Aprovisionamiento seguro de servidores

La seguridad de los servidores en los que desplegamos los servicios que dan soporte logístico a las actividades militantes es tan importante como la seguridad del software que ejecutamos en estos servidores. Por mucho que un software, por ejemplo un servidor de mensajería, sea muy seguro y provea cifrado en las comunicaciones, esta seguridad depende en gran medida de la seguridad de aquel sitio en el que se ejecuta.

A continuación, explicamos las líneas generales a seguir para desplegar un servidor de la forma más segura posible. Nótese que nos vamos a centrar en el software y no en el hardware, ya que consideramos que este último merece una guía aparte.

La guía aplica una brocha bastante gorda para mantener un equilibrio entre su carácter de guía general y su ilustración mediante ejemplos simples. Se da por sentado un modesto nivel de conocimiento en la materia por parte del lector, por lo que no se explican todos los pasos ni los comandos para aplicar las configuraciones que recomendamos.

Alojamiento

Las dos formas principales de aprovisionar un servidor son mediante el alquiler a un proveedor de hosting o mediante el uso de hardware propio.

Los proveedores de hosting son empresas o colectivos que ofrecen su infraestructura (firewall, sistemas operativos, servicios preconfigurados, etc.) para que otros usuarios u organizaciones utilicen parte de esa infraestructura. Generalmente se alquilan estos recursos a cambio de dinero, mientras que en el caso de los colectivos varía según la dependencia económica de estos. El *selfhosting* se refiere al uso de una infraestructura propia para la ejecución de software y su exposición a otras redes. A veces el *selfhosting* también se entiende como el despliegue de un servicio por cuenta propia en un servidor de un proveedor ajeno.

Para hacer *selfhosting* en hardware propio, evidentemente se necesita hardware en el que ejecutar el sistema operativo y los servicios, y también se necesita hardware para realizar conexiones entre redes. En el caso más sencillo, un servidor puede ser una computadora conectada a otras redes o a Internet y con los puertos abiertos para que sean accesibles desde otras redes o Internet. Consideramos que el conocimiento necesario para operar una infraestructura de este tipo es tal que una explicación como la que podríamos dar aquí le resultaría superflua a quien fuera a implementar una infraestructura de hosting por su cuenta.

De más interés puede ser hablar sobre los proveedores de hosting, ya que suele ser esta una opción más común entre las organizaciones que quieren levantar un servidor web, de correo o de mensajería. Consideramos que la elección de un proveedor debe hacerse en base a los requisitos que exponemos a continuación:

- Se deben valorar aquellos proveedores que son o bien colectivos anticapitalistas

que ofrecen su infraestructura para satisfacer necesidades organizativas o bien aquellos con buenas políticas de privacidad y anonimato. En el primer caso, un buen ejemplo puede ser Autistici/Inventati y cualquiera de los colectivos que aparecen en esta lista. En el segundo caso, la mejor opción puede ser Njalla, proveedor que ofrece hosting que puede pagarse mediante criptomonedas y que ofrece métodos de acceso anónimos.

- El país en el que están los servidores físicos. Hay que evitar en la medida de lo posible aquellos países donde las leyes facilitan el espionaje, las redadas o, en general, donde el marco legal supone una amenaza a la privacidad y seguridad. Por regla general, debemos evitar los países dentro de los Five Eyes, Nine Eyes o Fourteen Eyes.

- Los sistemas operativos que ofrece o si permite modificar el sistema operativo. Más adelante hablamos de los sistemas operativos que recomendamos. En cuanto a la modificación del sistema operativo, un ejemplo ilustrativo de a que nos referimos puede ser esta guía en la que se explica cómo instalar OpenBSD en un VPS del proveedor OVH, ya que este proveedor no ofrece OpenBSD como sistema operativo.

- Posibilidad de pagar en criptomonedas, ya que este tipo de pagos dificultan el rastreo. Aunque para muchos usuarios no es lo más cómodo, es preferible poder pagar en monedas digitales no rastreables como Monero (Bitcoin es rastreable), que hacerlo mediante métodos tradicionales como pueden ser las transacciones bancarias (que asocian una cuenta a una persona) o Paypal u otras plataformas de pago, ya que dependen de la banca tradicional.

En cuanto al precio a pagar, no lo incluímos en la lista anterior porque si bien es algo a valorar, no tiene nada que ver con la seguridad. Cada organización tiene que valorar la relación entre lo que paga y la seguridad que va a recibir.

Por último, es muy importante saber adaptar el servicio que se solicita a las necesidades que se quieren satisfacer. Es decir, si por ejemplo solo se quiere alojar una página web, con un hosting web como Neocities es más que suficiente y no es necesario utilizar todo un VPS. Si además en la organización se tienen conocimientos para desarrollar páginas web estáticas, es mejor hacer este desarrollo y deplegarlo en un simple servidor de páginas estáticas, que utilizar un CMS como Wordpress para desplegar una página estática. La simplicidad y seguridad suelen ir de la mano y deben ponerse por delante de la comodidad cuando las personas que van a gestionar los servicios tienen conocimientos como para no depender de software sobredimensionado y adaptado a usuarios sin conocimientos técnicos.

Sistemas operativos

La elección del sistema operativo (SO) es extremadamente importante. De esta decisión puede depender la mayor parte de las medidas de seguridad que se tomen posteriormente.

Si tenemos que recomendar un SO, no tenemos dudas en que el más seguro es

OpenBSD. Este SO debe ser siempre la primera opción y la que hay que intentar desplegar antes de valorar cualquier otra. Su postura de seguridad proactiva y la reputación que le precede entre los expertos en seguridad de la información, hacen de este sistema operativo el mejor en cuanto a seguridad.

La otra opción común para el despliegue de servidores, por la libertad que provee, por su variedad y por su eficiencia, es Linux. Sin embargo, es importante tener en cuenta que **Linux no es un sistema operativo seguro**. Nos gusta Linux, pero siendo realistas no es un sistema operativo seguro. Aún así, es una de las opciones más adoptadas en los servidores y por eso tenemos que recomendar que, en caso de no utilizar OpenBSD (la única opción realmente segura), se deben valorar distros como Alpine Linux o Void Linux. El criterio general para la decisión sobre qué distro de Linux utilizar es que reduzca la superficie de exposición lo máximo posible, por lo que hay que priorizar distros que no implementen Systemd, que implementen musl libc en vez de glibc, utilicen doas u OpenDoas en vez de sudo, compilen binarios como Position Independent Executables (PIE), etc. Como ambos (*BSD y Linux) son SOs tipo Unix, el salto de uno a otro no supone mucho tiempo de aprendizaje.

Como indicamos, la elección del SO debe adaptarse a las necesidades de la organización, aunque nosotros hayamos priorizado aquí los SOs de tal forma que la primera opción, OpenBSD, es la más segura, mientras que Linux, no siendo realmente segura, ofrece más cobertura de cara a implementar casos de uso más variados.

Reducción de la superficie de ataque

La superficie de ataque se refiere a la cantidad de puntos que un atacante podría utilizar para conseguir debilitar la seguridad de un sistema (estas vías de ataque se suelen denominar *vectores de ataque*). Por lo general, la superficie de ataque crece en dos medidas: la calidad y la cantidad del software. Cuanto más grande es el software (líneas de código y componentes implicados) y cuanto más software diferente hay en un sistema (servicios, interfaces, librerías, módulos del kernel), más crece la *probabilidad* de que haya agujeros de seguridad. Cuanto peor es la calidad de cada unidad de software, esa *probabilidad* también aumenta. Hablamos de *probabilidad* porque no hay una relación directa entre cantidad, calidad y seguridad.

Por tanto, el objetivo es reducir esta superficie de ataque al mínimo para dificultarle la labor a un posible atacante. La primera decisión es la explicada en el apartado anterior: un buen sistema operativo debería reducir la superficie de exposición al mínimo, mediante software de calidad y la reducción de este al mínimo necesario. Las subsiguientes medidas se aplican ya sobre el sistema operativo y sobre el software que se instalará en él para cumplir una determinada función.

SSH

Lo primero que se suele hacer cuando el servidor ha sido iniciado es acceder por Secure Shell (SSH). Aunque hay varios software que implementan un servidor SSH el más conocido y el más seguro es OpenSSH.

Lo primero que se debe hacer es crear un usuario con los justos privilegios (explicado

en el siguiente apartado) para desempeñar funciones básicas y configurar el servidor SSH de forma que:

- El superusuario o usuario root no pueda acceder por SSH. El acceso por SSH hay que hacerlo mediante usuarios con privilegios muy restringidos.
- La autenticación se haga **únicamente** mediante clave pública.
- Se cambie el puerto SSH por defecto.
- Se limite el máximo de intentos de autenticación.
- Se limite el máximo de sesiones SSH concurrentes.
- Se reduzca al mínimo las acciones que se pueden realizar a través de SSH.

Un buen ejemplo creemos que es la configuración SSH de los servidores de GrapheneOS, aunque debe adaptarse a las necesidades de cada organización. Es decir, que copiar y pegar esa configuración no va a funcionar.

Antes de restringir el acceso por clave pública, se debe crear una clave pública por cada máquina que accederá al servidor. Con OpenSSH se utiliza ssh-keygen(1). Es importante utilizar una buena criptografía (recomendamos ed25519, opción por defecto a partir de OpenSSH 9.4). El comando más básico para lograr esto es:

```
ssh-keygen
```

Si es antes de OpenSSH 9.4, recomendamos:

```
ssh-keygen -t ed25519
```

Preguntará la ruta en la que se quiere guardar la clave (la que viene por defecto suele ir bien).

Una vez generada, se debe trasladar al servidor, o bien copiando manualmente el contenido de la clave pública generada (la que acaba en .pub) en la ruta de las claves autorizadas (con OpenSSH normalmente es en *$HOME/.ssh/authorized_keys*) o bien utilizando ssh-copy-id:

```
ssh-copy-id -i clave.pub usuario@servidor
```

En el caso de la configuración del servidor OpenSSH (*sshd*), el acceso con superusuario se desactiva con la opción:

```
PermitRootLogin no
```

Para limitar el máximo de intentos de autenticación (por ejemplo a 3):

```
MaxAuthTries 3
```

Para limitar el máximo de sesiones concurrentes (por ejemplo a 2)

```
MaxSessions 2
```

Para configurar la autenticación con clave pública y prohibir el acceso mediante contraseñas:

```
PubkeyAuthentication yes
PasswordAuthentication no
```

Para cambiar el puerto de acceso (por ejemplo al puerto 1234):

```
Port 1234
```

Una buena medida de seguridad es aceptar únicamente claves públicas con criptografía potente. Por ejemplo Ed25519:

```
PubkeyAcceptedKeyTypes ssh-ed25519
```

Llevando la criptografía de clave pública más lejos, la utilización de autenticación basada en hardware, por ejemplo con Yubikeys u Onlykeys puede aumentar la seguridad añadiendo un factor más a la autenticación. En este enlace puede leerse un buen tutorial sobre cómo utilizar claves de este tipo.

Cómo reducir las acciones que se pueden realizar a través de SSH es un tema que da para largo. Recomendamos estudiar bien la documentación de sshd_config(5).

Firewall

El firewall no es una herramienta de seguridad, sino una herramienta que implementa una política de control de acceso. Una buena política de control de acceso es, la mayoría de las veces, una política de **mínimo privilegio**: aquella que prohíbe acceso a todo excepto a lo que se va a utilizar (por oposición a una política en la que todo está permitido, excepto algunas prohibiciones).

Por lo tanto, primero se debe decidir la política de acceso. Según nuestro criterio, lo más importante es prohibir **cualquier** tipo de tráfico. A partir de aquí vamos habilitando reglas. Normalmente, hay que permitir tráfico en la interfaz loopback (la interfaz que el sistema operativo utiliza para comunicarse consigo mismo). Luego se suele permitir iniciar conexiones **salientes** a los puertos 53 (DNS) y 123 (NTP), para que la máquina sea capaz de resolver dominios y mantener la hora sincronizada. El primer puerto al que se suele permitir acceder desde fuera de la máquina es el puerto SSH, normalmente el 22, **pero** como hemos recomendado más arriba, es buena idea cambiarlo. Con esta configuración seguramente ya tenemos una máquina funcional a la que podemos acceder mediante SSH. Ahora se trata de abrir únicamente los puertos de los servicios que vamos a ejecutar. Si vamos a utilizar el servidor para ejecutar un servidor web, normalmente abriremos los puertos 80 y/o 443. Lo mismo

con cualquier otro servicio.

Cada sistema operativo implementa su propio firewall. En OpenBSD se utiliza **pf(4)** y en Linux lo habitual es utilizar **iptables** o **UFW**.

A continuación, mostramos la configuración segura más básica para cada una de las implementaciones de firewall que hemos nombrado. Se restringe todo el tráfico, se habilita el tráfico en loopback, SSH y, por poner un ejemplo, se permite tráfico entrante los puertos 80 y 443 para simular el despliegue de un servidor web. Los ejemplos son **meramente ilustrativos** y deben adaptarse a cada entorno, añadiendo o quitando reglas. Si copias y pegas los ejemplos es muy probable que algo no te funcione. El ejemplo más común es que si no permites iniciar conexiones hacia afuera a los puertos 80 y 443, los gestores de paquetes (apk, apt, pacman, etc.) no podrán acceder a los repositorios.

Archivo de configuración para **pf** (OpenBSD):

```
# Permitir tráfico en loopback
set skip on lo

# Bloquear todo el tráfico entrante
block in all

# Permitir todo el tráfico saliente (no es la mejor práctica, pero vale p
ara el ejemplo)
pass out all

# Permitir tráfico entrante en el puerto SSH
pass in on egress proto tcp from any to any port 22

# Permitir tráfico entrante en los puertos del servidor web
pass in on egress proto tcp from any to any port {80, 443}
```

Comandos para **iptables** (permitimos todo el tráfico saliente, aunque no sea la mejor práctica):

```
# Permitir tráfico en loopback
iptables -A INPUT -i lo -j ACCEPT

# Permitir tráfico entrante al puerto SSH
iptables -A INPUT -p tcp --dport 22 -j ACCEPT

# Permitir tráfico entrante en los puertos del servidor web
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT

# Bloquear el resto del tráfico entrante
iptables -A INPUT -j DROP
```

Comandos para **UFW**:

```
# Prohibir todo el tráfico por defecto
ufw default deny incoming
ufw default deny outgoing

# Permitir tráfico saliente al puerto NTP
ufw allow out 123/udp

# Permitir tráfico saliente al puerto DNS
ufw allow out DNS

# Habilitar el puerto SSH y protegerlo ante ataques de fuerza bruta
ufw limit SSH

# Permitir tráfico entrante en los puertos del servidor web
ufw allow in 80/tcp
ufw allow in 443/tcp
```

Usuarios y privilegios

Mantener una buena gestión de usuarios y privilegios es fundamental a la hora de mantener la seguridad de un sistema operativo. Hay dos variantes principales para aplicar esta medida. La primera es relativa a los permisos de los usuarios que utilizamos nosotros, los humanos, que denominamos usuarios operadores. La segunda es la de los usuarios que ejecutan los servicios, como pueden ser un servidor web o un servidor de chat. A estos últimos los llamamos cuentas de servicio.

Una mala gestión de los privilegios de los usuarios puede, por ejemplo, dar acceso a una shell a un atacante que consiga acceso a través de la explotación de una vulnerabilidad en un servicio. Por poner un ejemplo más ilustrativo, si un atacante explota un servidor web que se ejecuta como superusuario, es muy probable que consiga acceso de administrador a la máquina y, en definitiva, logre completo acceso a ella. A la acción de conseguir mayor nivel de privilegio se le llama **escalada de privilegios** y es precisamente lo que pretendemos evitar.

Los usuarios operadores siempre tienen que tener unos permisos muy limitados (mínimo privilegio) para poder realizar las tareas que tienen asignadas. No deberían poder realizar cualquier tipo de tarea como superusuario (usando `sudo` o `doas`), sino solo las estrictamente necesarias. Por poner esto en práctica, aunque debería adaptarse a las necesidades de cada organización, creemos que es una buena práctica dar pocos privilegios al usuario que accederá a través de SSH y como mucho permitirle cambiar a superusuario, pero no ejecutar cualquier comando como superusuario. Imaginemos que este usuario se llama “usuario” y queremos que solo pueda reiniciar el sistema operativo (lo cual requiere permisos de superusuario). Con `doas`, habría que configurar el archivo *doas.conf* de una manera parecida a esta:

```
permit usuario cmd /sbin/reboot
```

Si quiere hacer otras acciones como superusuario, tendrá que cambiar a la cuenta *root*, haciendo uso de `su`. Hemos hecho este ejemplo con `doas`, aunque se puede hacer también con `sudo`. El objetivo es entender en qué consiste la separación de privilegios, dejando la implementación concreta bajo responsabilidad del lector, que deberá estudiarse la documentación de las herramientas de gestión de privilegios del sistema operativo que esté utilizando.

El otro factor a tener en cuenta es, como hemos dicho anteriormente, el de las cuentas de servicio. Brevemente, un servicio debe ejecutarse con el mínimo de privilegios para poder desempeñar sus funciones. Lo habitual suele ser crear una cuenta de servicio que apunte a */usr/sbin/nologin*, de tal modo que no pueda hacer login en una shell. La mayoría de gestores de paquetes de los sistemas operativos Unix suelen gestionar la creación de estas cuentas cuando se instala software, pero no siempre es así.

Hay diferentes formas de asignar la ejecución de servicios a una cuenta de servicio con mínimos privilegios. Un buen software suele permitir hacer esto mediante un archivo de configuración. Un buen software es también aquel que si necesita realizar gestiones como superusuario (por ejemplo, abrir puertos), ejecuta varios procesos, uno de los cuales lo hace con permisos de superusuario (abrir puertos) y el resto de procesos (por ejemplo, servir ficheros) se ejecutan con una cuenta de servicio sin (tantos) permisos.

Compartimentación

Si el atacante ha conseguido hackear un servicio expuesto en nuestro servidor, con suerte la propia seguridad del sistema operativo y la ausencia de privilegios del usuario con la que se ejecutaba el servicio, no le permitirán llevar adelante el ataque. Ahora bien, este escenario ideal no siempre ocurre. Una de las medidas que pueden salvaguardar la seguridad del sistema si las anteriores han fallado, es la **compartimentación** de los servicios.

Compartimentar servicios significa aislar su ejecución en entornos controlados y separados del resto del sistema. Hay muchas maneras de lograr esto, pero las más comunes suelen ser las máquinas virtuales, los contenedores o los *chroot* o jaulas. Evidentemente, que estos entornos separados estén efectivamente separados es algo que depende de la configuración de los orquestradores de contenedores, los hipervisores de las máquinas virtuales y otros elementos que dependen tanto del gestor (nosotros) como del desarrollo seguro de esas herramientas.

Como todo, la compartimentación no es infalible, pero un buen *chroot* puede impedir que un atacante pueda ejecutar una shell, mientras que una máquina virtual hace difícil que el atacante pueda escapar del hipervisor. Para implementar máquinas virtuales hay muchas herramientas, de las cuales nosotros recomendamos QEMU o vmm(4) para virtualización en OpenBSD. En cuanto a contenedores, algunos de los típicos son Docker, containerd o Podman.

Para aplicar jaulas *chroot*, existe LXC que gestiona contenedores que en realidad son *chroots*, aunque también puede orquestrar máquinas virtuales. Pero lo que más nos interesa de los *chroot* es entender que hay mucho software que permite hacer *chroot* en su ejecución. Es decir, que cuando se ejecuta, el servicio se encierra en un directorio específico y no puede salir de él ni ejecutar programas que no se encuentren en ese chroot. Siempre que un software permita hacer esto, es conveniente utilizar esta opción.

Resumiendo, cada servicio debería ejecutarse en un entorno aislado para sí mismo, para dificultar el avance de un ataque exitoso sobre dicho servicio hacia el resto del sistema operativo.

Conclusión

En esta guía hemos seguido los pasos esenciales para proteger progresivamente un servidor, empezando por la elección de un sistema operativo “seguro” para ahorrarnos muchos dolores de cabeza.

Luego hemos ido reduciendo la superficie de ataque, empezando por configurar de forma segura la conexión por SSH, que suele ser el método más utilizado para acceder a la gestión de un servidor. Una vez configurado el acceso por SSH, procedemos a restringir lo máximo posible el acceso remoto, mediante el uso del firewall.

En este punto, la mayoría de problemas de seguridad recaen sobre el software que se ejecuta en ese servidor de cara a Internet o a otras redes accesibles por posibles atacantes. Partiendo de que la seguridad de estos servicios nunca es perfecta (no existe la perfección en seguridad informática), tenemos que reducir las acciones que un eventual atacante puede realizar si ha conseguido acceso al servidor a través de la explotación de un servicio expuesto. Para ello, tenemos que asegurarnos de que el software explotado corre con la menor cantidad de privilegios posible y dificultarle escalar privilegios o acceder a otras partes del sistema; aislar al atacante, en resumen. Para esto último, hemos propuesto gestionar bien los privilegios de los usuarios y servicios, a la par que compartimentar los entornos en los que se ejecutan los servicios.

Colectivo 406