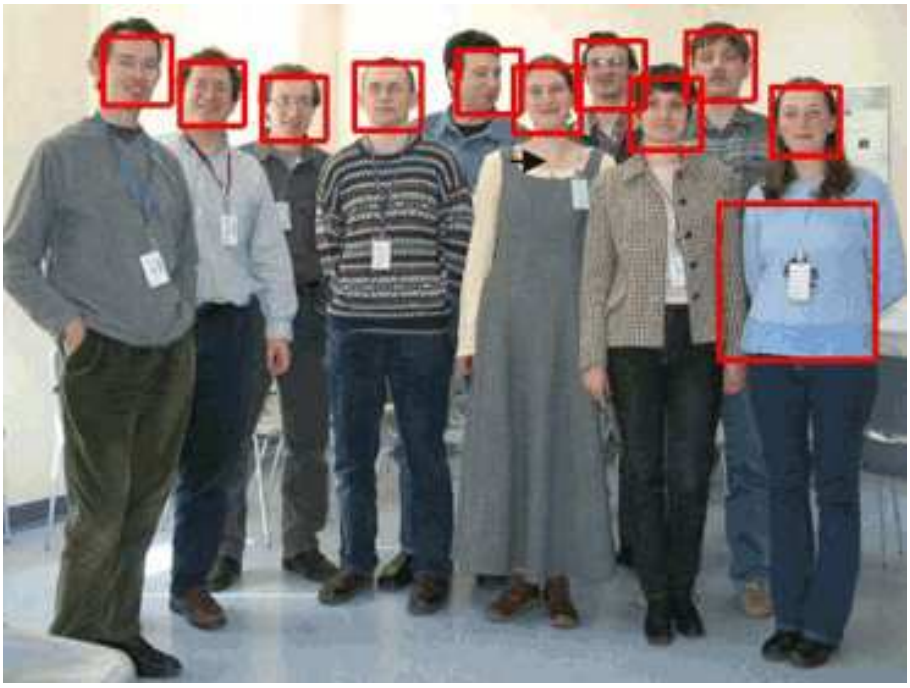
Actualmente existe un método capaz de detectar caras de cualquier tamaño en una imagen blanco y negro. Fue diseñado por Paul Viola y publicado en "Rapid Object Detection using a Boosted Cascade of Simple Features," (autores P. Viola y M. J. Jones, publicado en 2001 en el congreso del IEEE sobre visión artificial y reconocimiento de patrones: CVPR). Para este algoritmo disponemos de una implementación de código abierto (ver cuadro de recursos).

Este código es parte de la excelente biblioteca "OpenCV" (conjunto de algoritmos de visión artificial donados por Intel al mundo): http://opencv.willowgarage.com/wiki/. Ufff, las multinacionales, a veces, también tienen su corazoncito :).

El algoritmo de Viola se basa en la extracción de características de tipo Haar

El algoritmo se basa en las llamadas "Haar-like features" (características tipo Haar, que se llaman así porque se calculan de una forma parecida a una transformada llamada "transformada Haar" o wavelet de Haar). Cada característica es como un operador gradiente pensado para resaltar un patrón. Los patrones son características como: contornos, líneas, cruces... La imagen se recorre con una ventana a la que se le aplican varios clasificadores en serie, cada uno más complejo que el anterior, que usan las características para confirmar o descartar la hipótesis de que estamos ante el objeto buscado (en este caso una cara pero podría entrenarse para encontrar otros objetos). Si la hipótesis se rechaza en cualquier nivel, el proceso no continúa pero si se confirma tras todos los filtros tendremos un objeto detectado.

A: Imagen original. B: se ha aumentado la saturación logrando resaltar la piel visualmente (y hacer evidente el tinte anaranjado). C: detección del óvalo facial utilizando colorimetría (se ha obtenido un blob del que se ha calculado un contorno poligonal: en verde, y una elipse de ajuste: en rojo).



Resultado de aplicar la función OpenCV. Nótese que hay un falso positivo.

Los patrones se consideran girados en varios posibles ángulos. Además, el algoritmo puede ejecutarse a varias escalas para obtener objetos de diferentes tamaños o de tamaño desconocido.

Una vez que hemos localizado las caras en nuestra imagen, ha llegado el momento de reconocerlas. Se han probado muchas técnicas para el reconocimiento facial. Aquí sólo vamos a describir las dos más extendidas.

ANÁLISIS DE COMPONENTES PRINCIPALES

El análisis de componentes principales (o PCA: Principal Component Analysis) se basa en representar las caras como combinación lineal de ciertas caras básicas (autocaras).Además, conocemos cuáles son las autocaras más importantes por lo que podemos quedarnos sólo con ellas y descartar las demás. Los coeficientes que permiten representar la cara de entrada serán utilizados como características para el reconocimiento. A esta técnica también se le llama proyección en subespacios o también "transformada de Karhunen-Loève".

El proceso de reconocimiento hace uso de álgebra matricial

El nombre de proyección en subespacios viene de la teoría matemática que se utiliza: se trata de convertir el conjunto de imágenes de caras en un subespacio vectorial (un subconjunto de un espacio mayor: el de imágenes). Para realizar el análisis debemos encontrar una base del subespacio, esto es: un conjunto de caras tal que podamos representar cualquier cara como combinación lineal de las caras de la base. Es más: no nos vale cualquier base... queremos una base "ordenada en importancia".

El PCA nos permite generar una base de vectores ordenada por importancia

Quiero decir: el primer vector será el más importante, el segundo ya será menos importante pero más que los siguientes... Una base con esas características "concentra la información en los coeficientes de los primeros vectores". Podremos llegar a decidir que, por ejemplo, los 50 primeros componentes son suficientes con lo que los otros (N-50, con N muy grande) los ignoramos. Si al representar una imagen con los 50 primeros coeficientes obtenemos una distancia euclídea baja respecto de la imagen original: "Esa imagen será una cara". Si la distancia obtenida es grande, no será una cara; será otra cosa.

¿Cómo se consigue lo anterior? Bueno. . . en cuatro palabras: "utilizando álgebra de matrices" (no me atrevo a decir que es álgebra avanzada pero puede desafiar un poco los conocimientos olvidados del ingeniero medio). Suponiendo que nuestras caras son imágenes MxM, tendremos un espacio total de dimensión $N = M^2$. Tomando muchas caras podemos calcular una matriz de autocorrelación para esos vectores (que resumirá la "estadística" de las caras). Ahora empieza lo "fuerte":

- La matriz de autocorrelación será definida positiva (esto es: todos sus autovalores son reales y positivos) álgebra) por lo que podremos construir una base ortonormal de autovectores (creéroslo, es un teorema). ¿Qué es una base ortonormal? Aquélla formada por vectores unitarios y perpendiculares entre sí.

- Esa será nuestra base. Los autovectores se les llama "autocaras" porque son eso: caras (que pueden combinarse para formar cualquier cara). La autocara más importante es la asociada al mayor autovalor y así sucesivamente (ordenando los autovalores de mayor a menor tenemos ordenada la base).

- Una vez hecho esto habría que decidir cuántos vectores retenemos y cuántos descartamos... un indicativo puede ser el peso de los autovalores. Una medida simple como:

  $$\rho = (\sum_{i=1}^{p} \lambda_i)/(\sum_{i=1}^{N} \lambda_i)$$

  estima el peso de los p primeros autovalores respecto del total: si ρ es cercano a 1 el peso es grande. De todas formas, el número suficiente de componentes debe ser verificado experimentalmente. Esto es: si implementáis esto y no funciona tal vez haya que subir p. Otro método es empezar con p muy grande e irlo bajando hasta que falle (funcionar/fallar significa que el conjunto de p autocaras representa bien las caras (pequeño error euclídeo).

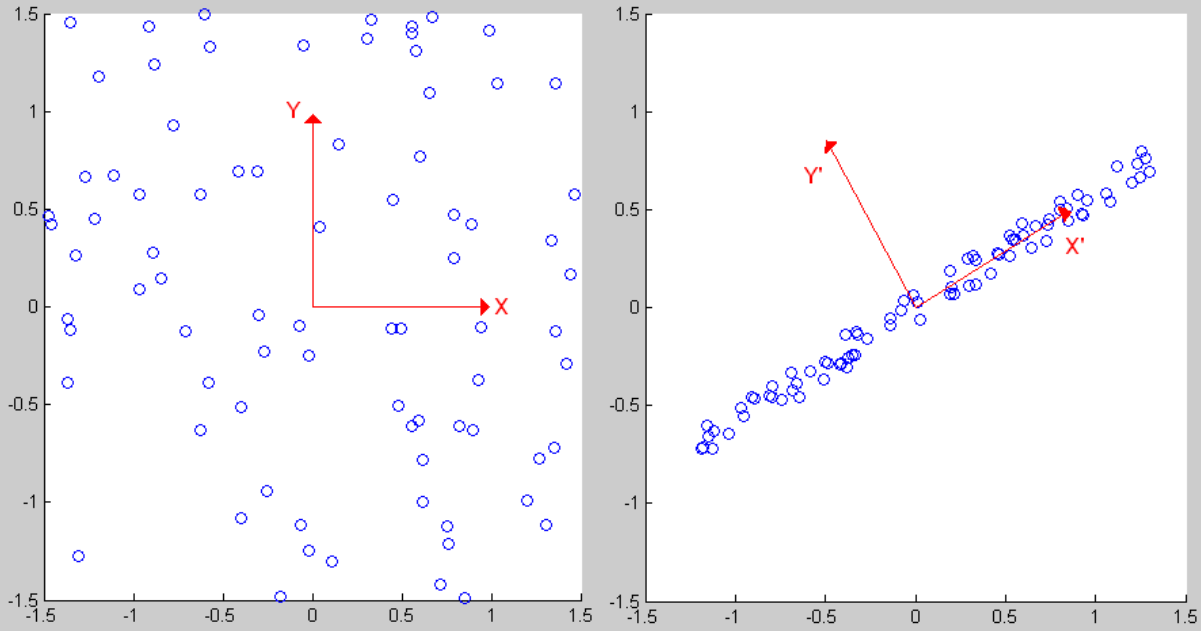
Pensad que con todo esto hemos reducido la dimensión del espacio de caras de $N = M^2$ a p. Si $M = 512$ y $p = 50$

Cuando queremos reconocer una imagen lo primero es calcular sus coeficientes para las autocaras seleccionadas. A eso se le llama "proyectar sobre el subespacio de autocaras" (de hecho, el coeficiente para cada autocara es muy fácil de calcular, basta un producto escalar entre la autocara y la cara de entrada).

Después podríamos comprobar si la entrada es realmente una cara (calculando el error entre la cara representada por las autocaras y la cara misma). Si esa comprobación es positiva (error bajo), ahora toca saber de quién es la cara... para eso, hay que comparar el vector de características de la cara de entrada (coeficientes de la cara respecto a las autocaras) con los vectores almacenados (procedentes de caras que queremos reconocer).

PCA fácil

Para intentar hacer más comprensible esto de los subespacios, permitidme un ejemplo muy simplificado. Supongamos que nuestras imágenes son de tamaño 2x1 (eso sí que es simplificar). Entonces las imágenes serán puntos del plano (figura de la izquierda).

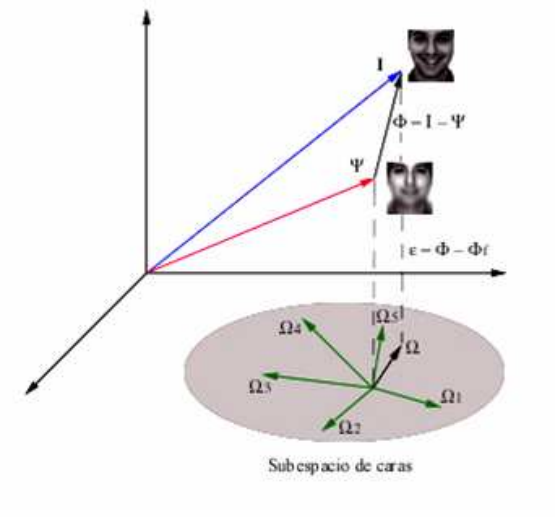


El "quiz" de la cuestión es que no nos interesan todas las imágenes, sólo queremos procesar caras. Siguiendo con nuestra simplificación, si las imágenes de interés son puntos que "aproximadamente" están situados sobre la misma recta (figura de la derecha); podremos hacer una simplificación (creación de una base de autovectores) que viene a ser un giro de ejes de forma que la primera componente quede en la dirección principal.

¿Podemos despreciar la componente Y'? Yo diría que sí. Además, podemos saber fácilmente si un vector es del conjunto: si la representación eliminando Y' comete un error pequeño ¿NO?

Ejemplo de un conjunto de autocaras. La autocorrelación se hace restando la media a cada muestra por lo que debemos tenerla también.

Aquí se pueden usar muchas técnicas pero la más simple (y una de las más efectivas) es la del vecino más próximo. Esta técnica consiste en calcular distancias entre los vectores de características (euclídea o norma 2, norma 1...) y considerar como resultado reconocido, el patrón almacenado más próximo al de entrada. ¡¡¡Ojo!!! puede haber varias muestras por individuo a reconocer.



Proyección en subespacio de caras.

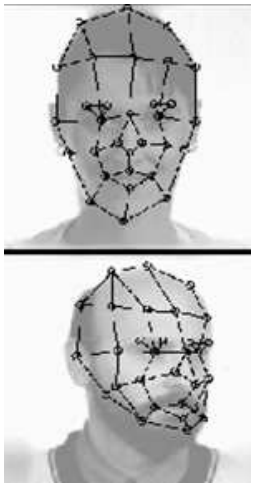
TÉCNICAS DE REPRESENTACIÓN CON MALLAS DEFORMABLES

La idea de este método es "colocar una máscara deformable sobre la cara (con forma de red o malla) y ajustarla para que los nodos de la malla coincidan con puntos importantes como los ojos, nariz... (puntos fiduciales)". Este método es la versión informática del método que hemos visto muchos en televisión: encontrar características como la distancia entre los ojos, la posición de la punta de la nariz...

La base matemática del método es aplicar filtros de Gabor en cada nodo y utilizar las respuestas obtenidas para ir ajustando la malla hasta que esté colocada sobre los puntos deseados. Como vimos en el capítulo de las huellas el filtro de Gabor extrae características del entorno del punto, además los filtros Gabor tienen una orientación que hace que sean sensibles a las variaciones con determinado ángulo. Aplicando los filtros en múltiples direcciones podemos calcular una especie de vector gradiente que nos permita mover el nodo en la dirección deseada.

la localización de los puntos fiduciales de la cara es más compleja de lo que parece en la TV

Una vez ajustadas las mallas, los sujetos se comparan mediante una medida de similitud que tiene dos términos: la similitud entre los vectores de textura de cada nodo y la similitud por deformación de cada malla.



Representación con mallas deformables (método EBGM: Elastic Bunch Graph Matching.

TRABAJOS DE LA ETSET VIGO

En nuestro centro, el profesor José Luis Alba (jalba@gts.tsc.uvigo.es) y varios colaboradores llevan algún tiempo trabajando en el reconocimiento a través de caras. Dichos trabajos forman parte de una línea más amplia dedicada al reconocimiento biométrico en general.

Originalmente sólo se trabajó con reconocimiento por voz (reconocimiento de locutores) y posteriormente se añadieron los reconocimientos de caras, iris y huellas. Actualmente, se está desarrollando un sistema de verificación multimodal (voz y cara) para acceso a servicios a través de Internet. Esta línea de investigación se sigue en cooperación con otros grupos de trabajo europeos en el seno de la acción COST275: "Biometrics-Based Recognition of people over the Internet". Además, estos estudios reciben financiación a través de proyectos del Ministerio de Educación y de la Xunta de Galicia y parten de un sistema ya desarrollado previamente con fondos públicos en el que se utilizaba reconocimiento de voz para acceder a un buzón de correo por línea telefónica (http://www.gts.tsc.uvigo.es/telcorreo/). ■

RECURSOS

Aalgoritmo de Viola

http://en.wikipedia.org/wiki/Robust_real-time_object_detection

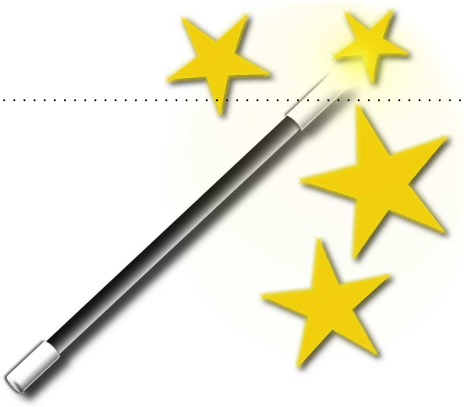
http://www.cmucam.org/wiki/viola-jones

Librería OpenCV

http://opencv.willowgarage.com/wiki/ http://sourceforge.net/projects/opencvlibrary/

Recursos en ETSET Vigo

http://www.gts.tsc.uvigo.es/telcorreo/



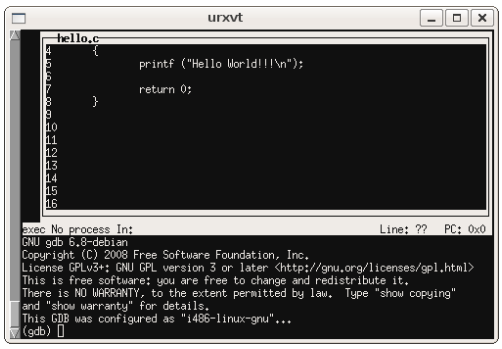
Con un par... de líneas

Chuletillas para hacer cosas mú rápido

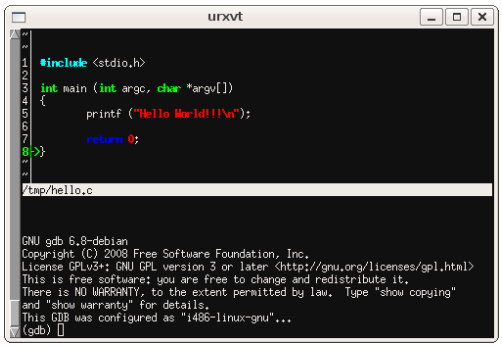
por Tamariz el de la Perdíz

GDB CON INTERFAZ MAS AMIGABLE

El depurador de GNU gdb, proporciona un interfaz *más amigable*, si es que eso es posible. Para echarle un ojo, solo tenéis que utilizar la opción `-tui` (algo así como *Text User Interface*). Lo que obtendréis es algo como esto:



Os adelantamos que, este interfaz se suele ir a paseo en cuanto nuestro programa empieza a escribir cosas en la consola. Si necesitáis algo todavía más amigable podéis probar `cgdb`.



DEPURANDO CON GDB Y LIBTOOL

Si utilizáis la autotools de GNU para compilar una aplicación que incluye librerías dinámicas, habréis observado que el supuesto ejecutable que generan las herramientas, es realmente un *script*. Si intentáis utilizar `gdb` con ese *script*, obviamente, solo conseguiréis un bonito mensaje de error.

Para poder depurar nuestro programa, tenemos que utilizar `libtool` de la siguiente manera.

```
$ libtool --mode=execute gdb ./mi_ejecutable
```

HELPDESK CON SCREEN

Seguro que alguna vez habéis estado al teléfono un montón de tiempo dictando comandos a alguien en el otro lado. Al mismo tiempo, vuestro interlocutor os va diciendo que es esa cosa que hace que no le funciona (porque falta un espacio, porque no ha pulsado ENTER, ...).

La utilidad `screen` puede hacernos la vida más fácil en estas situaciones. El modo multiusuario nos permite conectar dos instancias de `screen`, de forma que podremos ver lo que el otro usuario escribe y nuestro cliente podrá ver que es lo que nosotros escribimos.

Para conectarnos a una sesión `screen` en modo usuario, sólo tenemos que utilizar el flag `-x`, al invocarlo.

VARIAS IPs

La forma más sencilla de configurar varias IPs es disponiendo de varios interfaces de red. Si no tenemos varias tarjetas podemos utilizar *alias*. Esta es la forma de definirlos.

```
# ifconfig eth0:1 192.168.100.1 netmask 255.255.255.0
```

El comando anterior crea un alias de nuestro interfaz eth0, llamado eth0:1 con la nueva configuración de red. Notad que esto es solo un alias y no podéis utilizarlo, por ejemplo, en las reglas de vuestro firewall.

Para configuraciones más complejas disponéis de otras alternativas como las VLAN (ya hablaremos de ellas en otra ocasión).

PRODUCE VIDEOS DESDE TU WEBCAM

Solo tenemos que escribir esta línea:

```
$  mencoder -tv driver=v4l -ovc lavc -o movie.avi tv://
```

Y tener instalados los paquetes apropiados (mplayer, mencoder, ffmpeg, libavcodec).

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

Un Desafío Criptográfico

por Fernando Martín Rodríguez (fmartin@tsc.uvigo.es)

Después de escribir el artículo sobre criptografía me entraron unas ganas irresistibles de ponerme a inventar métodos. Hablando con el director de la revista se nos ocurrió esta idea: publicar un desafío. Esto es, lo que sigue es un algoritmo de criptografía de nuestra invención. Tan público va a ser que os vamos a dar el código fuente. Después va un criptograma y se trata de descifrarlo... sin saber la clave claro.

En el próximo número se publicará la solución y los nombres de todos aquéllos que hayan enviado soluciones válidas. Para enviar una solución, enviad un correo a occams-razor@uvigo.es con:

- Nombre completo.
- Criptograma descifrado.
- Valor de la clave.
- Una explicación breve de lo que hace el método y de cómo lo habéis atacado.

Ánimo, con las pistas dadas, no es difícil.

El cifrador son dos funciones de matlab... ahí van (sin comentarios):

Función principal:

```
function criptograma = CifrarTexto(llano,clave)
raiz = floor(sqrt(clave));
resto = clave-raiz^2;
N = length(llano);
for i=1:N,
    [raiz, resto, digito] = DigitoDecimalRaizCuadrada(raiz,resto);
    cfr = llano(i)+digito;
    if cfr>90
        cfr = (cfr-91)+65;
    end
    criptograma(i) = char(cfr);
end
```

Función auxiliar:

```
function [raiz, resto, digito] = DigitoDecimalRaizCuadrada(raiz0,resto0)
% Obtener el siguiente decimal de una raiz cuadrada
resto1 = resto0*10;
resto2 = resto1*10;
aux1 = 2*raiz0;
digito0 = floor(resto1/aux1);
for cont = digito0:-1:0,
    aux2 = aux1*10+cont;
    aux3 = aux2*cont;
    resto = resto2-aux3;
    if (resto>=0)
        digito = cont;
        break;
    end
end
raiz = raiz0*10+digito;
```

¡¡¡Aviso!!! No he "encriptado" los nombres de las funciones ni de las variables. No os doy el programa que hace el descifrado. ¿Podríais crearlo?

Y ahora el criptograma:

```
DWSGNGGJBVVPIWYXXKJSGHVLLPXVHQYWYAJBTLEDNLAROLXTBGINPFZYJRPXUMUFCVCJTMXRDKXIFUXRW
```

Por último, para comprender totalmente el método estaría bien leer esto:

http://platea.pntic.mec.es/anunezca/ayudas/algoritmo_raiz/algoritmo_raiz.htm ■

Participa en OCCAM's

mándanos tus propuestas de artículos a

occams-razor@uvigo.es

OCCAM'S RAZOR

Porque lo más sencillo es lo más probable

nº1

nº2

nº3

nº4

NO TE CORTES CON LA NAVAJA DE OCCAM

descarga todos los números en http://webs.uvigo.es/occams-razor