DebianHackers

Elementals

2014

Editorial

En Debian Hackers estamos de celebración. Ya son **cuatro años** desde que a Dabo y Diego les dió por abrir una web en la que contar sus fechorías a mandos de la espiral rosada; más de dos desde que nos unimos Euge y un servidor. Y esto sigue con la misma fuerza de siempre. Supongo que el secreto está en esa especie de ciberanarquía reinante que hace que publiquemos nuestra particular celebración una semana y pico después del aniversario. Cuando y como surja, sin presiones, sin obligaciones. Cualquier otra cosa sería contraria a la naturaleza del sitio, al espíritu con el que nació Debian Hackers. Pero mentiría si dijera que ese es el único secreto. Porque la verdad es que el motor de todo esto sois vosotros. Sí, **vosotros**, que nos leéis, que nos escribís, que nos ayudáis a dar forma a esta pequeña comunidad debianita. Es por vosotros que nos metimos en este fregao y hemos cargado cincuenta y tantas páginas de ilusión, ganas y conocimiento libre. Hablo en nombre de los cuatro cuando digo que es un verdadero orgullo publicar el *Elementals* y que confiamos en que lo disfrutéis al menos tanto como hemos disfrutado nosotros preparándolo.

Y que sean, por lo menos, otros cuatro.

Índice de contenidos

De puertos y firewalls

Publicado originalmente el 19 de Marzo de 2012

Debish

Científico en construcción y curioso por naturaleza. Defensor del conocimiento libre como motor del desarrollo social, tecnológico e intelectual. Debianita confeso desde tiempos de Sarge. Multitarea.

A diario establecemos conexiones, infinidad de conexiones con multitud de equipos con sede física en el lugar más recóndito que puedas imaginar. Conexiones que deben ser guiadas y compartidas entre varias aplicaciones para que todo funcione como esperamos, desde la más modesta LAN, hasta el conjunto de la World Wide Web. Veamos cómo se gestiona el tráfico de datos entre nuestras máquinas y las del resto del mundo.

¿Qué es un puerto de red?

> Un puerto de red es una **interfaz** no física mediante la cual dos máquinas intercambian datos a través de un servicio concreto. Según el modelo **OSI** (*Open System Interconnection*) su administración se corresponde con la capa 4 (transporte).

Cada puerto debe estar identificado por un número que lo hace único y permite asociarlo a un servicio determinado, de tal forma que las aplicaciones puedan saber qué tipo de información encontrarán en él. Gracias a esta identificación unívoca, es posible la **multiplexación** (en su acepción correspondiente a redes y modelo OSI), o lo que es lo mismo, la posibilidad de enviar información desde varias aplicaciones de forma simultánea a través de una conexión.

El número de puerto se indica mediante una palabra (cadena finita de bits) de 16 bits, por lo que existen $2^{16} = 65535$ puertos diferentes. Aunque en principio podemos utilizar cualquiera de ellos para cualquier servicio, la **IANA**[1] (*Internet Asigned Names Authority*) establece una relación estándar puerto-servicio, en virtud de la cual clasificamos los puertos de red en:

- **Puertos bien conocidos:** los inferiores al 1024. Suelen estar reservados para procesos del sistema y como su nombre indica son utilizados por protocolos “bien conocidos” como pueden ser https, ftp, ssh, etc. Su utilización requiere permisos de administrador. Suelen ir del lado del servidor.

1 En Debian, podéis echar un vistazo al fichero */etc/services* para ver una relación de los puertos y servicios a los que se encuentran asociados.

- **Puertos registrados:** Los comprendidos entre el 1024 y el 49151. Son de libre aplicación y además existe un listado en la IANA que especifica que protocolo usa cada uno de ellos.

- **Puertos dinámicos o privados:** del 49152 al 65535. Son puertos efímeros o de uso temporal. Normalmente son utilizados por la máquina cliente.

Puerto	Servicio
21	FTP
22	SSH
25	SMTP
53	DNS
80	HTTP
110	POP3
143	IMAP
443	HTTPS
993	IMAP SSL
995	POP SSL

Tabla1.- Puertos predeterminados para algunos de los servicios más utilizados.

¿Cómo puedo saber qué puertos tengo abiertos en mi máquina?

A pesar de que existen diversas aplicaciones on-line que escanean los puertos más comunes de tu máquina para comprobar si están abiertos, la mejor opción es utilizar una herramienta propia. En sistemas GNU/linux contamos con *netstat*:

```
user@machine:~ $ netstat -tuna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp 0 0 127.0.0.1:25 0.0.0.0:* LISTEN
tcp 0 0 0.0.0.0:46479 0.0.0.0:* LISTEN
tcp 0 0 0.0.0.0:111 0.0.0.0:* LISTEN
tcp 0 0 192.168.1.162:32795 74.125.230.228:80 ESTABLISHED
tcp6 0 0 ::1:25 :::* LISTEN
tcp6 0 0 :::39940 :::* LISTEN
tcp6 0 0 :::111 :::* LISTEN
udp 0 0 0.0.0.0:51873 0.0.0.0:*
udp 0 0 0.0.0.0:800 0.0.0.0:*
udp6 0 0 :::800 :::*
udp6 0 0 :::111 :::*
udp6 0 0 :::5353 :::*
udp6 0 0 :::53554 :::*
udp6 0 0 :::54643 :::*
```

Los parámetros *-t* y *-u* indican a netstat que muestre las conexiones realizadas mediante los protocolos **TCP** y **UDP** respectivamente, podéis filtrar la salida a vuestro antojo (muy recomendable echar un ojo al man de netstat).

Gestión de puertos: Iptables y ufw

Como ya sabéis, **netfilter** es el framework encargado de manipular y procesar paquetes de red en el kernel linux. E **iptables** una de las herramientas más potentes construídas sobre dicho framework, un firewall que permite configurar las tablas, cadenas y reglas de netfilter. Por tanto, la mejor manera de gestionar el tráfico de paquetes desde y hacia nuestra máquina es hacer uso de la mencionada herramienta.

Sin embargo, la configuración de iptables se puede complicar más de la cuenta, así que optaremos por utilizar otra aplicación que nos permita hacerlo de forma más sencilla. Me refiero a **ufw**, un sencillo y completo firewall que nos ayudará en la tarea y que además dispone de interfaz gráfica (**gufw**). Podéis encontrarlo sin problema en los repositorios Debian.

```
root@machine:~ # aptitude install ufw
```

Como cada máquina es un mundo y además cumple una serie de funciones, es imposible dar una receta universal a seguir al pie de la letra. Sin embargo, una buena forma de proteger nuestras máquinas es cerrar de forma predeterminada todos los puertos y luego abrir tan sólo aquellos que necesitemos.

Veamos cómo hacerlo:

1) Arrancamos el firewall

```
root@machine:~ # ufw enable
```

2) Cerramos todos los puertos de forma predeterminada

```
root@machine:~ # ufw default deny
```

3) Añadimos reglas para abrir los puertos que nos interesen. Se puede indicar el nombre del servicio o bien del puerto:

```
root@machine:~ # ufw allow ssh
root@machine:~ # ufw allow 21
```

Si en algún momento queremos eliminar alguna de las reglas creadas basta con teclear:

```
root@machine:~ # ufw delete allow ssh
```

Además, podemos ver el estado del firewall mediante:

```
root@machine:~ # ufw status
```

Y activar un registro de logs, que se almacenarán en */var/log/messages*, tecleando:

```
root@machine:~ # ufw logging on
```

Por último, decir que también se podría haber realizado el proceso inverso, esto es, abrir todos los puertos por defecto (idem que el punto 2, pero con el parámetro *allow*) e ir cerrando uno por uno (idem que el punto 3, pero indicando *deny*). Si echáis un ojo al manual veréis que también es posible permitir o denegar las conexiones desde una determinada IP o rango de IP's, permitir el acceso a una IP a un sólo servicio y casi tantas combinaciones en forma de reglas como se os ocurran.

```
root@machine:~ # ufw allow from 192.168.1.0/24
root@machine:~ # ufw allow from 192.168.1.0/24 to any app ssh
```

Y eso es todo, como veis ha sido un artículo bastante generalista en el que se ha tratado lo más básico de entre lo básico. A ver si con esta pequeña introducción os animáis a indagar más en el apasionante mundo de las redes.

Nmap: escáner de puertos

Publicado originalmente el 8 de Octubre de 2012

Debish

Científico en construcción y curioso por naturaleza. Defensor del conocimiento libre como motor del desarrollo social, tecnológico e intelectual. Debianita confeso desde tiempos de Sarge. Multitarea.

Siguiendo con la estructura del artículo anterior hablaremos de nmap. Nuevamente, a nivel básico, para que sirva de recordatorio a quienes ya lo conocían y de introducción a los más noveles. En este caso, la documentación y cantidad de opciones de uso son infinitas, así que para todo lo que no cabe en esta entrada recomiendo usar la guía de referencia de nmap.

¿Y por qué nmap?

Porque no sólo es una herramienta fundamental para los profesionales de la seguridad en sus procedimientos de *data gathering* sino que además puede ser de tremenda utilidad para el usuario en multitud de ocasiones. Puede servirte para ver quien está conectado a tu red wifi, securizar tu servidor, detectar los nodos de una red y mil y una cosas más. Su función más extendida, no obstante, suele ser la de comprobar qué puertos de una máquina están abiertos con la finalidad de **regular su tráfico** y detectar posibles **vulnerabilidades** asociadas a las aplicaciones trabajando a través de ellos.

¿Qué es nmap?

Nmap es una herramienta para la exploración de redes y, de forma idónea, la realización de auditorías de seguridad. Se trata de un software desarrollado para **escanear redes completas**, aunque funciona sin problemas contra un servidor concreto. Nmap rastrea los puertos de la máquina o máquinas en cuestión y establece si un puerto está abierto, cerrado o protegido por un cortafuegos. Así, es capaz de identificar máquinas dentro de una red, determinar qué servicios utiliza dicha máquina, definir cuál es su sistema operativo e incluso devolver cierta información sobre el hardware de la máquina.

¿Cómo funciona?

A grandes rasgos, de dos formas: **enviando paquetes** o **realizando una llamada de conexión** (***connect system call***). Una vez hecho esto, Nmap es capaz de distinguir entre seis estados diferentes para cada puerto:

- ***Abierto (open)***: quiere decir que hay una aplicación aceptando conexiones TCP, **datagramas** UDP o asociaciones **SCTP** en el puerto.
- ***Cerrado (closed)***: el puerto es accesible pero no existe ninguna aplicación escuchando en él.
- ***Filtrado (filtered)***: el paquete que se ha enviado ha sido filtrado por un firewall, reglas del router, etc y nmap no puede determinar si está abierto o no.
- ***Sin filtrar (unfiltered)***: quiere decir que el puerto es accesible pero nmap no es capaz de determinar si está abierto o cerrado. Este estado sólo lo devuelve el tipo de escaneo ACK (lo veremos más adelante).
- ***Open | filtered – closed | filtered***: nmap no es capaz de definir si el puerto está abierto/cerrado o filtrado. Ocurre cuando los puertos abiertos no generan una respuesta.

Una sintaxis general simplificada de la orden sería:

```
user@machine:~$ nmap (opciones_escaneo) (máquina_a_escanear)
```

Aunque en función del tipo de escaneo, es posible que se requieran privilegios de superusuario.

Tipos de escaneo

Existen 13 tipos de escaneo, veamos algunos de ellos:

Escaneo TCP SYN: es el predeterminado y probablemente el más utilizado. También el más rápido y discreto, ya que no llega a completar una conexión TCP. Su funcionamiento consiste en enviar un paquete SYN y esperar una respuesta por parte del servidor: si se recibe un SYN/ACK el puerto está abierto (si lo recibe sin el flag ACK también lo considera así), si se recibe un RST (reset) está cerrado y si no se recibe respuesta tras varios intentos se considera filtrado.

Comando:

```
root@machine:~# nmap -sS (máquina)
```

Escaneo TCP connect: el predeterminado cuando no tenemos acceso root y por tanto no podemos enviar paquetes en crudo. Nmap solicita al sistema que establezca una conexión con la máquina objetivo a través del puerto elegido mediante una llamada de tipo connect. Se trata de una opción menos eficiente que TCP SYN, ya que requiere más tiempo y paquetes para obtener la misma información.

Comando:

```
user@machine:~$ nmap -sT (máquina)
```

Escaneo UDP: dado que el escaneo de puertos UDP es más lento y dificultoso que el de TCP, muchas veces se deja de lado su auditoría. Es un error, ya que tanto DNS (puerto 53), como SNMP (puertos 161/162) y DHCP (puertos 67/68) -servidor/cliente- corren sobre éste. El escaneo UDP funciona mediante el envío de un paquete UDP a los puertos seleccionados, de tal forma que si se devuelve un error "ICMP unreachable" el puerto se considera cerrado o filtrado (en función del código de error) mientras que si hay respuesta mediante un paquete UDP se considera abierto.

Comando:

```
root@machine:~# nmap -sU (máquina)
```

Escaneo SCTP INIT: este tipo de escaneo se establece como alternativa a los TCP y UDP y sería el equivalente a un escaneo TCP SYN en el ámbito **SCTP**. Se trata de un tipo de escaneo rápido y que distingue bien entre los estados abierto, cerrado y filtrado. Además, es muy poco intrusivo, ya que no completa la asociación STCP sino que envía un paquete INIT como si se pretendiera abrir una conexión y espera la respuesta: si recibe un INIT-ACK, el puerto está abierto, mientras que si recibe un ABORT el puerto está cerrado. En caso de no recibir respuesta tras varios intentos, el puerto se marca como filtrado.

Comando:

```
root@machine:~# nmap -sY (máquina)
```

Escaneo TCP personalizado:esta modalidad tiene como finalidad permitir que el usuario defina su análisis a la carta, especificando las TCP flags a utilizar (URG, ACK, PSH, RST, SYN, FIN) y el tipo de escaneo TCP (-sF, -sA).

Comando:

```
root@machine:~# nmap -sF/sA --scanflags URG
```

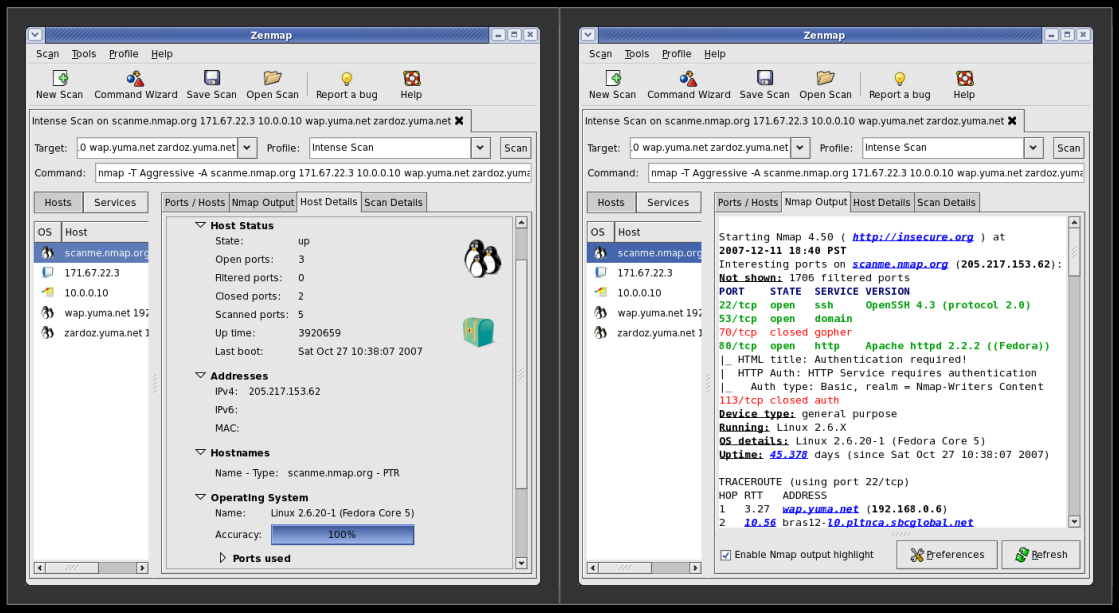
(Por ejemplo)

Recuerda que existe una **completísima guía sobre nmap** a la que puedes acudir para cada caso en concreto. Este artículo tan sólo recoge algunos de los usos más comunes.

Trabajando en modo gráfico: zenmap

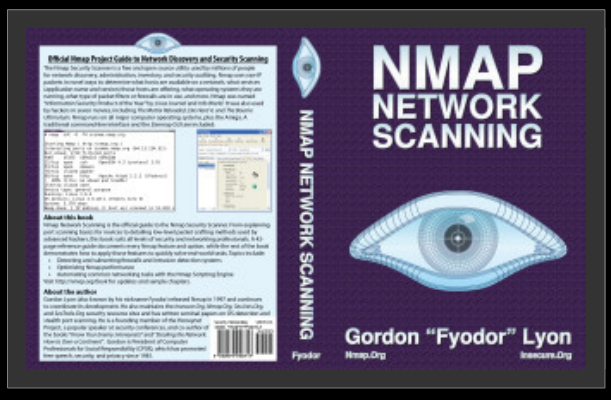
Aunque no es lo habitual en los usuarios de nmap (normalmente con conocimientos avanzados y más que acostumbrados a trabajar en la terminal), cabe la posibilidad de que por una u otra razón necesiten o prefieran una interfaz gráfica para trabajar con nmap. Para esos casos existe **Zenmap** que recoge toda la potencia de nmap en una GUI muy intuitiva, completa y útil tanto para análisis sencillos como aquellos más complejos o personalizados.

Aquí, un par de capturas de pantalla para que os hagáis una idea de su aspecto, aunque lo más recomendable es que lo probéis vosotros mismos:



Más información

Si queréis más información sobre el uso de nmap, ejemplos prácticos, sugerencias sobre cómo afrontar casos reales mediante nmap, etc os recomiendo adquirir su libro:



Entendiendo la multiarquitectura en Debian GNU/Linux con Citrix

Publicado originalmente el 12 de Marzo de 2013

n1mh (aka Diego Martínez Castañeda)

Linux user, debian user, blogger, podcaster, geek y escritor sin ideas. Asturiano en Mérida. Nadador contra corriente.

Me doy perfecta cuenta de que el título es, cuando menos, confuso. Y está hecho aposta, debo añadir. El motivo: que este sencillo tutorial no da para comprender la multiarquitectura que han implantado en Debian, pero un poco sí que se acerca. Además, he conseguido aplicarlo para un programa concreto y muy orientado al mundo empresarial, el cliente Citrix pero me falta el que considero el caballo de batalla del ambiente doméstico, Skype.

> Quien quiera ampliar sus conocimientos sobre la multiarquitectura, en el wiki de Debian tiene la mejor de las guías.

Este cambio de arquitectura ha venido propiciado porque he jubilado a mi vieja estación de trabajo (¡gracias por todo! No vuelvas ;)) y me han dado otro ordenador, más moderno, con un procesador de 64 bits. Por supuesto, he instalado Debian para 64 bits (`amd64`), configurado y actualizado todo desde cero, como corresponde. Con algunos programas tuve más lío (`firefox+flash+java`), simplemente por no recordar que había cambiado de arquitectura pero, una vez unificados criterios, todo funcionó como se esperaba.

Hasta llegar al cliente ICA.

Como es de suponer, se parte con un equipo actualizado correctamente (en mi caso, con `sid`), sin paquetes atascados ni cosas raras.

Lo primero es añadir la segunda arquitectura a `dpkg` y actualizar la base de datos de paquetes.

```
$ sudo dpkg --add-architecture i386
$ sudo aptitude update
```

Después instalamos la dependencia más peliaguda de Citrix, `ia32-libs`. Ahora nos tiene que dejar, sin más. Como estos paquetes sólo están disponibles para una arquitectura, no hay que especificar nada.

```
$ sudo aptitude install ia32-libs ia32-libs-gtk ia32-libs-gtk-i386 ia32-libs-i386
```

Seguimos instalando dependencias.

```
$ sudo aptitude install libmotif4:i386
```

En esta ocasión hay que especificar la arquitectura para que no instale la de `amd64`, que sería la opción por defecto. En caso de que esto suceda, la ejecución del programa dará el siguiente error:

```
/opt/Citrix/ICAClient/wfcmgr: error while loading shared libraries: libXm.so.4: wrong ELF class: ELFCLASS64
```

Y, por último, instalamos nspluginwrapper para 64 bits. Al igual que con que `libmotfi4`, la instalación de la arquitectura incorrecta se traduciría en un bonito error. Este paquete, además, he tenido que bajarlo directamente de la web de Debian porque sólo está disponible para squeeze.

```
$ sudo dpkg -i ./nspluginwrapper_1.3.0-1_amd64.deb
```

Cumplidas las dependencias (¡por fin!), ya podemos instalar el paquete oficial de Citrix para 64 bits, sin que reviente por algún sitio:

```
$ sudo dpkg -i icaclient_12.1.0_amd64.deb
```

Es probable que de un error durante la instalación, con salida de error 2:

```
dpkg: error processing icaclient (--configure):
el subproceso instalado el script post-installation devolvió el código de salida de error 2
```

Si buscamos el error en el script de post-instalación, veremos que el cliente ICA, paradójicamente, no está preparado para reconocer los sistemas de esta arquitectura. Se puede obviar puesto que el software está instalado y ya funciona pero, aparecerá en todos las actualizaciones posteriores, como un error.

Por eso recomiendo perder dos minutos y, siguiendo las explicaciones de la Inet Survival Guide, editar el fichero `DEBIAN/postinst` y comentar la línea 2670, es decir, dejarla de esta guisa:

```
#exit 2
```

Una vez compilado el paquete, se instala de nuevo, la salida será limpia esta vez.

En este punto, sólo queda iniciar `synapse` y buscar `Citrix Receiver` , paladeando el momento.

Instalar Skype en debian multiarquitectura

Publicado originalmente el 7 de Mayo de 2013

n1mh (aka Diego Martínez Castañeda)

Linux user, debian user, blogger, podcaster, geek y escritor sin ideas. Asturiano en Mérida. Nadador contra corriente.

Si, lo sé, los títulos empiezan a rayar lo imposible pero prometo estarme tranquilo una temporada.

Una de las partes, a mi juicio, más interesantes del anuncio de la liberación de Debian GNU/Linux 7, «wheezy», es la inclusión oficial de la multiarquitectura. La posibilidad de instalar un software hecho a medida para una determinada arquitectura en otra y sin que esto suponga un quebradero de cabeza (como venía siendo últimamente), es un avance enorme.

Hace algún tiempo publiqué una entrada dedicada a este tema, Entendiendo la multiarquitectura en Debian GNU/Linux con Citrix y ya en la primera línea dejaba claro que el cliente ICA es para un entorno corporativo y que la verdadera prueba de fuego sería Skype. Bien, pues ha llegado el día de las pruebas.

Han coincidido varios factores. Por un lado, la última actualización ya estaba pidiendo la desinstalación de la librería `ia32-libs`; hace dos días se liberó wheezy con este tema presente en todas las notas, avisos y comentarios; y, además, hoy estoy de ese humor en que el cuerpo te pide riesgos.

Pero volvamos al tema. Lo primero de todo es hacer una pequeña actualización y limpieza del sistema antes de empezar:

```
diego@denox:~$ sudo aptitude update
diego@denox:~$ sudo aptitude full-upgrade
```

Probablemente, desinstalará varios paquetes, entre ellos la mencionada `ia32-libs` y todas sus dependencias. Nada de lágrimas, eso es lo que perseguimos. A continuación, añadimos la segunda arquitectura a `dpkg`, en mi caso, `i386`:

```
diego@denox:~$ sudo dpkg --add-architecture i386
```

El resto es sencillo. Podría usar `dpkg` para comprobar las dependencias pero ya he dicho que hoy estoy guerrero, así que instalo directamente skype (bajado de la web oficial):

```
diego@denox:~$ sudo dpkg -i skype-debian_4.1.0.20-1_i386.deb
Seleccionando el paquete skype previamente no seleccionado.
(Leyendo la base de datos ... 126982 ficheros o directorios instalados
actualmente.)
Desempaquetando skype (de skype-debian_4.1.0.20-1_i386.deb) ...
dpkg: problemas de dependencias impiden la configuración de skype:
skype depende de libqtwebkit4 (>= 2.1.0~2011week13).
dpkg: error al procesar skype (--install):
problemas de dependencias - se deja sin configurar
Procesando disparadores para mime-support ...
Procesando disparadores para gnome-menus ...
Procesando disparadores para desktop-file-utils ...
Se encontraron errores al procesar:
skype
Nos comenta que depende de libqtwebkit4 y, al tratar de instalar dicha
librería sin indicar la arquitectura, nos corrige:

diego@denox:~$ sudo aptitude install libqtwebkit4
Se configurarán los siguientes paquetes que están ahora parcialmente
instalados:
skype:i386{b}
No se instalará, actualizará o eliminará ningún paquete.
0 paquetes actualizados, 0 nuevos instalados, 0 para eliminar y 0 sin
actualizar.
Necesito descargar 0 B de ficheros. Después de desempaquetar se usarán
0 B.
No se satisfacen las dependencias de los siguientes paquetes:
skype:i386 : Depende: libqtwebkit4:i386 (>= 2.1.0~2011week13) pero no
será instalado.
Las acciones siguientes resolverán estas dependencias
Eliminar los paquetes siguientes:
1) skype:i386
¿Acepta esta solución? [Y/n/q/?]
```

Hace hincapié varias veces en que tanto skype como la librería de la que depende están disponibles para la arquitectura i386, únicamente. Así pues, sólo tenemos que complacer al dpkg, indicándole la arquitectura deseada:

```
diego@denox:~$ sudo aptitude install libqtwebkit4:i386
```

```
Se instalarán los siguiente paquetes NUEVOS:
libgstreamer-plugins-base0.10-0:i386{a} libgstreamer0.10-0:i386{a}
liborc-0.4-0:i386{a} libqtwebkit4:i386
Se configurarán los siguientes paquetes que están ahora parcialmente instalados:
skype:i386
0 paquetes actualizados, 4 nuevos instalados, 0 para eliminar y 0 sin actualizar.
Necesito descargar 8.876 kB de ficheros. Después de desempaquetar se usarán 32,5 MB.
¿Quiere continuar? [Y/n/?]
```

Y pensar que tenía cierto reparo y hasta miedo...

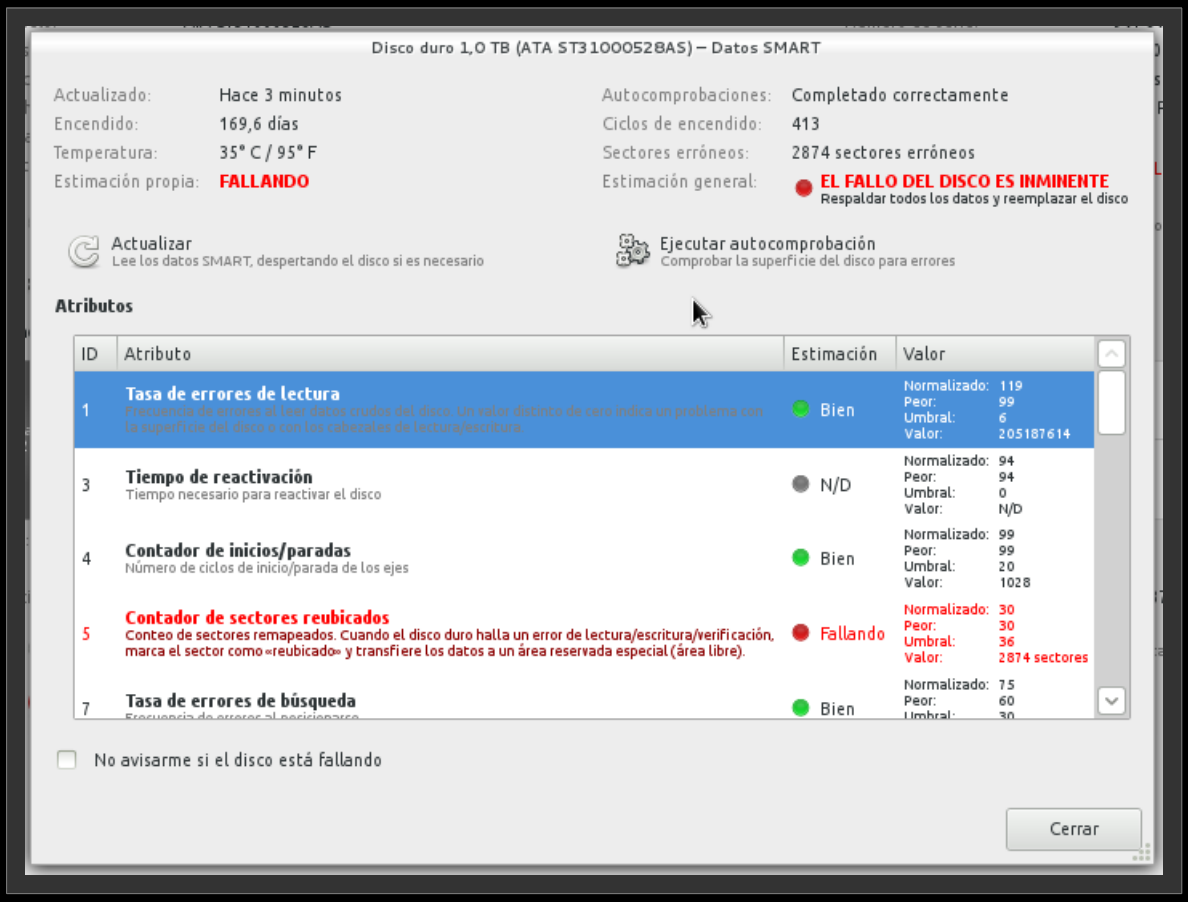
S.M.A.R.T monitoring o como evitar lágrimas y sudores con tus discos duros

Publicado originalmente el 16 de Marzo de 2012

n1mh (aka Diego Martínez Castañeda)

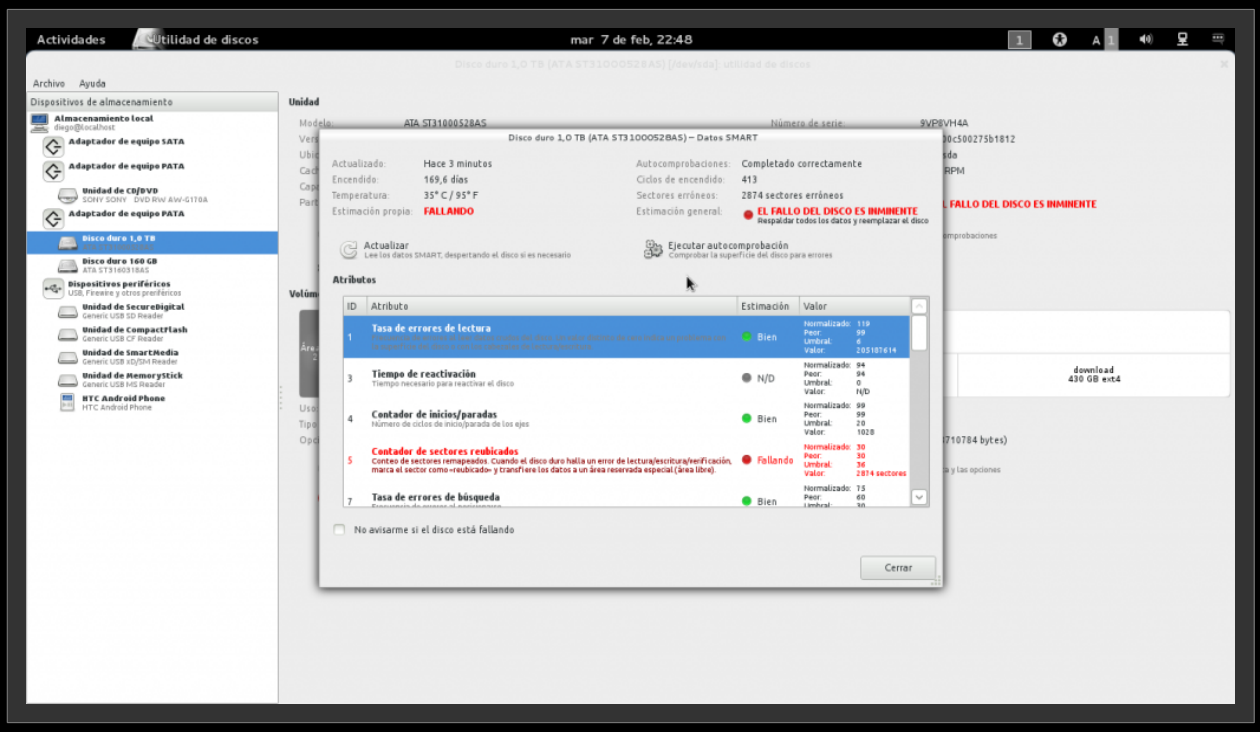
Linux user, debian user, blogger, podcaster, geek y escritor sin ideas. Asturiano en Mérida. Nadador contra corriente.

Imagínate el siguiente escenario: un día cualquiera llegas a casa por la noche y enciendes el ordenador para leer el correo y ver las novedades en debianhackers, que se rumorea que tienen nuevos fichajes. El arranque es normal, sin mensajes apocalípticos ni volcados de pila y, entonces, al iniciar la sesión en gnome, aparece un mensaje que no habías visto antes.



Es muy tranquilizador leer, en color rojo y mayúsculas, los mensajes ***FALLANDO*** y ***EL FALLO DEL DISCO ES INMINENTE*** e, inmediatamente debajo, *Respaldar todos los datos y reemplazar el disco*. Lo que realmente te pone de mala uva es saber que el dispositivo tiene menos de un año de uso, que es disco principal del ordenador y que es de un terabyte, el mayor espacio de almacenamiento del que has disfrutado nunca y donde tienes absolutamente todo. ¿Dónde se supone que voy a volcar los datos? ¿Cuántos de mis discos duros de 12 giga bytes necesitaré para respaldar a semejante monstruo? Las preguntas surgen sin cesar...

Un momento después, más tranquilo y de vuelta de la cocina con una taza de café (la noche no va a ser larga, va a ser eterna, intuyes), te tomas un tiempo en analizar el mensaje antes de dejarte llevar por el pánico.



El programa que muestra el mensaje es `palimpsest`, también conocido como la Utilidad de discos, que forma parte del paquete gnome-disk-utility y que desconocías completamente. Este programa se encarga, principalmente de realizar ciertas operaciones con discos duros y particiones (particionado, asignación del sistema de ficheros, etiquetado, montaje, desmontaje) y de la monitorización S.M.A.R.T. de los mismos.

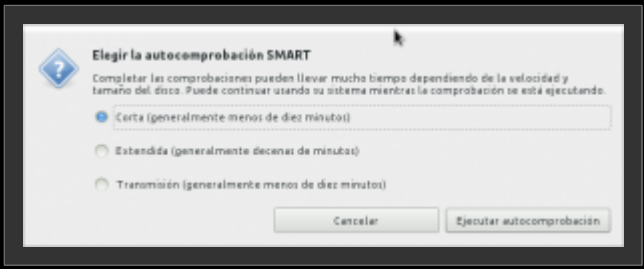
> S.M.A.R.T. es una tecnología que incorporan las placas base y los discos duros para prevenir muertes súbitas

A esas horas de la noche, leer de qué va la monitorización `smart` (voy a omitir los puntos en el resto de la entrada), mientras haces una copia de seguridad de todo fichero valioso que puebla tu disco duro, para qué negarlo, se hace cuesta arriba. Pero recuerdas (porque ya te las habías visto con ella) que es una tecnología que incorporan las placas base y los discos duros para prevenir muertes súbitas y lágrimas de desconsuelo.

En otras palabras, monitoriza ciertos valores de los discos para prevenir la aparición

de fallos o, en caso de que ocurran (shit happens), para disponer de un tiempo precioso con el que salvar tu información.

En este caso, el parámetro díscolo, ***Contador de sectores reubicados*** no te suena de nada, así que decides ver qué test te deja efectuar `palimpsest`, en la opción ***Ejecutar autocomprobación***.



A eso de las dos de la mañana, con todos los datos salvados en tres discos duros convenientemente desconectados y puestos sobre la mesa (ya no te fías ni de la electricidad), te das cuenta que da igual que realices un test u otro porque todos terminan en un par de segundos, con el mismo resultados: dos frases escritas en color rojo y letras mayúsculas que, a esas horas, ya no te parecen tan feas.

Así pues, `smart` funciona y, aunque esos mensajes apocalípticos asustan bastante por muy curtido que esté uno, finalmente reconoces que la otra opción, esa en que sabes que el dispositivo está mal en el mismo instante en que certificas su defunción, es mucho peor. La comunicación entre placa base y monitor `smart` permite comprar tiempo, que es mucho.

¿Cómo termina esta historia? Instalando el programa que la marca del disco duro ha creado (sólo para Windows, por supuesto) para volver a otros errores escritos en color rojo y mayúsculas y una invitación a enviarlo a Alemania para su sustitución por otro nuevo. Mientras tanto, instalas Debian en otro disco duro, más modesto y viejo, para seguir tirando.

Un mes después de ver por primera vez el mensaje (creo que terminaré tatuándomelo en el pecho), recibes una caja con un terabyte de espacio, limpio, nuevo, esperando particiones y datos y comienzas a instalar Debian a eso de la medianoche.

Para programar, primero entiende a tu ordenador

Publicado originalmente el 6 de Marzo de 2012

Eugenia Bahit

GLAMP Hacker y Programadora eXtrema, avocada a la docencia en el ámbito de la Ingeniería de Software y a la investigación de nuevas técnicas, metodologías y paradigmas relacionadas con la seguridad de aplicaciones. Miembro de la Free Software Foundation y The Linux Foundation.

¿De verdad crees que conoces con exactitud la arquitectura de tu ordenador? Si quieres programar, lo mejor que puedes hacer, es conocer con precisión, como está formado tu ordenador y que función exacta cumple cada componente. En este artículo, intentaremos "destripar" virtualmente nuestro ordenador, para poder entenderlo, ayudándonos del comando `lshw`, al cual podremos acceder como super-usuario.

Arquitectura de computadoras: conociendo el hardware

Un ordenador, con respecto al hardware, se encuentra compuesto por una serie de dispositivos, clasificados según la función que éstos desempeñen. Dicha clasificación, se compone de:

- Dispositivos de entrada
- Dispositivos de salida
- Dispositivos de comunicación
- Dispositivos de almacenamiento
- Dispositivos de cómputo

```
root@cocochito:~# lshw -short
```

```
Bus info          Device      Class          Description
=========================================================
                              system         To Be Filled By O.E.M.
                              bus            775i65G.
                              memory         64KiB BIOS
cpu@0                         processor      Intel(R) Celeron(R) CPU 2.80GHz
                              memory         16KiB L1 cache
                              memory         256KiB L2 cache
                              memory         2GiB System Memory
                              memory         1GiB DIMM SDRAM Synchronous
                              memory         1GiB DIMM SDRAM Synchronous
pci@0000:00:00.0              bridge         82865G/PE/P DRAM Controller/Host-Hub Interface
pci@0000:00:01.0              bridge         82865G/PE/P PCI to AGP Controller
pci@0000:01:00.0              display        NV34 [GeForce FX 5500]
pci@0000:00:06.0              generic        82865G/PE/P Processor to I/O Memory Interface
pci@0000:00:1d.0              bus            82801EB/ER (ICH5/ICH5R) USB UHCI Controller #1
pci@0000:00:1d.1              bus            82801EB/ER (ICH5/ICH5R) USB UHCI Controller #2
pci@0000:00:1d.2              bus            82801EB/ER (ICH5/ICH5R) USB UHCI Controller #3
pci@0000:00:1d.3              bus            82801EB/ER (ICH5/ICH5R) USB UHCI Controller #4
pci@0000:00:1d.7              bus            82801EB/ER (ICH5/ICH5R) USB2 EHCI Controller
pci@0000:00:1e.0              bridge         82801 PCI Bridge
pci@0000:02:00.0  wlan0       network        RT2561/RT61 802.11g PCI
pci@0000:02:02.0              communication  SM56 Data Fax Modem
pci@0000:02:05.0  eth0        network        RTL-8139/8139C/8139C+
pci@0000:00:1f.0              bridge         82801EB/ER (ICH5/ICH5R) LPC Interface Bridge
pci@0000:00:1f.1  scsi1       storage        82801EB/ER (ICH5/ICH5R) IDE Controller
scsi@1:0.0.0      /dev/cdrom  disk           DVDRAM GSA-H42N
scsi@1:0.1.0      /dev/sda    disk           122GB Maxtor 6Y120L0
scsi@1:0.1.0,1    /dev/sda1   volume         109GiB EXT4 volume
scsi@1:0.1.0,2    /dev/sda2   volume         4805MiB Extended partition
                  /dev/sda5   volume         4805MiB Linux swap / Solaris partition
pci@0000:00:1f.3              bus            82801EB/ER (ICH5/ICH5R) SMBus Controller
pci@0000:00:1f.5              multimedia     82801EB/ER (ICH5/ICH5R) AC'97 Audio Controller
usb@5:1.1         scsi2       storage
scsi@2:0.0.0      /dev/sdb    disk           SCSI Disk
```

Los **dispositivos de entrada** son todos aquellos que permiten la entrada de datos a un ordenador. Estos dispositivos, son los que permiten al usuario interactuar con el ordenador. Ejemplos: teclado, mouse (ratón), micrófono, webcam, scanner, etc.

Los **dispositivos de salida**, son todos aquellos que permiten mostrar la información procesada por el ordenador. Ejemplos: monitor, impresora, auriculares, altavoces, etc.

Los **dispositivos de comunicación** son aquellos que permiten la comunicación entre dos o más ordenadores. Ejemplos: modem, router, placa de red, bluetooth, etc.

Los **dispositivos de almacenamiento**, son todos aquellos que permiten almacenar datos en el ordenador. Ejemplos: disco duro, pendrive, disket, CD, DVD, etc.

Los **dispositivos de cómputo**, son aquellos encargados de realizar las operaciones de control necesarias, sobre el resto de los dispositivos del ordenador. Estos dispositivos, se encuentran disponibles, en todos los ordenadores, y los mismos se describen a continuación.

CPU

La **CPU** (Central Processing Unit – Unidad Central de Procesamiento), también llamada **procesador** o **microprocesador**, es un circuito microscópico que interpreta y ejecuta instrucciones. La CPU se ocupa del control y del proceso de datos en las computadoras. Generalmente, la CPU es un microprocesador fabricado en un chip, un único trozo de silicio que contiene millones de componentes electrónicos. El microprocesador de la CPU está formado por una unidad aritmético-lógica que realiza cálculos y comparaciones y toma decisiones lógicas (determinando si una afirmación es cierta o falsa mediante las reglas del álgebra de Boole. Para aceptar órdenes del usuario, acceder a los datos y presentar los resultados, la CPU se comunica a través de un conjunto de circuitos o conexiones llamado bus. El bus conecta la CPU a los dispositivos de almacenamiento (por ejemplo un disco duro), los dispositivos de entrada (por ejemplo un teclado o un mouse) y los dispositivos de salida (por ejemplo un monitor o una impresora).

```
root@cocochito:~# lshw -C bus -short
H/W path              Device      Class          Description
============================================================
/0                                bus            775i65G.
/0/100/1d                         bus             82801EB/ER (ICH5/ICH5R) USB UHCI Controller
#1
/0/100/1d.1                       bus             82801EB/ER (ICH5/ICH5R) USB UHCI Controller
#2
/0/100/1d.2                       bus             82801EB/ER (ICH5/ICH5R) USB UHCI Controller
#3
/0/100/1d.3                       bus             82801EB/ER (ICH5/ICH5R) USB UHCI Controller
#4
/0/100/1d.7                       bus            82801EB/ER (ICH5/ICH5R) USB2 EHCI Controller
/0/100/1f.3                       bus            82801EB/ER (ICH5/ICH5R) SMBus Controller
```

Cuando se ejecuta un programa, el registro de la CPU, llamado contador de programa, lleva la cuenta de la siguiente instrucción del programa, para garantizar que las instrucciones se ejecuten en la secuencia adecuada.

La unidad de control de la CPU coordina y temporiza las funciones de la CPU, tras lo cual recupera la siguiente instrucción desde la memoria. En una secuencia típica, la CPU localiza la instrucción en el dispositivo de almacenamiento correspondiente. La instrucción viaja por el bus desde la memoria hasta la CPU, donde se almacena en el registro de instrucción. Entretanto, el contador de programa se incrementa en uno para prepararse para la siguiente instrucción.

A continuación, la instrucción actual es analizada por un decodificador, que determina lo que hará la instrucción. Cualquier dato requerido por la instrucción es recuperado desde el dispositivo de almacenamiento correspondiente y se almacena en el registro de datos de la CPU.

Luego, la CPU ejecuta la instrucción y, los resultados se almacenan en otro registro o

se copian en una dirección de memoria determinada[2].

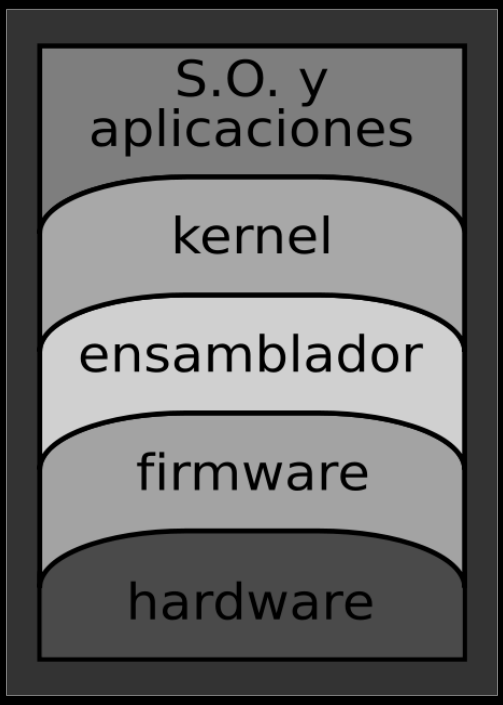
Memoria

La **memoria** es la encargada de almacenar toda la información que el ordenador se encuentra utilizando. Existen tres tipos de memoria: **memoria RAM**, **memoria ROM** y **memoria caché**.

```
root@cocochito:~# lshw -C memory -short
H/W path              Device      Class          Description
=============================================================
/0/0                              memory         64KiB BIOS
/0/4/5                            memory         16KiB L1 cache
/0/4/6                            memory         256KiB L2 cache
/0/e                              memory         2GiB System Memory
/0/e/0                            memory         1GiB DIMM SDRAM Synchronous
/0/e/1                            memory         1GiB DIMM SDRAM Synchronous
```

Listando componentes de memoria

Memoria RAM



La **memoria RAM** (Random Access Memory), es la memoria desde la cual, el procesador recibe las instrucciones y guarda los resultados. Se utiliza como memoria de trabajo para el sistema operativo, los programas y la mayoría del software. Es allí donde se cargan todas las instrucciones que ejecutan el procesador y otras unidades de cómputo. Se denominan "de acceso aleatorio" (random access) porque se puede leer o escribir en una posición de memoria con un tiempo de espera igual para cualquier posición, no siendo necesario seguir un orden para acceder a la información de la manera más rápida posible. Durante el encendido del ordenador, la rutina POST verifica que los módulos de memoria RAM estén conectados de manera correcta. En el caso que no existan o no se detecten los módulos, la mayoría de tarjetas madres (motherboard) emiten una serie de pitidos que indican la ausencia de memoria principal.

2 Fuente: http://es.wikiversity.org/wiki/Estructura_del_computador

Terminado ese proceso, la memoria BIOS[3] (memoria ROM) puede realizar un test básico sobre la memoria RAM indicando fallos mayores en la misma[4].

```
root@cocochito:# lshw -C memory -short | grep -i ram
/0/e/0                    memory         1GiB DIMM SDRAM Synchronous
/0/e/1                    memory         1GiB DIMM SDRAM Synchronous
```

Filtrando la memoria RAM

Memoria ROM

La **memoria ROM** (Read Only Memory), es permanente, ya que lo que permanece en la ROM no se pierde aunque el ordenador se apague. Su función principal es guardar información inicial que el ordenador necesita para colocarse en marcha una vez que se enciende. Permite solo la lectura de la información y no su escritura, independientemente de la presencia o no de una fuente de energía. Los datos almacenados en la ROM no se pueden modificar, o al menos no de manera rápida o fácil. Se utiliza principalmente para contener el firmware (programa que está estrechamente ligado a hardware específico, y es poco probable que requiera actualizaciones frecuentes) u otro contenido vital para el funcionamiento del dispositivo, como los programas que ponen en marcha el ordenador y realizan los diagnósticos[5].

```
root@cocochito:# lshw -C memory -short | grep -i bios
/0/0                          memory         64KiB BIOS
```

Filtrando la memoria ROM (también llamada BIOS)

Memoria Caché

La **memoria caché** es aquella que se usa como puente entre el CPU y la memoria RAM para evitar demoras en el procesamiento de los datos. Existen varios núcleos de esta memoria (denominados con la letra L y un número, por ejemplo L1). Cuanto menor el número más rápida es la memoria. Por proximidad a la CPU, es mucho más rápida que la memoria RAM y también, mucho mas pequeña.

---

3 En este caso, el término BIOS hace referencia a la memoria ROM (descripta anteriormente), y NO, al Sistema Básico de Entrada y Salida.

4 Fuente: http://es.wikipedia.org/wiki/Memoria_RAM

5 Fuente: http://es.wikipedia.org/wiki/Memoria_ROM

```
root@cocochito:~# lshw -C memory -short | grep cache
/0/4/5                          memory         16KiB L1 cache
/0/4/6                          memory         256KiB L2 cache
```

Filtrando la memoria caché

Bus de datos

El **bus de datos** (o canal de datos) es un sistema digital que transfiere datos entre los componentes de una computadora o entre computadoras. Está formado por cables o pistas en un circuito impreso, dispositivos como resistores y condensadores además de circuitos integrados.

Notas adicionales sobre el comando `lshw`

Como hemos podido notar, el comando **`lshw`** nos permite **listar el hardware de nuestro equipo**. En este artículo, hemos utilizado el comando **`lshw`** con las siguientes opciones:

- **`-short`**
  Utilizada para listar el hardware en formato compacto.
- **`-C` *`tipo_de_componente`***

  Utilizada para filtrar la salida, por el tipo de componente indicado.

Otras opciones pueden conocerse mediante **`man lshw`**.

También hemos concatenado (con **`|`**) el comando **`lshw`** con el comando **`grep` *`patrón`*** a fin de que la salida de **`lshw`** sea a la vez, filtrada por el patrón indicado. Cuando utilizamos la opción **`-i`** del comando **`grep`**, lo hicimos para que la búsqueda no distinga entre mayúsculas y minúsculas.

Entendiendo los lenguajes de programación

Publicado originalmente el 29 de Febrero de 2012

Eugenia Bahit

GLAMP Hacker y Programadora eXtrema, avocada a la docencia en el ámbito de la Ingeniería de Software y a la investigación de nuevas técnicas, metodologías y paradigmas relacionadas con la seguridad de aplicaciones. Miembro de la Free Software Foundation y The Linux Foundation.

Los lenguajes de programación, forman parte del grupo de lenguajes informáticos. Ampliamente, puede decirse que un lenguaje informático es un idioma artificial, utilizado por ordenadores, cuyo fin es transmitir información de algo a alguien.

Los **lenguajes informáticos**, pueden clasificarse en:

- lenguajes de programación (Python, PHP, Perl, C, etc.);
- lenguajes de especificación (UML);
- lenguajes de consulta (SQL);
- lenguajes de marcas (HTML, XML);
- lenguajes de transformación (XSLT);
- protocolos de comunicaciones (HTTP, FTP); entre otros.

> “un lenguaje informático es un idioma artificial utilizado por ordenadores

Mientras que algunos lenguajes informáticos como (X)HTML o CSS, han sido

diseñados para diagramar y decidir la forma en la cual la información será presentada al usuario, los lenguajes de programación, tienen como fin, **expresar órdenes e instrucciones precisas, que deben ser llevadas a cabo por una computadora para realizar una o más tareas específicas**. Se utilizan para crear programas que controlan el comportamiento físico o lógico de un ordenador. Están compuestos por una serie de símbolos, reglas sintácticas y semánticas que definen la estructura del lenguaje.

Lenguajes de Programación según su nivel de abstracción

En un primer estado de clasificación, los lenguajes de programación se dividen según su nivel de abstracción, en lenguajes de bajo nivel, lenguajes de medio nivel y lenguajes de alto nivel, dependiendo de su grado de "cercanía al hardware".

Cuanto más cercano al hardware se encuentra el lenguaje, más bajo nivel posee éste. Mientras que cuanto más acercado al usuario se encuentre, más alejado del hardware estará y, en consecuencia, de mayor nivel será el lenguaje.

Lenguajes de Programación de Bajo Nivel

Los **lenguajes de bajo nivel**, son aquellos que dependen intrínsecamente del ordenador. Aquellos programas informáticos, programados con lenguajes de bajo nivel, al ser exclusivamente dependientes del hardware, no pueden migrarse, ya que están justamente diseñados, para un hardware específico.

Existen dos tipos de lenguajes de bajo nivel: el **lenguaje máquina** y el **lenguaje ensamblador**.

El **lenguaje de máquina** (también denominado lenguaje de primera generación) es el sistema de códigos directamente interpretable por un circuito microprogramable, como el microprocesador de una computadora o el microcontrolador de un autómata . Este lenguaje está compuesto por un conjunto de instrucciones que determinan acciones a ser tomadas por la máquina. Un programa consiste en una cadena de estas instrucciones de lenguaje de máquina (más los datos). Estas instrucciones son normalmente ejecutadas en secuencia, con eventuales cambios de flujo causados por el propio programa o eventos externos. El lenguaje de máquina es específico de cada máquina o arquitectura de la máquina, aunque el conjunto de instrucciones disponibles pueda ser similar entre ellas[6].

6 Fuente: http://es.wikipedia.org/wiki/Lenguaje_m%C3%A1quina

```
8B542408 83FA0077 06B80000 0000C383
FA027706 B8010000 00C353BB 01000000
B9010000 008D0419 83FA0376 078BD98B
C84AEBF1 5BC3
```

Función en 32-bits en código de maquina x86, para calcular el enésimo número de la serie de Fibonacci[7]

Un **lenguaje ensamblador**, o **assembler** (assembly language) es un lenguaje de programación de bajo nivel para los ordenadores, microprocesadores, microcontroladores, y otros circuitos integrados programables. Implementa una representación simbólica de los códigos de máquina binarios y otras constantes necesarias para programar una arquitectura dada de CPU y constituye la representación más directa del código máquina específico para cada arquitectura legible por un programador. Esta representación es usualmente definida por el fabricante de hardware, y está basada en los mnemónicos[8] que simbolizan los pasos de procesamiento (las instrucciones), los registros del procesador, las posiciones de memoria, y otras características del lenguaje. Un lenguaje ensamblador es por lo tanto, específico a cierta arquitectura de computador física (o virtual). Esto está en contraste con la mayoría de los lenguajes de programación de alto nivel, que, idealmente son portables[9].

Este lenguaje, también es conocido como **lenguaje de segunda generación**.

```
; HOLA.ASM
; Programa clasico de ejemplo. Despliega una leyenda en pantalla.
STACK     SEGMENT STACK                   ; Segmento de pila
                  DW     64 DUP (?)      ; Define espacio en la pila
STACK     ENDS

DATA      SEGMENT                         ; Segmento de datos
SALUDO      DB    "Hola mundo!!",13,10,"$" ; Cadena
DATA      ENDS

CODE      SEGMENT                         ; Segmento de Codigo
          ASSUME CS:CODE, DS:DATA, SS:STACK

INICIO:                                   ; Punto de entrada al programa
          MOV  AX,DATA                    ; Pone direccion en AX
          MOV  DS,AX                      ; Pone la direccion en los registros
          MOV  DX,OFFSET SALUDO           ; Obtiene direccion del mensaje
          MOV  AH,09H                     ; Funcion: Visualizar cadena
          INT     21H                     ; Servicio: Funciones alto nivel DOS
          MOV  AH,4CH                     ; Funcion: Terminar
```

7 Fuente: http://en.wikipedia.org/wiki/Low-level_programming_language

8 En informática, un mnemónico es una palabra que sustituye a un código de operación (lenguaje de máquina), con lo cual resulta más fácil la programación, es de aquí de donde se aplica el concepto de lenguaje ensamblador. Fuente: http://es.wikipedia.org/wiki/Mnem%C3%B3nico

9 Fuente: http://es.wikipedia.org/wiki/Lenguaje_ensamblador

```
          INT     21H
CODE      ENDS
          END  INICIO                    ; Marca fin y define INICIO
```

Ejemplo desarrollado en lenguaje ensamblador que usa llamadas de MS-DOS (system calls) para imprimir el mensaje Hola mundo!! en pantalla. Extraído de http://homepage.mac.com/eravila/asmix862.html

(para ver la explicación detallada del ejemplo, seguir el enlace anterior)

Para ampliar la información sobre los lenguajes de bajo nivel, puede leerse el siguiente artículo de Karmany.net.

También es recomendable, leer el siguiente artículo sobre Lenguaje Ensamblador en Wikipedia.

Lenguajes de Programación de Medio Nivel

La clasificación de lenguajes de programación, mediante un nivel de abstracción medio, es bastante discutible. Personalmente sostengo sólo por dos niveles de abstracción: bajo nivel y alto nivel. Sin perjuicio de ello, se plasmarán aquí, aquellos argumentos sostenidos, por quienes aceptan este tercer nivel de clasificación.

Quienes sostienen la clasificación de lenguajes de programación medio, argumentan que éstos, **son aquellos lenguajes que se encuentran, justamente, entre los de bajo nivel y los de alto nivel, ya que poseen características que permiten interactuar directamente con el sistema**. Un ejemplo de ello, sería el lenguaje C, el cual puede trabajar (entre otras características) con direcciones de memoria. Sin embargo, dicho acceso, no es efectuado de forma directa (a través de lenguaje máquina o ensamblador), sino que requiere ser “traducido” previamente por su compilador. Por dicha razón, es que asumo a C como lenguaje de alto nivel y descarto la clasificación de lenguajes de medio nivel.

Suele colocarse como ejemplo de lenguaje de programación de medio nivel, anterior a C, a **BCPL**[10], diseñado para escribir Sistemas Operativos y Compiladores.

```
GET "libhdr"

LET start() = VALOF
{ FOR i = 1 TO 5 DO writef("fact(%n) = %i4*n", i, fact(i))
```

10 Ver manual de Referencias de BCPL

```
  RESULTIS 0
}

AND fact(n) = n=0 -> 1, n*fact(n-1)
```

Ejemplo de código BCPL para impresión de factoriales[11]

Lenguajes de Programación de Alto Nivel

Los lenguajes de alto nivel, son aquellos cuya característica principal, consiste en una estructura sintáctica y semántica legible, acorde a las capacidades cognitivas humanas. A diferencia de los lenguajes de bajo nivel, **son independientes de la arquitectura del hardware**, motivo por el cual, asumen mayor portabilidad.

Son ejemplo de lenguajes de alto nivel: Python, Perl, PHP, Ruby, Lisp, Java, Fortran, C++, C#, entre otros.

```
print "Hola Python!"
```

Archivo: hola_mundo.py . Imprime Hola Python! En pantalla

```
<?php
echo "Hola PHP!";
?>
```

Archivo: hola_mundo.php . Imprime Hola PHP! En pantalla

```
print "Hola Perl!";
```

Archivo: hola_mundo.pl . Imprime Hola Perl! En pantalla

Hola Mundo en otros lenguajes (para curiosos): www.holamundo.es

Clasificación de Lenguajes de programación, según su forma de ejecución
Según su forma de ejecución, los lenguajes de programación pueden ser: **compilados** o **interpretados**.

Los **lenguajes de programación compilados**, son lenguajes de alto nivel que requieren que las instrucciones (código fuente del programa), sean traducidas a lenguaje máquina por un compilador, a fin de generar un ejecutable del programa. Ejemplo de lenguajes compilados son Pascal, C, C++, Ada, entre otros.

11 Fuente: http://en.wikipedia.org/wiki/BCPL#Examples

```
#include

int main()
{
        printf("Hola mundo");
        return 0;
}
```

Ejemplo en C que imprimirá “Hola mundo” en pantalla tras ser compilado.

```
program Hello;
   begin
      writeln ('Hola mundo')
   end.
```

Mismo ejemplo, pero en Pascal.

Los **lenguajes interpretados**, a diferencia de los compilados, no requieren de un compilador para ser ejecutados sino de un intérprete. Un intérprete, actúa de manera casi idéntica a un compilador, con la salvedad de que ejecuta el programa directamente, sin necesidad de generar previamente un ejecutable. Ejemplo de lenguajes de programación interpretado son Python, PHP, Ruby, Lisp, entre otros.

```
(print "Hola Mundo!")
```

Ejemplo de código Lisp que imprime “Hola Mundo!” en pantalla

Es importante además, hacer notar que la mayoría de los lenguajes de programación, puede ejecutarse tanto de modo compilado como interpretado.

Cómo crear backups cifrados e incrementales en Debian y derivados con Déjà Dup

Publicado originalmente el 29 de Febrero de 2012

David Hernández

Consultor y formador en seguridad y sistemas GLAMP. Responsable del área de Hacking en apachectl.com. Mi otro blog es daboblog.com y en Twitter: @daboblog. Más info en davidhernandez.es. Debianita hasta el final de los bytes y aspirante a Alpinista.

Ya he comentado en algún podcast que por lo general, siempre suelo instalar mis máquinas Debian (escritorio) con LVM cifrando la raíz, /home y la swap (una buena "feature" de Debian, si cifras vía dm-crypt tu /home, por seguridad te obliga desde el instalador a cifrar la swap por los datos que se podrían extraer almacenados en RAM).

Como **además de mis equipos portátiles también tengo cifrados los de escritorio**, según mi criterio, no tendría mucha lógica guardar las copias de seguridad sin esa capa de cifrado y ahí es donde entra en escena Déjà Dup (el nombrecito se las trae, lo sé ;D).

También es importante comentar que tras más años de los que recuerdo usando KDE, llevo unos meses **con GNOME** (modo clásico y "flashback según equipos, menos recursos que un Shell "puro") y XFCE ya que la integración con (en adelante) Deja Dup es muy buena (este tuto lo estoy escribiendo desde Debian Testing y GNOME 3.8.4).

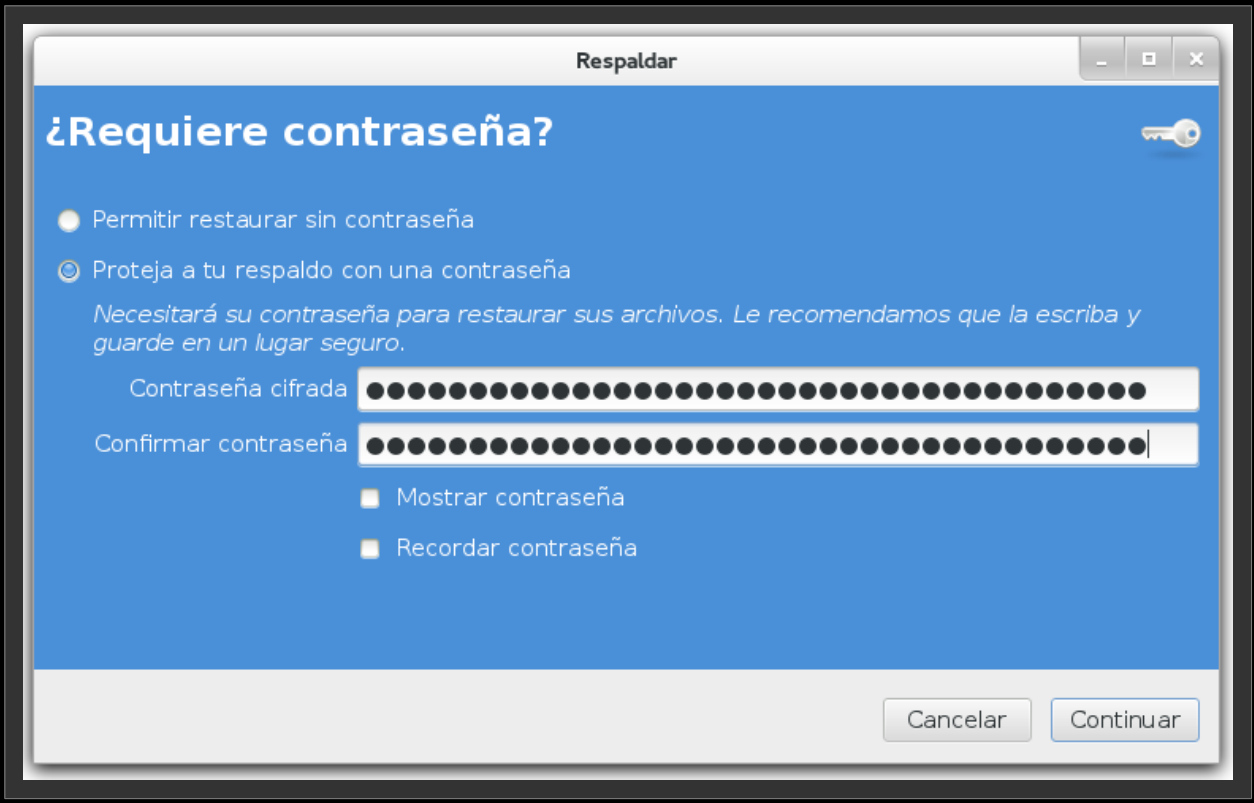
Cuando **hablo de mejor integración**, me refiero a poder hacer click con el botón del ratón y ver un menú contextual (más abajo veréis las capturas) con el texto:

“restaurar los ficheros que faltan” y es pinchar el disco, y ver restaurado el fichero o directorio sin problemas. También los indicadores de progreso de la copia de seguridad, o el aviso emergente caso de haber programado una tarea, del inicio de la copia.

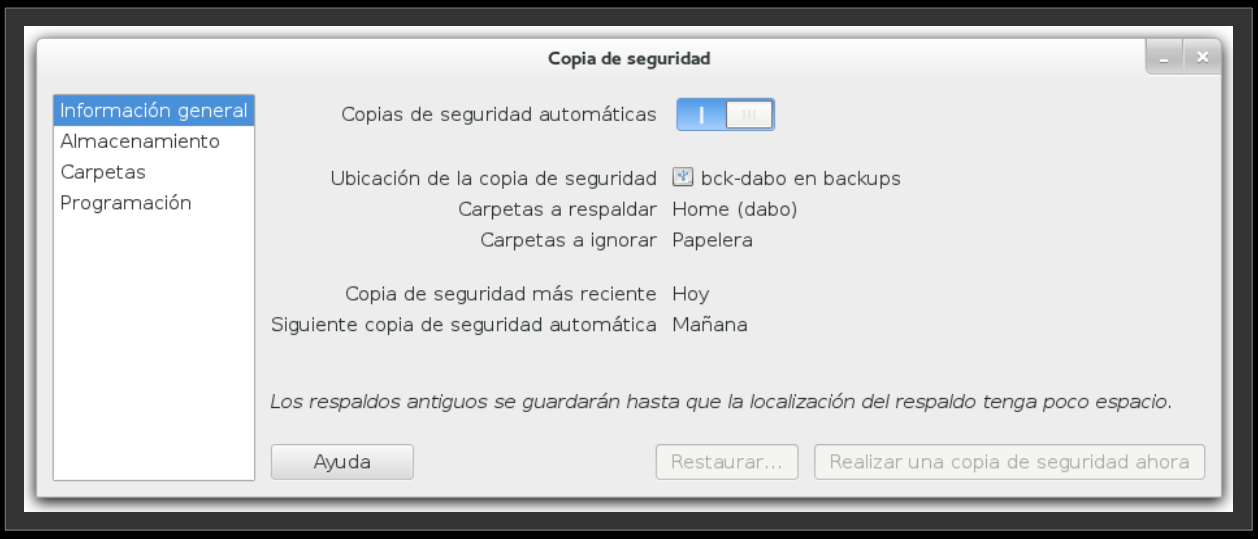
En el título pongo **Debian y derivados,** ya que es válida para otras distros como Ubuntu o Mint, pero está empaquetado en muchas más con algún ligero cambio. Por ejemplo, en Ubuntu se pueden lanzar backups contra Amazon s3 o Ubuntu One, pero las funcionalidades son similares. Para el cifrado utiliza **GPG** (GNU Privacy Guard) todo un clásico, mediante un sistema de cifrado simétrico. También trabaja con herramientas tan conocidas como Rsync, etc. **Os recomiendo ver** info más detallada de cómo funciona.

Así que ya sabéis, sólo un **aptitude install deja-dup** y empezar con esos backups. Os pongo unas cuantas capturas de pantalla para que podáis ver en modo gráfico lo que os he ido comentando. Sencillo, seguro y efectivo, concepto KISS a tope.

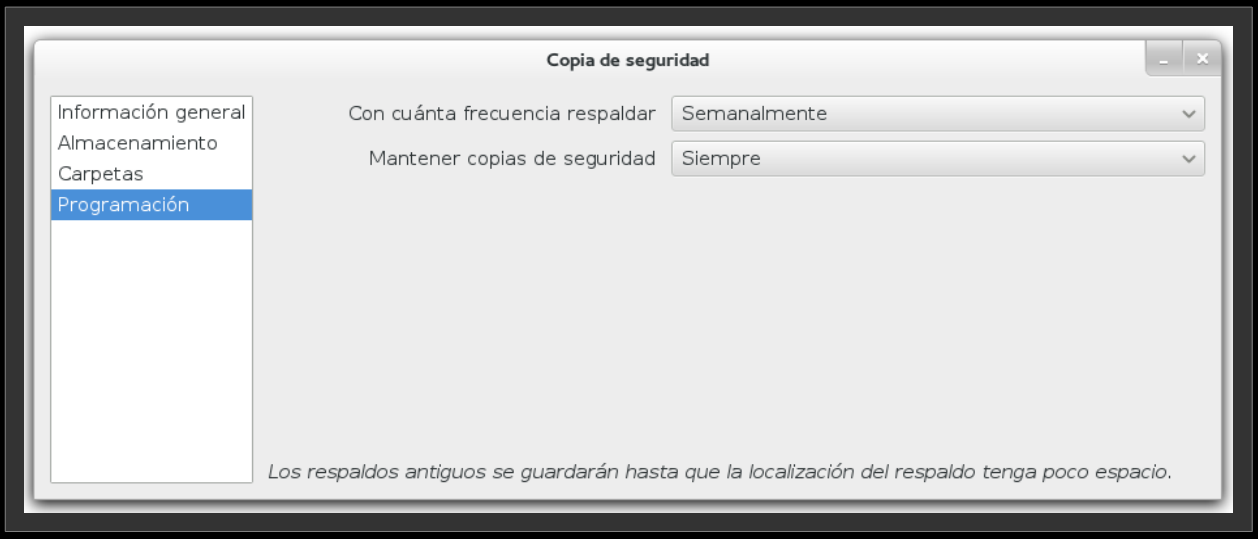
Seleccionando la **opción de cifrado** e introduciendo la frase de paso.



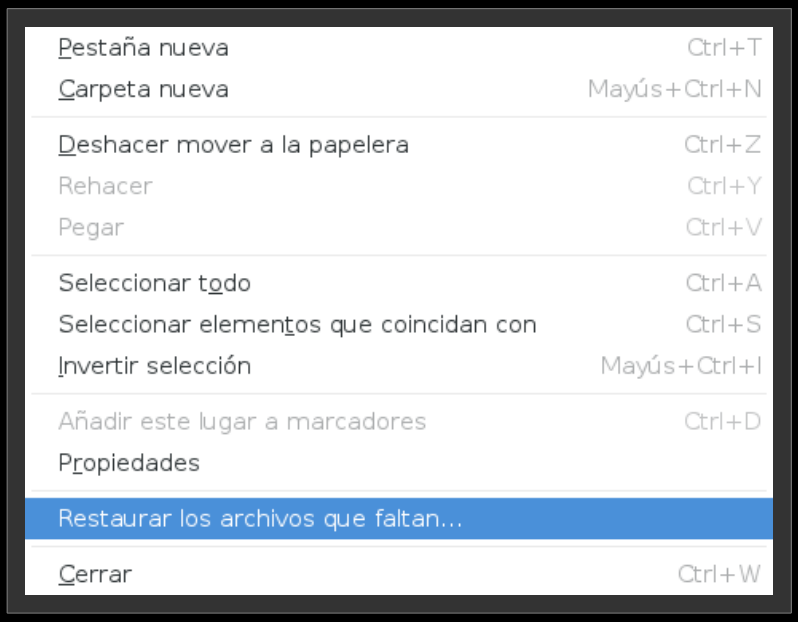
Aquí podéis **configurar opciones** de la copia de seguridad.



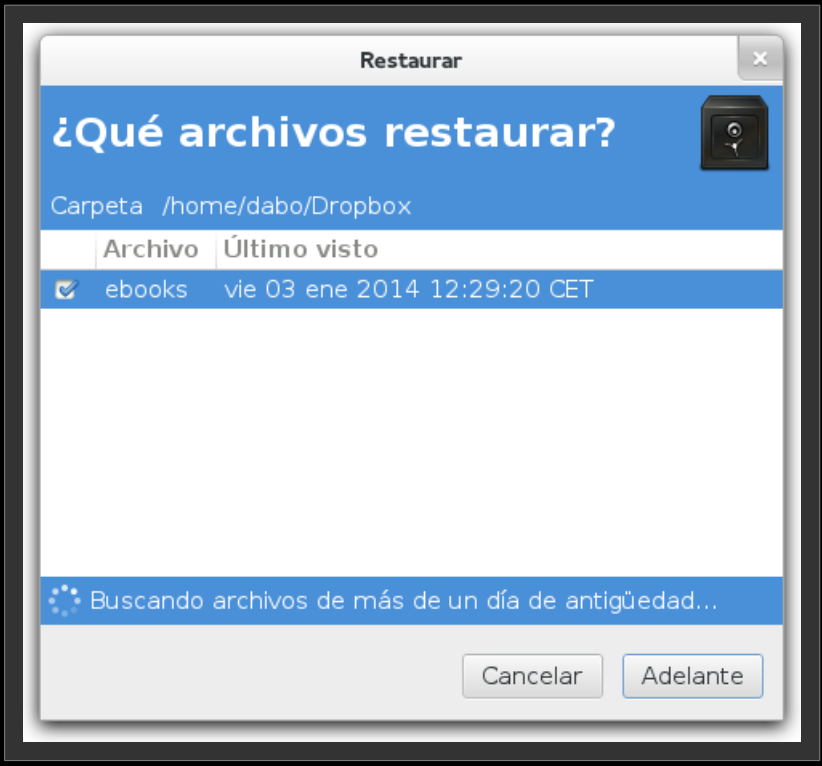
Un ejemplo con **la frecuencia** de las copias de seguridad.



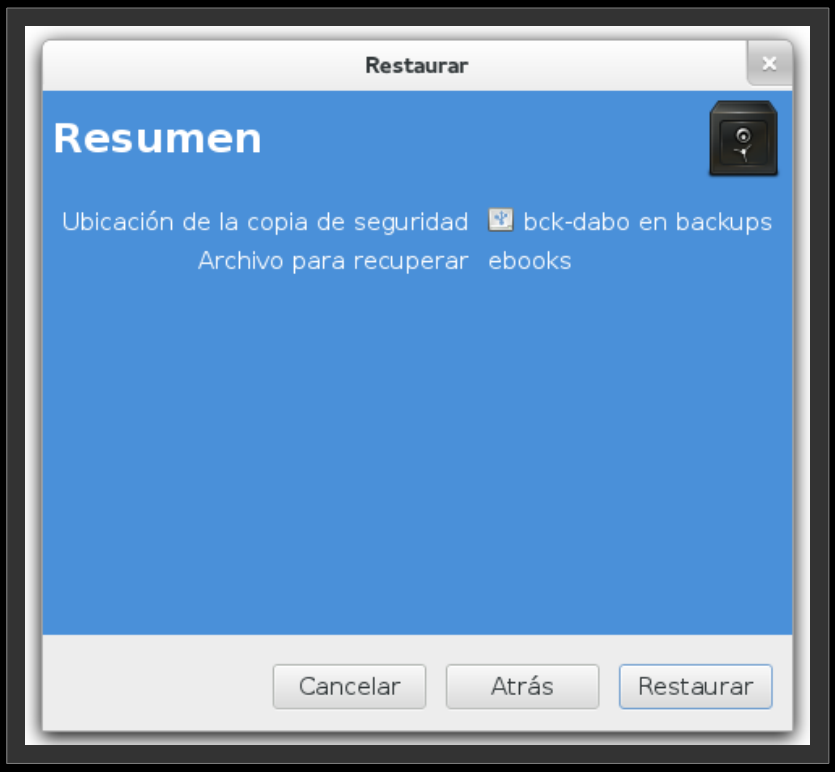
La opción que os comentaba de **restauración** y el menú contextual.



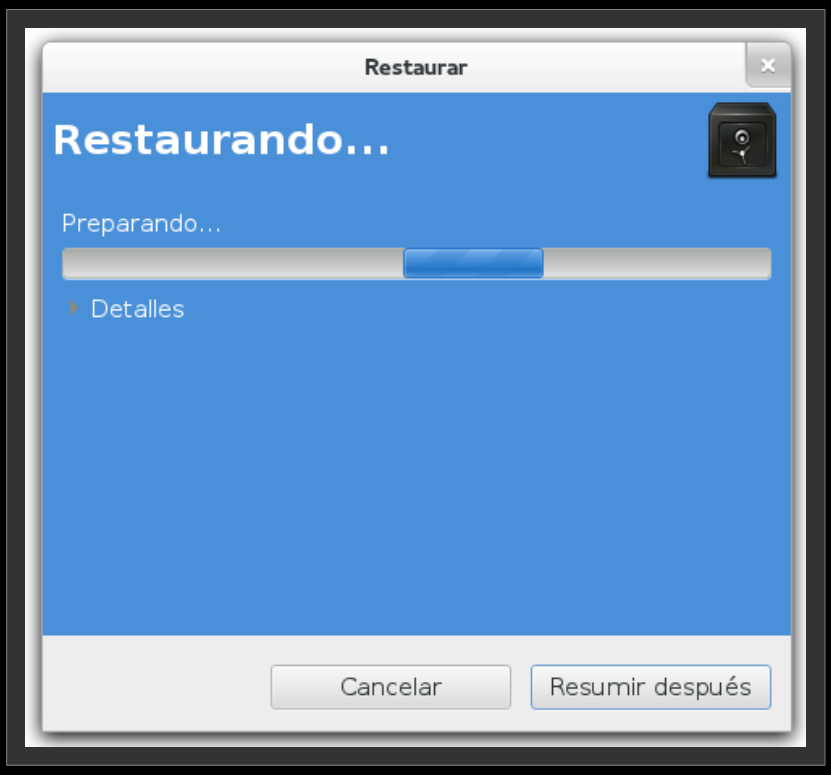
Seleccionando el **fichero a restaurar**.



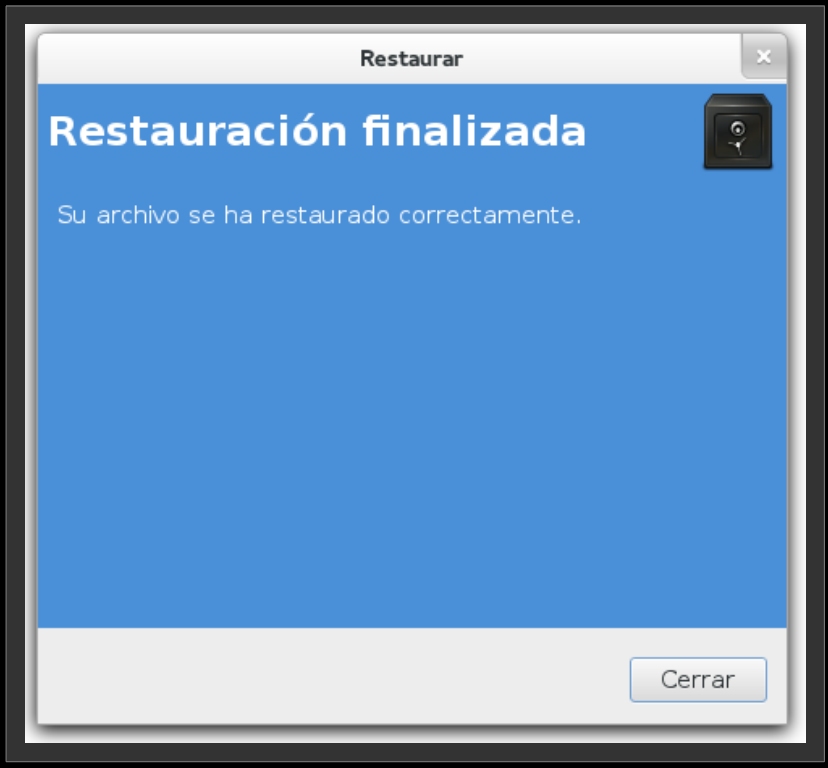
Confirmación y ubicación (coge la ruta de forma automática).



Progreso de la restauración.



Y **confirmación** de que todo ha salido ok.



Un vistazo después a los **ficheros .gpg**.

```
  52415551 dic 11 12:40 duplicity-full.20131211T112815Z.vol99.difftar.gpg
  52400951 dic 11 12:30 duplicity-full.20131211T112815Z.vol9.difftar.gpg
1382985589 dic 11 16:24 duplicity-full-signatures.20131211T112815Z.sigtar.gpg
       452 dic 12 12:35 duplicity-inc.20131211T112815Z.to.20131212T113039Z.manifest.gpg
  52407332 dic 12 12:31 duplicity-inc.20131211T112815Z.to.20131212T113039Z.vol1.difftar.gpg
  52474884 dic 12 12:34 duplicity-inc.20131211T112815Z.to.20131212T113039Z.vol2.difftar.gpg
  52471254 dic 12 12:34 duplicity-inc.20131211T112815Z.to.20131212T113039Z.vol3.difftar.gpg
  44657257 dic 12 12:35 duplicity-inc.20131211T112815Z.to.20131212T113039Z.vol4.difftar.gpg
       985 dic 22 15:48 duplicity-inc.20131212T113039Z.to.20131222T143845Z.manifest.gpg
  52492670 dic 22 15:46 duplicity-inc.20131212T113039Z.to.20131222T143845Z.vol10.difftar.gpg
  52469049 dic 22 15:46 duplicity-inc.20131212T113039Z.to.20131222T143845Z.vol11.difftar.gpg
  52456275 dic 22 15:47 duplicity-inc.20131212T113039Z.to.20131222T143845Z.vol12.difftar.gpg
  52485130 dic 22 15:47 duplicity-inc.20131212T113039Z.to.20131222T143845Z.vol13.difftar.gpg
  16500845 dic 22 15:48 duplicity-inc.20131212T113039Z.to.20131222T143845Z.vol14.difftar.gpg
  52468921 dic 22 15:39 duplicity-inc.20131212T113039Z.to.20131222T143845Z.vol1.difftar.gpg
  52406380 dic 22 15:41 duplicity-inc.20131212T113039Z.to.20131222T143845Z.vol2.difftar.gpg
  52453741 dic 22 15:44 duplicity-inc.20131212T113039Z.to.20131222T143845Z.vol3.difftar.gpg
  52478064 dic 22 15:46 duplicity-inc.20131212T113039Z.to.20131222T143845Z.vol4.difftar.gpg
  52478756 dic 22 15:46 duplicity-inc.20131212T113039Z.to.20131222T143845Z.vol5.difftar.gpg
  52465056 dic 22 15:46 duplicity-inc.20131212T113039Z.to.20131222T143845Z.vol6.difftar.gpg
  52467558 dic 22 15:46 duplicity-inc.20131212T113039Z.to.20131222T143845Z.vol7.difftar.gpg
  52452433 dic 22 15:46 duplicity-inc.20131212T113039Z.to.20131222T143845Z.vol8.difftar.gpg
  52475471 dic 22 15:46 duplicity-inc.20131212T113039Z.to.20131222T143845Z.vol9.difftar.gpg
  63931382 dic 12 12:35 duplicity-new-signatures.20131211T112815Z.to.20131212T113039Z.sigtar.gpg
  80787862 dic 22 15:48 duplicity-new-signatures.20131212T113039Z.to.20131222T143845Z.sigtar.gpg
```

Eso es todo, espero que lo encontréis útil y no dejéis de hacer vuestros backups.

Seguridad y optimización de servidores GLAMP

David Hernández

Consultor y formador en seguridad y sistemas GLAMP. Responsable del área de Hacking en apachectl.com. Mi otro blog es daboblog.com y en Twitter: @daboblog. Más info en davidhernandez.es. Debianita hasta el final de los bytes y aspirante a Alpinista.

Disclaimer

A continuación iremos repasando aspectos a tener en cuenta para configurar un servidor GLAMP (GNU/Linux, Apache, MySQL y PHP) tanto para estar preparados para soportar una carga más alta (o servir más páginas), como para hacerlo con mayor seguridad.

Todo ello partiendo de la base que la disponibilidad es seguridad, y un servidor web es uno de los entornos más expuestos que podemos administrar. Trataremos la optimización de recursos revisando directivas esenciales de Apache, MySQL, PHP, módulos de “caching”, etc, además de configuraciones por defecto que en casos de una carga elevada, o determinadas peticiones, pueden dejar inoperativo el servidor web con tráfico legítimo. Tambíen de MySQL buscando valores más potentes de cara a su optimización.

También cuestiones de seguridad (por oscuridad, medidas proactivas, preventivas y bloqueos de ciertos ataques comunes en un servidor), detección de posibles puertas traseras (rootkits), malware en aplicaciones web, intentaremos “parar” a una herramienta tan potente como Nmap sobre la que tenéis gran material en este número de la mano de debish, etc. Pero sobre todo, tener un punto de partida a para luego poder ir mejorando.

Vamos a ver algún sistema de análisis de logs y filtrado de eventos de interés, cómo se usa un IDS / IPS, algo de iptables, control de conexiones y opciones para detectar peticiones maliciosas, etc. Todo ello con ejemplos entre secciones del artículo incluyendo otros artículos publicados por mi a modo de ejemplo, con casos reales vistos en servidores en producción dentro de mi día a día en APACHEctl como responsable del área de Hacking.

Antes de comenzar, es importante aclarar que la gente de Apache, considera que

tiene que funcionar tanto en un portátil con mínimos recursos, como en una máquina más potente y es labor de quien administre el sistema adecuar esos valores antes de poner el entorno en producción. Esto quizás responda a la pregunta de ¿por qué las configuraciones no están ya cambiadas por defecto? De eso trata este artículo.

Doy por supuesto que quienes van a leerlo, conocen el manejo básico de un editor de texto, además de estar familiarizados con el uso de una línea de comandos y un entorno GLAMP. Si alguien se pregunta el motivo por el que me centro en Apache en lugar de Nginx, es porque si bien el segundo es muy recomendable (ojo con Apache 2.4 y su entrada y salida asíncrona, proxy inverso y sistema nativo de "caching"), considero que lo ideal es controlar bien los módulos multiproceso de Apache, junto a su "problemática" a la hora de dedirse por uno y cómo mejorar los tiempos de respuesta y la carga del servidor.

También otro motivo es que por lo general si estáis empezando (o lleváis un tiempo en esto) os encontraréis con un Apache en modo Worker funcionando. En eso nos centraremos.

En algunos casos la información se presenta en modo esquema y en otros con algún texto explicativo. Además, incluyo enlaces revisados y seleccionados con detalle a otros sitios web, para que además de recomendaros una herramienta, podáis ver cómo se configura.

Esta información es válida para Servidores basados en Debian GNU/Linux y su familia de distribuciones derivadas. También servirían para otras distros, pero las configuraciones difieren y puede que falten dependencias. Además, estamos en DebianHackers ;)

Como nota final de esta introducción, recomiendo encarecidamente probar cualquiera de estas medidas y cambios en modo local, antes de hacerlo en producción. Tened en cuenta que también puede haber otros "agentes" que puedan interferir en un correcto funcionamiento como por ejemplo un panel de control tipo Plesk o Cpanel.

Mi recomendación si os hace falta una ayuda de este tipo, sería Webmin dado lo poco que interfiere en una instalación estándar y su facilidad de uso. ¿Otra alternativa? ISP Config.

Particionado y sistema de ficheros

La recomendación general sería ext4. (Ejemplo de particionado “clásico”).

```
/
/swap
/usr
/tmp
/var
/home
```

Install, particionado, ficheros en “The Debian Administrator's Handbook”. Es importante tanto en el caso de /var como /tmp, separarlas del resto por un posible incremento de logs o ficheros temporales.

Otra opción a una instalación “tradicional”, sería optar por usar LVM (Administrador de volúmenes lógicos). De este modo, podemos por ej redimensionar el tamaño de una partición en caliente sin reinstalar. Artículo recomendado para entender el uso y posibilidades de LVM.

Kernel Hardening

Sin entrar en detalles a fondo, la recomendación sería el parche de “GR Security”, podéis ver una comparativa entre SELinux o AppArmor junto a un resumen de sus funcionalidades. Este parche puede proteger de ataques a tipo escalada de privilegios, desbordamiento de la pila, “counter o integer” overflows, etc, siempre dejando claro que “puede proteger” bajo determinadas circunstancias (puede afectar al rendimiento si hablamos de compilación).

Enlaces recomendados:

- Compile GRSecurity Official Debian Patch on Wheezy”
- Repositorios e información en el Wiki de Grsecurity de Debian
- Paquetes y parches de Grsecurity disponibles en Debian

Optimización de Apache y módulos multiproceso (MPM)

Algunos aspectos a tener en cuenta antes de entrar en detalles:

✔ Es importante controlar los módulos cargados o que estén activos (revisamos en */etc/apache2/mods-enabled/*).

✔ **Revisión de sintaxis en Apache**: apachectl configtest. Para saber los datos de compilación: apachectl -V (o ver si se ejecuta en modo Worker o Prefork. Más comandos de Apache.

✔ Verificamos con *tail -f /var/log/apache2/error.log* si todo está ok cuando se carga algún módulo en Apache, modificamos apache2.conf, o tocamos php.ini. También interesa hacerlo tras ejecutar un *apachectl restart* o */etc/init.d/apache2 reload*.

✔ **Apache 2** y Multiproceso **(MPM).**"Control dinámico de los procesos según la carga bajo demanda". Para configurar estos valores, editamos en **/etc/apache2/apache2.conf**.

✔ **Prefork** (default) Procesos hijo, mod_php. En teoría más estable y compatible con más módulos o software. Mayor capacidad de aislamiento que Worker si hay un problema con una petición respecto al resto de procesos "hijo". El escenario habitual que os podréis encontrar en la mayoría de instalaciones y que necesariamente, ha de ser revisado.

✔ **Worker** lanza procesos hijos e hilos (hebras o subprocesos) por hijo, PHP se ejecuta en este caso como fastCGI. Se consigue un menor consumo de memoria que con Prefork, por resumirlo más rendimiento. ¿La pega? si un hilo falla, afectará al resto con lo que conlleva.

✔ **TimeOut** segundos antes de que se cancele un conexión por falta de respuesta (30s).

✔ **Keepalive** (conexiones persistentes) recomendable (salvo excepciones) activo = "On". Permite múltiples peticiones sobre la misma conexión TCP. Útil para sesiones HTTP de larga duración (por ej foros).

✔ **MaxKeepAliveRequests** limita el número de peticiones permitidas por conexión con KeepAlive activo. Por defecto 0 (ilimitadas). Se recomienda un valor alto (500-1000).

✔ **KeepAliveTimeout** Segundos que Apache esperará peticiones subsiguientes antes de cerrar una conexión persistente. Si ya ha sido recibida una petición, se aplica directiva Timeout para cerrar la conexión. A un valor mayor, más procesos del servidor ocupados esperando en conexiones con clientes no activos. (Rec,3 -5 seg)

Configuraciones Prefork y Worker

Prefork, multiproceso, procesos “hijo” y sin hebras.

StartServers: Número de procesos hijo creados en el arranque .

MinSpareServers: Número mínimo de procesos hijo a la espera de peticiones.

MaxSpareServers: Número máximo de procesos hijo a la espera de peticiones.

MaxClients: Número máximo de conexiones simultáneas (se aumenta con ServerLimit).

MaxRequestsPerChild: Número máximo de peticiones que sirve un proceso hijo.

Worker, multiproceso, procesos “hijo” y hebras o hilos.

MinSpareThreads: Número mínimo de hebras en espera para atender peticiones.

MaxSpareThreads: Número máximo de hebras en espera.

ThreadLimit: Límite estricto del número de hebras del servidor.

ThreadsPerChild: Número de hebras creadas por cada proceso hijo.

ServerLimit: Límite estricto de procesos hijo activos posibles. Debe ser > = MaxClients.

“Server Status”, se puede ejecutar desde la línea de comandos con un: apachectl fullstatus, o accediendo vía web (un ejemplo en apache.org). Tutorial de configuración.

Aquí os enlazo a un post que escribí hace un tiempo sobre cómo detectar y bloquear ataques con Apache Status, Google, Whois e iptables (sigue en vigor, data de 2006).

Sobre **escalado vertical y horizontal**, junto a soluciones tipo "Cloud" como Amazon, os recomiendo estos dos podcasts a modo de apoyo (y repasar el texto que les acompaña):

1) Podcast "Especial SysAdmin" con Ricardo Galli (UIB – Menéame)

2) Podcast "Especial Amazon EC2" con Ricardo Galli y Raúl Naveiras.

(Es muy importante calcular la dimensión del proyecto a poner en marcha antes de contratar un servicio de hosting y valorar las opciones de expansión si llegan incrementos de tráfico).

PHP, WPO (Web Performance Optimization), sistemas de "caching"

Instalamos Memcached a nivel de Server, el módulo para Apache memcache y X-Cache como sistema de caching para scripts de PHP. Si está instalado Memcached o X-Cache, hay CMS que pueden usarlo "por defecto" (Foros SMF) o a través de un plugin como en el caso de WordPress (Memcached Object Cache y otro muy potente como W3 Total Cache).

Instalación:

```
aptitude install memcached php5-xcache php5-memcache
```

Comprobamos si están cargados los módulos con un *php info*, o tecleando en la línea de comandos *php - -ri* (y detrás iría el módulo, por ejemplo memcache o xcache).

Relacionado con esto: Tutorial para instalar Memcached y el módulo de Apache (válido para Debian 6 y 7) y este otro más completo con soporte para WordPress. Wiki de Memcached.

Revisamos las configuraciones por defecto de los módulos ya que no son las adecuadas (X-Cache).

***Más sistemas**: Sobre todo APC (lista) y en PHP 5.5 tenemos ya Opscode.

MySQL Server. Configuraciones, seguridad, comandos y scripts útiles

El fichero de configuración de MySQL está en */etc/mysql/my.cnf*, como siempre, os recomiendo hacer una copia de seguridad del original. Para que se apliquen los cambios, es necesario teclear */etc/init.d/mysql restart*. Revisad valores el tamaño de la caché o el log de consultas lentas para detectar cuellos de botella.

Tal y como podéis leer en la conf, los errores, el log general, o logs del reinicio o arranque de MySQL Server, en Debian van a través del syslog del sistema. También es importante recalcar que hay una estrecha relación entre el valor de máximas conexiones en MySQL y el número máximo de clientes en Apache (tienen que ir de acorde el uno con el otro).

Enlaces de interés:

- "Optimizing The Server"
- MySQL Performance Tuning Scripts
- Script para analizar rendimiento y configuración
- Comprobando vía web conf y memoria necesaria
- Un ejemplo de resolución de problemas en rendimiento
- Guía de seguridad general MySQL
- Replicación de MySQL con un túnel cifrado SSL
- Bases Datos, estrategias, escalado, tipos, etc

Copias de seguridad, reparación y mantenimiento

Os dejo algunos enlaces interesantes:

- Backups de MySQL incrementales y cifrados
- Documentación oficial sobre mysqldump
- Info sobre mysqlcheck (MyISAM) → "InnoDBRecovery"

PHP, medidas de seguridad y configuraciones recomendadas

1) PhpSecInfo. Se descomprime en la raíz del sitio el fichero y se accede vía web para comprobar el estado por defecto de la instalación. Junto a cada sección, hay link con información según su color, dependiendo del impacto. Resumen y explicación. Tras un cambio en */etc/php5/apache2/php.ini* recordad que para que los lea PHP, tecleamos: */etc/init.d/apache2 reload* (o *restart*, pero cortaríamos conexiones establecidas)

2) Protección en PHP con Suhosin Patch (a 19/06/2013 fuera de Debian 7 por un bug).

3) Mod_chroot en Apache y Debian 6. Desde la v 2.2.10, soportado nativamente en Apache.

disable_functions

```
disable functions=

pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifst
opped,pcntl_wifsignaled,pcntl_wexitstatus, pcntl_wtermsig, pcntl_wstopsig,
pcntl_signal, pcntl_signal_dispatch, pcntl_get_last_error,pcntl_strerror,
pcntl_sigprocm       ask,       cntl_sigwaitinfo,       pcntl_sigtimedwait,
pcntl_exec,pcntl_getpriority, pcntl_setpriority,

system, exec, passthru, shell, shell_exec, popen, pclose, proc_nice,
proc_terminate,    proc_get_status,    proc_close,    pfsockopen,    leak,
apache_child_terminate,    posix_kill,    posix_mkfifo,    posix_setpgid,
posix_setsid,  posix_setuid,  escapeshellcmd,  escapeshellarg,  phpinfo,
proc_open, show_source, passthru
```

El primer bloque lo podréis ver por defecto en algunos servers, el segundo es un añadido.

Lecturas recomendadas:

- Varios artículos sobre el tema y relacionados
- PHP Insecurity Notes | Hardening php.ini | Info sobre suPHP.
- 25 PHP Security Best Practices | PHP IDS
- Inserting Vulnerabilities in Web Applications
- Evitar ataques XSS con PHP Input Filter
- Seguridad en PHP y tipos de ataques | Manual online de PHP, sección: seguridad

Seguridad por oscuridad (ocultando información a posibles atacantes)

Con estas técnicas, nos protegemos de posibles atacantes que buscan servicios activos en el server, intentan detectar la versión del S.O, descubrir módulos de Apache cargados, o versiones de CMS (relacionado: Webcast, Seguridad en WordPress. Demo práctica y real).

Usamos la seguridad por oscuridad como prevención de ataques automatizados buscando versiones de software vulnerables, o para protegernos de un "0 day" (una vulnerabilidad sin solución o parche que esté siendo explotada). Por lo general, se tratan de ataques automatizados del tipo "escaneo aleatorio y, caso de encontrar algún bug se ejecuta un exploit", aunque no se pueden descartar ataques dirigidos directamente contra el servidor.

También hay situaciones en las que no siempre se puede parchear. Es necesario por lo tanto, cambiar las configuraciones por defecto de varios servicios y ocultar sus "banners".

Apache > En */etc/apache2/conf.d/security*:

```
ServerTokens Prod
ServerSignature Off
```

Evitar listados de directorios, editamos apache2.conf e incluímos (bajo la directiva "log format por ejemplo) *Options – Indexes*, (y en la conf del virtualhost también debe estar el - indexes) o vía *.htaccess*. No olvidemos cargar la nueva conf con */etc/init.d/apache2 reload*.

PHP > A pesar de estos cambios, con un *curl -i laweb.com | less*, seguimos viendo versiones de SSL o PHP. En */etc/php5/apache2/php.ini* dejamos en "**Off**" el valor de expose_php.

Como en el caso anterior, para cargar una nueva configuración de PHP, hay que relanzar la de Apache.

SSH > Frente a un ataque con Nmap y aún con un firewall activo, sigue saliendo la versión y S.O or el banner de SSH. Dicha versión de SSH se oculta como una opción de compilación. También editando */etc/ssh/sshd_config* y: "**DebianBanner no**" y */etc/init.d/sshd* restart para actualizar cambios, no revela tanta info, sólo la versión. Más sobre seguridad y SSH.

MySQL > Recordad que para admitir sólo conexiones desde el localhost, hay que insertar en *my.cnf* la siguiente línea:

```
bind-address = 127.0.0.1
```

(Y después, un /etc/init.d/mysql restart).

Otros > (BIND, Postfix, etc) Os recomiendo leer esta entrada de Infosec Institute sobre el tema.

Importante incluir y configurar correctamente un *robots.txt*, frente a ataques tipo Google Hacking .Si hablamos de **ataques "extraños" y buscadores**, como caso práctico y real, os recomiendo este post, en el que podemos ver como Google realiza de forma automatizada (e insistente) ataques a un FTP, a pesar de que se le bloquea (visto en un *auth.log*).

Portsentry o, bloqueando a Nmap con un IDS / IPS

Portsentry es un gran IDS / IPS y según situaciones, puede usarse de forma "silenciosa" en modo IDS (sistema de detección de intrusos), o como un IPS (modo prevención y bloqueo). Se instala a través de aptitude o apt-get ya que está en los repositorios de Debian.

Durante la instalación, veréis una pantalla informativa explicando que por defecto actúa en modo IDS (detecta y registra el "portscan") y para que bloquee la IP atacante, es necesario activarlo en su fichero de configuración (de ese modo ya funciona en modo IPS).

Para ello, cambiamos el valor por defecto "0" (no se bloquea ningún escaneo de

puertos) a "**1**" en "*etc/portsentry/portsentry.conf*". En concreto, las líneas son: "**BLOCK_UDP**" y "**BLOCK_TCP**". Os recomiendo leerlo al completo y ver otros comandos de bloqueo.

Para probarlo, **usamos Nmap** intentando identificar puertos, servicios, versiones o el S.O del servidor. En este mismo número de "Elementals", debish os explica al detalle cómo hacerlo.

Otro valor a tener en cuenta es el "**SCAN_TRIGGER**". Puede que mientras se prueba, Nmap pase la barrera de Porsentry, en ese caso dejad el valor en 0 y tras los cambios, para que sean efectivos teclead: */etc/init.d/porsentry restart*.

Es muy importante en cualquier herramienta de este tipo, **usar las listas blancas para evitar bloqueos indeseados** a la propia IP del server, balanceador, "*failover*", etc (revisad "*portsentry.ignore*").

Para desbloquear una IP en Porsentry, tenéis que ejecutar el siguiente comando:

```
/sbin/route del -host ip_a_desbloquear reject
```

Otra opción a tener en cuenta y válida junto a Portsentry es OSSEC.

Interactuando con iptables a través de APF Firewall

De todos es sabido que la sintaxis de **iptables** no es muy amigable, mi recomendación para interactuar y ejecutar reglas a través de iptables, (siempre teniendo claro lo que se está haciendo) sería la implementación de **APF Firewall**.

Es una buena forma de iniciarse en el mundo de los firewalls para quienes empiezan y cuenta con opciones muy potentes de configuración. Son interesantes sus actualizaciones automáticas desde sitios como: spamhaus, dshield , etc. o vuestros propios repos locales y remotos) con direcciones IP consideradas maliciosas, añadiéndolas a la lista de bloqueo.

Se pueden ver sus opciones desde la línea de comandos tecleando: *apf* (-d bloquea IP, -u desbloquea, etc). Como siempre, la recomendación es leer con calma sus opciones de configuración (en */etc/apf*) y al principio del fichero, veréis la línea **"DEVEL_MODE="1"**.

Por defecto y con buen criterio para prevenir "autobloqueos", cada 5 minutos y vía el cron del sistema, lo activa y desactiva. Hasta que no tengáis claro que está bien configurado,no pongáis el valor **DEVEL_MODE="0"** .

Ya está disponible desde los sources de Debian y con aptitude o apt-get podéis

instalarlo. Os recomiendo leer con detalle y también para entender mejor que es lo que estamos haciendo (o para comprender la salida de por ej: *iptables -L*, este tutorial de iptables.

Insisto en la necesidad de documentarse a fondo antes de poner en producción estas medidas. Pero sobre todo y más en este caso, revisar las listas blancas, exclusiones, etc.

Ataques de fuerza bruta, denegación de servicio (DoS) / Mod Security

En este punto es importante recalcar que en un número muy alto de ocasiones, un servidor puede llegar a caer por cuestiones que sin estar relacionadas directamente con la seguridad, afectan del mismo modo. Es decir, por muchas medidas de protección que implementemos o por muchos recursos que tenga nuestra máquina, si las configuraciones de directivas esenciales de Apache, PHP o MySQL no están afinadas, puede que por algún pico de tráfico legítimo, el servidor quede "K.O" a la primera de cambio.

Es por ello que este artículo está escrito de forma secuencial y las medidas de optimización puestas en marcha como hemos visto al principio, nos ayudarán a que ya desde la parte más proactiva o enfocada a la seguridad, podamos bloquear este tipo de ataques.

En el enunciado no he puesto "DDoS" (ataque de denegación de servicio distribuido) ya que la forma de pararlos se escapa del ámbito de este artículo y requiere otro tipo de medidas (como por ejemplo con ataque de amplificación DNS). Rel: DoS basados en mitigación.

Hablamos de bloquear excesos de peticiones de una o varias direcciones IP dando ya por supuesto que nuestro entorno GLAMP está preparado para absorber más tráfico y servirlo con más fluidez. Recordemos que **"la disponibilidad es seguridad"** y nuestro objetivo es mantener el entorno funcionando y ahí la optimización y seguridad van de la mano.

A continuación, veremos una serie de herramientas y enlaces recomendados para implementarlas, os recomiendo abrirlos e ir leyendo con calma para ver las posibilidades reales de cada una de ellas. También veremos cómo probar y monitorizar su uso.

1) Ataques SSH / fuerza bruta: Para proteger SSH, como primera opción, podemos usar Denyhosts y también Fail2ban que controla más servicios como FTP, mail o Apache.

Otros: BFD junto a APF y SSHGuard. Rel: Protegiendo FTP / SSH con **Latch** (Plugin SSH).

2) Enjaulado: Puede ser interesante en ocasiones hacer un chroot para SSH / SFTP OpenSSH y usuarios SFTP. Tutorial VSFTP, parche para fallo en chroot y Wheezy.

3) Mod Evasive: Todo un clásico. Tutorial de instalación junto a Mod Security . En THW, explicación a fondo, donde citan un bug en las políticas de bloqueo según versiones.

4) Mod Security: reconocido WAF (Web Aplication Firewall) Tutorial para Debian 7, configuración y un ejemplo para WordPress. Otros**:** como apoyo, el libro de Ivan Ristic. Análisis a fondo de sus directivas por THW. Reglas ModSec: OWASP - AtomiCorp – SpiderLabs (como veréis, algunas son de pago, podéis repasar funcionalidades).

5) Dos Deflate: Sencillo y útil script que puede bloquear con iptables o APF. Una vez instalado, revisad con un: crontab -e (usuario root) si ha añadido la tarea o también podéis mirar en el syslog con el siguiente comando: grep -i ddos /var/log/syslog

Los ficheros de configuración están en: */usr/local/ddos/*

Repositorio y paquetes para Debian 6 y 7 (Grsec, Mod Evasive, Mod Security – CRS, etc).

¡Ojo! Todas estas medidas de seguridad deben ser auditadas y probada su eficiencia, para ello os propongo varias herramientas. Tened en cuenta que las pruebas deben hacerse en un entorno controlado y según su alcance, si hablamos de ataques DoS o fuerza bruta, bien con el permiso del proveedor de Hosting, o de todas las partes implicadas.

6) Pruebas DoS - DDoS / Fuerza buta:

Escrita en Ruby y actualmente (abril 2014) sin continuidad, revisad las opciones de PenTBox. También puede resultaros de utilidad el comando ab, por ej: ab -n 500 -c 150 donde “n” es el número de peticiones y “c” el número de clientes (recordemos en la parte de Apache, directivas como MaxClients ya que está estrechamente relacionado).

* **Otros:** DDoS Simulator, Hydra, Load Impact, Jmeter. “Testing Denial Of Service” (OWASP), Hulk.

Os recomiendo también visitar este listado de aplicaciones relacionadas.

7) Servicios CDN y protección DDoS: CloudFlare, Incapsula, Prolexic. Para ver cómo afronta un proveedor de hosting estos temas, podéis ver la Protección Anti DDoS en **OVH**.

Del mismo modo que os aconsejo que se prueben las medidas de seguridad, también es interesante que ese tráfico generado contra el server pueda ser monitorizado. Para ello os propongo varios comandos y aplicaciones hablando de **monitorización**: iptraf, vnstat, comandos de netstat; otros: lsof, iostat, iotop, iftop, Arpwatch, Monitorix.

Comandos de ejemplo con netstat (tráfico SYN, peticiones, número conexiones, etc):

```
netstat -an | grep :80 | sort

netstat -plan|grep :80 | awk {'print $5'}|cut -d: -f 1|sort|uniq -c|sort -n

netstat -lpn|grep :80 |awk '{print $5}'|sort

netstat -an | grep :80 | awk '{ print $5 }' | awk -F: '{ print $1 }' | sort
| uniq -c | sort -n

netstat -n -p | grep SYN_REC | awk '{print $5}' | awk -F: '{print $1}'
```

Control de registros del sistema (logs) y monitorización de servicios

Desde para tener logs de sistema fuera del servidor por si no podemos acceder a la máquina, a contar con sistemas de monitorización accesibles vía web, con gráficas e información sobre lo más relevante del sistema, a otras soluciones como Monit capaz de trabajar de forma proactiva si algún recurso cae. Veamos esta lista que os propongo:

1) Logcheck (aptitude install logcheck), aplicación que **analiza los logs** de la máquina y muestra o envía por email un informe (por defecto cada hora) sobre sucesos relevantes del sistema. Información y web del proyecto.

2) Logwatch (aptitude install logwatch), similar a logcheck, pero proporcionando más información. Envía **un informe diario por email** con registros de /var/log (revisad el fichero de configuracion). Información y web del proyecto.

3) Monit y munin:

Monit, monitoriza procesos daemon (sendmail, Apache, MySQL, etc). Puede **reaccionar a los eventos**, intentando **levantar el proceso caído,** avisando vía email de su intervención. Interesante también el servicio M/Monit. Rel: Pingdom avisa de caídas (SMS, Mail, Twitter).

Munin es otra herramienta de monitorización con múltiples plugins que permiten hacer un seguimiento detallado y ver gráficamente lo que sucede en el sistema. Ejemplo de uso.

Monit y Munin se instalan vía aptitude. Rec, guía de instalación (ENG), otro tuto (Wheezy).

Nagios puede realizar prácticamente el trabajo de los 4 anteriores. Múltiples plugins y "checks", robusto y fiable. Vía aptitude está **Nagios 3**, pero os sugiero leer este completo tutorial (rama 2.x) junto a otro sobre la versión 3.3.1 en Debian 6.0.1. Más información: blog "Nagios en Español".

4) Pandora FMS es otra opción similar a Nagios, altamente configurable, muy bien documentado y con un monitor muy potente y alertas para controlar en tiempo real lo que sucede en el server. Información y web del proyecto.

5) Monitorización y Zabbix, otra alternativa recomendable. Tutorial Debian 7 / Ubuntu 13.10.

6) Hardware y **recursos** con PhpSyInfo. Tutorial de instalación (ojo, disable_functions en PHP).

7) GoAccess, tutorial, analiza logs de Apache y Nginx. Y relacionado con esto**:** apachetop, Afterglow y Rsyslog.

Detección de Rootkits, malware y otros recursos relacionados

Para terminar, os propongo una serie de herramientas y enlaces que os resultarán útiles para comprobar la integridad de ficheros, detectar posibles puertas traseras en el sistema, código malicioso en aplicaciones web y otros recursos que espero os sean de utilidad.

1) Detección de rootkits: RKHunter + Unhide (revisad las opciones) y Chkrootkit. Ambos se instalan con aptitude / ap-get. Rel: Sobre los IDS o cómo funciona Unhide, Kernel Panic 43.

Otras aplicaciones de este estilo, para comprobar la integridad de ficheros o binarios además de otros usos son: AIDE o Tripwire (Rkhunter también compara hashes MD5 de ficheros importantes con su firma correcta en una base de datos en línea, aunque puede dar algún "falso positivo" que conviene siempre revisar si lo es realmente o no).

2) Chequeo general de parámetros de seguridad: Lynis (sintaxis básica lynis -c). Es práctica para ver de forma rápida el estado general de la seguridad del sistema. Un ejemplo e info.

3) Detección de WebShells comunes en un WordPress o Joomla comprometidos. LMD, Una vez instalado, analiza todos los ficheros buscando cadenas sospechosas y luego vía el cron, lo hace de forma incremental con cada fichero nuevo a diario (sintaxis: maldet -a /path). Revisad la configuración de LMD ya que por defecto, no pone en cuarentena los afectados.

4) Análisis de código estático de aplicaciones PHP en busca de vulnerabilidades con RIPS. **Otros**: Webserver Malware Scanner o Sucury Web Site Scanner (varios planes de pago).

Y para terminar:

WordPress, recopilación de plugins y medidas en Apache para mejorar su seguridad, backups, detectar malware, etc. Rel: Protección de WordPress con **Latch** y otros.

Otros: JackTheStripper. Forat "Project", Políticas de acceso y usuarios, Hardening básico, eBook "Debian 7 SysAdmin", libro "Hardening Servers" 0xWORD, problemas en Wheezy.

INTECO CERT: Seguridad y Servers; DNSSEC; "Servers Hardening".

Y en DragonJar TV, podéis ver mi demo con varios temas comentados aquí (las diapos).

Enhorabuena, ¡has llegado al final!

Espero que podáis aplicar toda esta información tanto en vuestros servidores de pruebas como en producción. No puedo terminar por aquello de los “créditos”, sin dar las gracias a **mis compañeros de DebianHackers** por animarme a transcribir en “modo humano” ;D, la información del PDF sobre mi demo de ConectaCON Jaén, así como de una parte del material que proporcioné a los alumnos de mi lab en Rooted CON 2014 sobre “Seguridad y Optimización en servidores GLAMP”.

Aprovecho la ocasión para daros también las gracias a todos vosotros en esta primera entrega de “DH Elementals”, por estos cuatro años de apoyo continuado a un proyecto como el nuestro que sólo busca compartir nuestra pasión por Debian y su familia de .deb.

Y por supuesto, a toda la comunidad que hacen posible que Debian siga fuerte y estable, proporcionándonos el que para mi sin duda es el mejor Sistema Operativo que puedo usar. Tanto en servidores web como en mi escritorio.

¡Debian GNU/Linux rules!