

Simplificamos la Denegación de Servicio

APACHE HTTPD

Muerte lenta a la configuración predeterminada. **POR KURT SEIFRIED**

Imaginemos el siguiente experimento que, aunque no tanto como para hacer historia, es lo suficientemente preciso como para aportar datos interesantes, en el que programamos un servidor basado en sockets. El primer cliente se conecta, solicita un archivo, recibe dicho archivo, se desconecta y todos tan contentos. El tiempo total empleado en la transacción es de alrededor de 300 milisegundos. Pero un poco después recibimos un reporte indicándonos que alguien ha desplegado el servidor web y le va cada vez más y más lento hasta que termina muriendo. ¿Qué hacemos? Después de diagnosticar el problema, nos damos cuenta de que hay clientes que no gestionan la conexión de forma correcta. La crean pero no hacen nada con ella, simplemente la abandonan dejándola abierta indefinidamente. El resultado es un servidor que va agotando sus conexiones simultáneas disponibles hasta llegar al punto de no poder atender más solicitudes. Es un problema fácil de arreglar: Utilizamos una directiva *TimeOut* de modo que las acciones que utilicen recursos (como las conexiones) sean terminadas cuando en realidad no estén haciendo nada.

Este método funciona bastante bien durante un tiempo, pero entonces nos damos cuenta de que las páginas tardan en cargar cada vez más y más debido a que no se trata de páginas simples, sino de

páginas con imágenes, archivos CSS, archivos JavaScript, etc. Como buenos programadores, permitimos al cliente mantener abierta la conexión para que pueda reutilizarla en posteriores peticiones. Se trata de una buena solución, ya que ahorra el coste de la conexión (un saludo TCP de tres vías [1] que lleva su tiempo). Ahora, las segundas peticiones son mucho más rápidas y todo el mundo vuelve a estar contento. Hemos aprendido de nuestros errores, así que ponemos una directiva *KeepAliveTimeout* preventiva. La vida es bella, la gente descarga cosas de nuestro servidor, y muy pronto aproximadamente el sesenta por ciento de la web está usando el servidor HTTPD de Apache.

Predeterminados Inocuos

Los valores de configuración predeterminados son uno de los problemas más desconcertantes, ya que, las configuraciones predeterminadas no vienen bien a todo el mundo. El Sitio A podría dedicarse a servir millones de pequeñas imágenes, mientras que el Sitio B podría optar por servir cosas a través de un gran framework de aplicaciones, incluso puede que los sitios de la C a la Z no sepan aún muy bien a qué se dedicarán. De cualquier modo, el servidor web parece que funciona bastante bien, así que ¿por qué preocuparse por la configuración predeterminada? Los fabri-

cantes de sistemas operativos no están dispuestos a cambiar la configuración predeterminada de todo el software que proporcionan a menos que cuenten con buenas razones para hacerlo, simplemente porque es una obligación más que atender. Además, como consecuencia directa, el software se comportaría de forma inesperada, con las consiguientes llamadas de petición de soporte con las que nadie quiere lidiar, especialmente en el caso de organizaciones gestionadas por voluntarios. Así que no queda otra que confiar en que el proyecto elegirá la configuración predeterminada más adecuada para el software, algo que probablemente sea cierto, ya que son ellos quienes mejor lo conocen y mejor saben qué se puede romper al trastear los mecanismos de éste. Por tanto, todo el mundo acaba usando los valores que mejor se ajustan a la mayoría, dando por hecho que no sucederá nada extraño, como que se conecten simultáneamente 1000 o más clientes con conexiones de red verdaderamente lentas.

¿Qué Pasa Cuando...?

Cuando 1000 clientes se conectan a un servidor a través de enlaces realmente lentos (o un cliente simula mil conexiones desde un enlace lento), todos a la vez, resulta que el servidor deja de funcionar. Mejor dicho, aún funciona, pero está limitado al número de conexiones que es capaz de atender, por lo que aunque no se consuman todos los recursos de la máquina, el servidor no dispondrá de más conexiones con las que atender a sus clientes legítimos. A ojos del mundo, nuestro servidor habrá muerto. Esta situación supone un problema grave, ya que cualquier usuario que disponga de un programa al estilo de Slowloris [2] dispondrá del potencial necesario para atacar un sitio de gran envergadura desde una sola máquina y desde un enlace de red relativamente modesto (como pueda ser una conexión DSL o un cable módem).

¿Y no se podría evitar fácilmente limitando el número de conexiones que se pueden crear y mantener abiertas desde una sola dirección IP o segmento de red? De

establecer un valor muy bajo, bloquearíamos el acceso a los usuarios a los que no les queda más remedio que usar proxy. AOL, por ejemplo, obliga a sus usuarios a atravesar proxies web (algo que les supone un ahorro económico tremendo en cuanto a ancho de banda). Son muchos los casos legítimos en los que una sola dirección IP o pequeño grupo de direcciones IP de una red local podría abrir muchas conexiones (por ejemplo, cada usuario de AOL en el medio oeste de América). El problema aquí radica en encontrar un límite que prevenga daños al servidor al tiempo que es lo suficientemente alto como para no bloquear a usuarios legítimos. Lo mejor es establecer un límite lo más alto posible (es decir, que aunque afecte al servidor, no lo tumbe completamente) para bloquear a la menor cantidad de usuarios posible.

Una forma genérica de determinar el límite de conexiones por IP es acotando la tasa de transferencia mediante *iptables*, algo que se puede hacer de forma selectiva en puertos individuales. Además, es posible especificar un bloque de tiempo y el número máximo de conexiones que se pueden realizar dentro de ese espacio de tiempo. El siguiente código creará un bloque de sesenta segundos con un límite máximo de cinco conexiones. A partir de la sexta conexión aplicará una regla DROP para los paquetes, haciendo que el cliente continúe reintentándolo por tiempo indefinido. Al finalizar una de las conexiones anteriores, se permitirá el establecimiento de una nueva.

```
iptables -I INPUT -p tcp \
-dport 80 -m state \
-state NEW -m recent --set
iptables -I INPUT -p tcp \
-dport 80 -m state \
-state NEW -m recent --update \
--seconds 60 --hitcount 6 \
-j DROP
```

Claro está que un atacante decidido usará más máquinas, saturando nuestro intento de limitar la tasa, pero podemos dificultar su tarea de manera significativa.

Sacrificando Rendimiento en Pos de la Supervivencia

En caso de no disponer de los fondos necesarios, o simplemente de que no se quiera gastar dinero en desplegar más hardware y software destinado a repeler ataques de denegación de servicio, tiene la opción de



Figura 1: Toma su nombre de un primate muy lento con un agarre muy fuerte.

afinar el rendimiento para conseguir una mayor supervivencia a este tipo de ataques. Como defensa al ataque Slowloris, la acción más rápida y posiblemente efectiva consiste en cambiar el valor predeterminado de *Timeout* (que normalmente es de 300 segundos o cinco minutos) a un valor mucho menor (cinco segundos, o menos en caso de que fuese necesario). Además de esto, para prevenir un eventual abuso de la directiva *keep-alive* se puede deshabilitar ésta simplemente poniéndola a *off*. Cabe destacar que ninguna de estas soluciones alternativas arreglará en realidad el problema de manera significativa, aunque servirán de ayuda ante atacantes con recursos limitados [3]. Nótese además que este problema no sólo afecta al servidor HTTPD de Apache: El proxy web Squid, así como un número de servidores web distintos son también vulnerables al ataque Slowloris.

Necesidades a Largo Plazo

A pesar de que nunca podremos mitigar completamente los riesgos y efectos de los ataques de denegación de servicio (en el peor de los escenarios una botnet podría enviar peticiones legítimas que consumirían sin remedio todos los recursos disponibles), sí que podemos hacer que nuestros sistemas sobrevivan a pequeños ataques, obliguen al atacante a dedicar más recursos al ataque y con suerte le disuadan de su intención de atacarnos. La mejor solución a largo plazo parece consistir en integrar en las aplicaciones una mejor funcionalidad de limitación de la tasa y, lo que es más importante aún, permitir que estas aplicaciones modifiquen su configuración bajo demanda al ser atacadas (por ejemplo, reduciendo el timeout de la conexión cuanto más ocupadas estén y comenzando a rechazar a los clientes lentos cuando se pasan del

tiempo máximo establecido). De esta forma permitimos a las aplicaciones sobrevivir no sólo a ataques de denegación de servicio, sino también a picos de carga elevados. A modo de ejemplo, en el momento de conocerse la noticia de la muerte de Michael Jackson, el sitio web de *CNN.com* parecía ligeramente roto, pero aún así seguía cargando el texto de la página, por lo que no llegó a quedar completamente inservible.

Un ejemplo puede ser el parche de Andreas Krennmair [4]. Este parche añade una monitorización del porcentaje de carga a través del marcador HTTPD de Apache. Conforme la carga aumenta, se ajusta el timeout. A un 60% de carga, reduce el timeout a la mitad; al 70%, lo reduce a un cuarto, y así sucesivamente. Aunque simplista, es un buen ejemplo de cómo dotar de un poco de inteligencia, así como de un instinto de “supervivencia”, a la aplicación; por desgracia, no sirve para cerrar conexiones ya establecidas, por lo que un atacante que cuente con los suficientes recursos aún podrá hacer que la máquina deje de responder.

Hasta el Infinito y Más Allá

La verdadera ironía de los ataques de denegación ralentizada de servicios que se apropian de los recursos encargados de la gestión de la conexión, es que en la mayoría de los casos no provocan una ralentización del servidor; simplemente evitan que los clientes legítimos puedan conectar al servidor al provocar el agotamiento de las conexiones disponibles. Y si alguna conexión volviera a estar disponible, el atacante podría tratar de recuperarla mediante técnicas agresivas aventajándose sobre los clientes legítimos. El beneficio adicional de modificar las aplicaciones para que puedan lidiar con los ataques de denegación de servicios es que también podrán gestionar cargas mayores de tráfico legítimo; algo que todos ansiamos. ■

RECURSOS

- [1] Saludo TCP de tres vías: http://en.wikipedia.org/wiki/Transmission_Control_Protocol
- [2] DoS http Slowloris: <http://ha.ckers.org/slowloris/>
- [3] Trucos de Seguridad para Apache: http://httpd.apache.org/docs/trunk/misc/security_tips.html#dos
- [4] Parche Anti-Slowloris para Apache HTTPD: <http://synflood.at/tmp/anti-slowloris.diff>