Laboratorio de Packet Sniffing y Spoofing

1 Descripción General

Los conceptos de Packet Sniffing y Packet Spoofing, son dos de lo más importantes en el campo de la seguridad de redes; ambos representan las mayores amenazas en las comunicaciones de redes. Poder entender estas amenazas es esencial para comprender las medidas de seguridad en las redes. Existen muchas herramientas para hacer sniffing y spoofing tales como `Wireshark`, `Tcpdump`, `Netwox`, `Scapy`, etc. Algunas de ellas son ampliamente usadas por expertos en seguridad como así también por atacantes. Saber usar estas herramientas es algo importante para los estudiantes, pero lo que es más importante para ellos en un curso de seguridad de redes es entender su funcionamiento, es decir como son implementados el sniffing y el spoofing de paquetes en el software.

El objetivo de este laboratorio es doble: El primero es aprender el uso de estas herramientas y conocer las tecnologías subyacentes de las mismas. El segundo será escribir un programa simple para hacer sniffing y spoofing de paquetes y así obtener un conocimiento profundo y detallado de los aspectos técnicos de estos programas. Este laboratorio cubre los siguientes tópicos:

- Como funciona el Packet Sniffing y Spoofing
- Usar la librería `pcap` y Scapy para hacer Packet sniffing
- Usando Raw Sockets y Scapy para hacer Packet spoofing
- Manipulación de paquetes usando Scapy

Lecturas y Videos. Para una cobertura más detallada sobre Sniffing y Spoofing puede consultar

- Capítulo 15 del Libro de SEED, *Computer & Internet Security: A Hands-on Approach*, 2nd Edition, by Wenliang Du. Para más detalles `https://www.handsonsecurity.net`.
- Sección 2 del curso de SEED en Udemy, *Internet Security: A Hands-on Approach*, by Wenliang Du. Para más detalles `https://www.handsonsecurity.net/video.html`.

Entorno de Laboratorio. Este laboratorio ha sido testeado en nuestra imagen pre-compilada de una VM con Ubuntu 20.04, que puede ser descargada del sitio oficial de SEED. Sin embargo, la mayoría de nuestros laboratorios pueden ser realizados en la nube para esto Ud. puede leer nuestra guía que explica como crear una VM de SEED en la nube.

Nota para los intructores. En este Laboratorio existen dos Sets de Tareas. El primer set se centra en el uso de herramientas para hacer packet sniffing y spoofing. Esto sólo requiere un conocimiento mínimo en Python; los estudiantes no necesitan tener un background previo de programación en Python. El segundo set de tareas está pensado para estudiantes de Ciencias de la Computación/Ingeniería. Los estudiantes deberán escribir sus propios programas desde cero usando el lenguaje C para llevar a cabo el sniffing y el spoofing

de paquetes. De esta forma podrán entender en mayor profundidad como es el funcionamiento de un sniffer y de un spoofer. Para abordar esta tarea, es necesario que los estudiantes tenga un sólido conocimiento en el lenguaje de programación C. Ambos Sets de Tareas son independientes; los instructores en base al background de programación que tengan sus estudiantes pueden elegir asignar un sólo Set o ambos.

2 Configuración del entorno usando Contenedores

En este laboratorio, usaremos dos máquinas conectadas a la misma LAN. Podemos usar dós Máquinas Virtuales o dos Contenedores. La Figura 1 describe la configuración del entorno de laboratorio utilizando Contenedores. Realizaremos todos los ataques desde el Contenedor del atacante, mientras que el otro contenedor será la máquina del usuario.

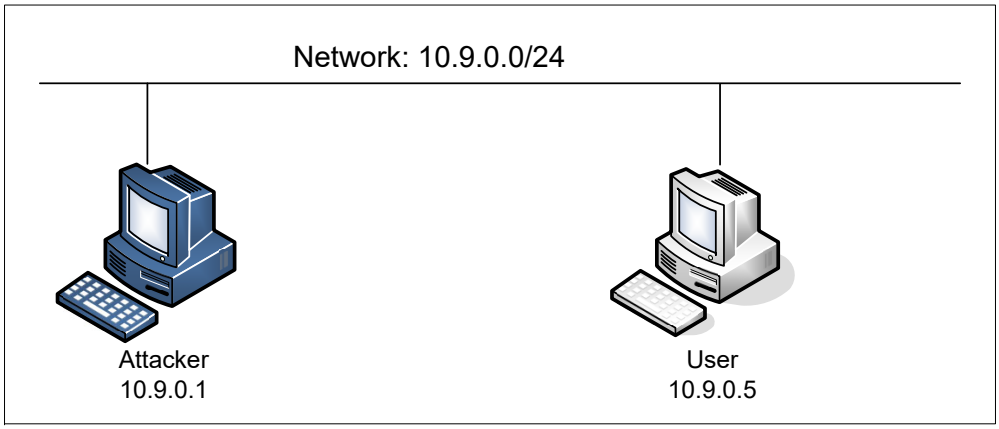

Figure 1: Configuración del entorno

2.1 Setup del Contenedor y sus Comandos

Para empezar a preparar el contenedor, deberá descargarse el archivo `Labsetup.zip` ubicado en el laboratorio correspondiente dentro del sitio web oficial y copiarlo dentro de la Máquina Virtual prevista por SEED. Una vez descargado deberá descomprimirlo y entrar dentro del directorio `Labsetup` donde encontrará el archivo `docker-compose.yml` que servirá para setear el entorno de laboratorio. Para una información más detallada sobre el archivo `Dockerfile` y otros archivos relacionados, puede encontrarla dentro del Manual de Usuario del laboratorio en uso, en el sitio web oficial de SEED.

Si esta es su primera experiencia haciendo el setup del laboratorio usando contenedores es recomendable que lea el manual anteriormente mencionado.

A continuación, se muestran los comandos más usados en Docker y Compose. Debido a que estos comandos serán usados con mucha frecuencia, hemos creados un conjunto de alias para los mismos, ubicados en del archivo `.bashrc` dentro de la Máquina Virtual provista por SEED (Ubuntu 20.04)

```
$ docker-compose build  # Build the container image
$ docker-compose up     # Start the container
$ docker-compose down   # Shut down the container

// Aliases for the Compose commands above
$ dcbuild       # Alias for: docker-compose build
```

```
$ dcup          # Alias for: docker-compose up
$ dcdown        # Alias for: docker-compose down
```

Dado que todos los contenedores estarán corriendo en un segundo plano. Necesitamos correr comandos para interactuar con los mismos, una de las operaciones fundamentales es obtener una shell en el contenedor. Para este propósito usaremos "`docker ps`" para encontrar el ID del contenedor deseado y ingresaremos "`docker exec`" para correr una shell en ese contenedor. Hemos creado un alias para ello dentro del archivo `.bashrc`

```
$ dockps        // Alias for: docker ps --format "{{.ID}}  {{.Names}}"
$ docksh <id>   // Alias for: docker exec -it <id> /bin/bash

// The following example shows how to get a shell inside hostC
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// Note: If a docker command requires a container ID, you do not need to
//       type the entire ID string. Typing the first few characters will
//       be sufficient, as long as they are unique among all the containers.
```

En caso de problemas configurando el entorno, por favor consulte la sección “Common Problems” en el manual ofrecido por SEED.

2.2 El Contenedor del Atacante

Para este laboratorio, podemos usar la Máquina Virtual o el Contenedor como la máquina del Atacante. Si hecha un vistazo al archivo de Docker Compose, verá que el contenedor del Atacante está configurado de manera diferente del resto de los contenedores. Estas son las diferencias:

- *Directorio Compartido.* Cuando usemos el contenedor del atacante para realizar los ataques, necesitamos poner el código de ataque dentro del contenedor.

  La edición del código es más conveniente dentro de la Máquina Virtual que dentro del contenedor, ya que podemos usar nuestro editor de texto preferido. Para que la Máquina Virtual y el contenedor puedan compartir archivos, hemos creado un directorio compartido entre ambos para esto hemos usado `volumes` de Docker. Dentro del archivo de Docker Composer, encontrará que se ha agregado esta entrada en algunos de los contenedores. Esta entrada indica que se montará el diretorio `./volumes` en la Máquina Host (es decir nuestra Máquina Virtual) y se podrá usar dentro del contenedor. Escribiremos nuestro código dentro del directorio `./volumes` (en la Máquina Virtual) y este podrá ser usado en el contenedor.

  ```
  volumes:
        - ./volumes:/volumes
  ```

- *Modo Host.* En este laboratio, el atacante va a necesitar sniffear los paquetes, pero correr el programa de sniffing dentro del contenedor del atacante tiene sus inconvenientes, ya que el contenedor está atachado a un switch virtual y sólo podrá ver su propio tráfico y no el del resto de los contenedores.

Para solucionar este problema, usaremos el modo `host` para el contenedor del atacante. Esto permite que el contenedor del atacante vea el tráfico de toda la red. La siguiente entrada es usada para el contenedor del atacante:

```
network_mode: host
```

Cuando un contenedor está en modo `host`, este puede ver todas las interfaces de red de los hosts que la componen, inclusive tiene la misma dirección IP como si fuera el host principal. Básicamente es ponerlo en el mismo espacio de red como si fuera la Máquina Virtual de Host. Sin embargo, el contenedor sigue siendo una máquina diferente.

Obteniendo el nombre de la interfaz de red. Cuando usamos el archivo de Compose para generar los contenedores para el laboratorio, se crea una nueva red que conecta a la Máquina Virtual y a los contenedores. El prefijo de esta red es `10.9.0.0/24` y se configura dentro del archivo `docker-compose.yml`. La dirección IP que se asigna a nuestra Máquina Virtual es `10.9.0.1`. Vamos a necesitar encontrar el nombre de cada una de las interfaces de red en nuestra Máquina Virtual ya que las utilizaremos en nuestros programas. El nombre de la interfaz es la resultante de la concatenación de `br-` y el ID de red creado por Docker. Cuando usamos `ifconfig` para listar las interfaces de red, veremos algunas de ellas. Observe la dirección IP `10.9.0.1`.

```
$ ifconfig
br-c93733e9f913: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        ...
```

Otra forma de obtener el nombre de la interfaz es usar el comando "`docker network`" y encontrar el id de red por nuestra cuenta (el nombre de la red es `seed-net`)

```
$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
a82477ae4e6b        bridge              bridge              local
e99b370eb525        host                host                local
df62c6635eae        none                null                local
c93733e9f913        seed-net            bridge              local
```

3 Set 1 de Tareas: Usando Scapy para el Sniffing y el Spoofing de Paquetes

Se pueden utilizar muchas herramientas para hacer sniffing y spoofing pero la mayoría ofrece funciones limitadas. Scapy es diferente: puede ser usada no sólo como una herramienta sino también como módulo para construir herramientas de sniffing y spoofing es decir podemos integrar funcionalidades de Scapy en nuestros propios programas. En este Set de Tareas usaremos Scapy.

Para usar Scapy, podemos escribir un programa en Python y ejecutarlo usando este interpréte. Vea el siguiente ejemplo. Debemos correr este programa de Python usando privilegios de root, dado que son necesarios para realizar operaciones de spoofing sobre paquetes. Al comienzo del programa (Línea ①), importamos todos los módulos de Scapy.

```
# view mycode.py
#!/usr/bin/env python3

from scapy.all import *    ①
```

```
a = IP()
a.show()

# python3 mycode.py
###[ IP ]###
  version   = 4
  ihl       = None
  ...


// Make mycode.py executable (another way to run python programs)
# chmod a+x mycode.py
# mycode.py
```

Podemos utilizar el modo interactivo de Python y correr nuestro programa línea a línea. Esto puede ser más conveniente si necesitamos cambiar de forma frecuente nuestro código en un experimento.

```
# python3
>>> from scapy.all import *
>>> a = IP()
>>> a.show()
###[ IP ]###
  version   = 4
  ihl       = None
  ...
```

3.1 Tarea 1.1: Sniffing de Paquetes

Una de las herramientas más populares para hacer sniffing es Wireshark, aprender a usar esta herramienta es sencillo y la usaremos durante todo el laboratorio. Sin embargo es difícil usar Wireshark para integrarlo a nuestras aplicaciones y crear módulos con diversas funcionalidades. Para este propósito usaremos Scapy. El objetivo de esta Tarea es aprender a usar Scapy para hacer sniffing de paquetes a través de programas de Python. A continuación se da un ejemplo:

```
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
  pkt.show()

pkt = sniff(iface='br-c93733e9f913', filter='icmp', prn=print_pkt)
```

El código mostrado anteriormente, hará sniffing de todos los paquetes en la interfaz `br-c93733e9f913`. Por favor lea las instrucciones en la sección del Setup del Contenedor para saber como obtener esta interfaz. Si queremos hacer sniffing en más de una interfaz, podemos usar una lista de python y asignarlas a una variable, en este caso la llamaremos `iface`. A continuación se da un ejemplo:

```
iface=['br-c93733e9f913', 'enp0s3']
```

Tarea 1.1A. En el programa anterior, por cada paquete capturado, la función `print_pkt()` será invocada como un callback; esta función mostrará información detallada del paquete. Ejecute el programa con privilegios de root y demuestre como captura los paquetes. Después de eso corra el programa sin privilegios de root; Describa y explique sus observaciones.

```
// Make the program executable
# chmod a+x sniffer.py

// Run the program with the root privilege
# sniffer.py

// Switch to the "seed" account, and
// run the program without the root privilege
# su seed
$ sniffer.py
```

Tarea 1.1B. Generalmente, cuando hacemos sniffing de paquetes, estamos interesados en ciertos tipos de paquetes y no en todos. Para ello usaremos filtros al hacer el sniffing, estos filtros provistos por Scapy usan la sintáxis BPF (Berkeley Packet Filter); Puede encontrar más información sobre BPF en el manual oficial. Por favor use los siguientes filtros mencionados a continuación y corra el programa nuevamente (Cada filtro debe ser seteado en forma separada):

- Capturar solamente paquetes ICMP.
- Capturar cualquier paquete TCP que provenga de una IP y cuyo puerto destino sea el puerto 23.
- Capturar paquetes que vienen o se dirigen hacia una subnet en particular. Puede elegir cualquier subnet como `128.230.0.0/16`; no debería de elegir la subnet a la cual su Máquina Virtual está atachada.

3.2 Tarea 1.2: Spoofing de Paquetes ICMP

Como cualquier herramienta de spoofing, Scapy nos permite establecer los valores de los campos de un paquete IP . El objetivo de esta Tarea consiste en spoofear paquetes IP con una dirección IP origen arbitraria. Haremos el spoofing sobre paquetes ICMP echo request , y los enviaremos hacia otra Máquina Virtual en la misma red. Usaremos Wireshark para ver si nuestro paquete es aceptado por la máquina destino. Si este paquete es aceptado, se enviará un paquete ICMP echo reply a la dirección IP de origen. El código a continuación muestra un ejemplo de como spoofear paquetes ICMP.

```
>>> from scapy.all import *
>>> a = IP()              ①
>>> a.dst = '10.0.2.3'    ②
>>> b = ICMP()            ③
>>> p = a/b               ④
>>> send(p)               ⑤
.
Sent 1 packets.
```

La línea ① del código mostrado anteriormente, crea un objeto IP usando la clase IP; para cada campo del encabezado IP la clase define un atributo. Podemos usar `ls(a)` o `ls(IP)` para ver todos los atributos en forma clave/valor que están disponibles para esa clase. También se puede usar a.show() y IP.show() con

el mismo fin. La línea ② muetra como setear el campo que corresponde a la IP de destino del objeto IP. Si este campo no recibe ningún valor, se usará un valor por defecto.

```
>>> ls(a)
version    : BitField (4 bits)       = 4               (4)
ihl        : BitField (4 bits)       = None            (None)
tos        : XByteField              = 0               (0)
len        : ShortField              = None            (None)
id         : ShortField              = 1               (1)
flags      : FlagsField (3 bits)     = <Flag 0 ()>     (<Flag 0 ()>)
frag       : BitField (13 bits)      = 0               (0)
ttl        : ByteField               = 64              (64)
proto      : ByteEnumField           = 0               (0)
chksum     : XShortField             = None            (None)
src        : SourceIPField           = '127.0.0.1'     (None)
dst        : DestIPField             = '127.0.0.1'     (None)
options    : PacketListField         = []              ([])
```

La línea ③ crea un objeto ICMP. El tipo por defecto que usa este objeto es echo request. En la línea ④, se apilarán `a` y `b` para formar un nuevo objeto. Esto realiza a través de la sobrecarga del operador `/` que implementa la clase IP, este operador no representa una división en este caso; este hará que se agregue `b` como el campo de payload de `a`, modificando este campo de forma transparente al programador. Como resultado se obitene un nuevo objeto que representa un paquete ICMP. En la línea ⑤ se envía este paquete usando `send()`. Por favor realice los cambios necesarios en el código anterior y demuestre que puede spoofear un paquete ICMP del tipo echo request usando una dirección IP origen arbitraria.

3.3 Tarea 1.3: Traceroute

El objetivo de esta Tarea es usar Scapy para estimar la distancia, en términos de número de routers entre su Máquina Virtual y un destino que elija. Está función es la que implementa una herramienta como `traceroute`. En este tarea, escribiremos nuestra propia herramienta. La idea es sencilla: sólo hay que enviar cualquier tipo de paquete a un destino con el valor del campo Time-To-Live (TTL) seteado en 1 como primer paso. Este paquete será dropeado por el primer router que enviará un ICMP error message, avisándonos que se ha superado el time-to-live. Así es como obtenemos la dirección IP de nuestro primer router. El paso siguiente será incrementar el valor del campo time-to-live a 2, repetir el proceso y obtener la dirección IP del segundo router. Repetiremos este proceso hasta que nuestro paquete llegue al destino deseado. Cabe aclarar que en este experimento sólo obtendremos un resultado aproximado, en teoría no todos los paquetes que se envían siguen el mismo camino (pero en la práctica, podrían hacerlo por un breve período de tiempo). El código a continuación muestra el procedimiento:

```
a = IP()
a.dst = '1.2.3.4'
a.ttl = 3
b = ICMP()
send(a/b)
```

Si es un programador experimentado de Python, puede escribir su herramienta para realizar todo el procedimiento de forma automática. Si no lo es puede hacerlo cambiando manualmente el campo TTL en cada ronda de ejecución, y registrar la dirección IP basado su observación de Wireshark. Cualquier tipo de implementación es aceptable siempre y cuando se consiga el objettivo.

3.4 Tarea 1.4: Sniffing y luego Spoofing

En esta Tarea, combinará las técnicas de sniffing y spoofing para implementar un programa de sniff-and-then-spoof. Necesitará correr dos máquinas en la misma LAN: la Máquina Virtual y un contenedor de usuario. Desde el contenedor de usuario hará un `ping` a una IP X. Esto generará un paquete ICMP echo request. Si X está vivo, el programa `ping` recibirá un echo reply y mostrará la respuesta. Su programa de sniff-and-then-spoof estará corriendo en la Máquina Virtual, que monitorea la LAN a través del paquet sniffing. Cada vez que vea un ICMP echo request, sin importar cual sea la dirección IP o de donde provenga, el programa debería de enviar un echo reply usando la técnica de packet spoofing. Además, sin importar si la máquina X esté viva o no, el programa `ping` siempre recibirá una respuesta que indique que X está viva. Para llevar a cabo esta Tarea necesitará usar Scapy. En su informe de laboratorio deberá poner la evidencia que demuestre que su técnica funciona.

En su experimento, debería de hacer `ping` a las siguientes direcciones IP del contenedor de usuario. Reporte su observación y explique los resultados.

```
ping 1.2.3.4     # a non-existing host on the Internet
ping 10.9.0.99   # a non-existing host on the LAN
ping 8.8.8.8     # an existing host on the Internet
```

Pista: Para poder explicar correctamente sus observaciones, debería de entender como funciona el protocolo ARP. También necesita saber un poco sobre routing. El comando a continuación lo ayudará a encontrar el router para un destino específico:

```
ip route get 1.2.3.4
```

4 Set 2 de Tareas: Escribiendo programas para hacer sniffing y spoofing

Para este conjunto de Tareas, debe compilar código en C en la Máquina Virtual Host y correrlo dentro del contenedor.

4.1 Tarea 2.1: Escribiendo el programa para hacer Packet Sniffing

Los Sniffers pueden ser escritos usando la librería `pcap`. Usar `pcap` facilita la tarea del desarrollo de un sniffer ya que implica invocar una serie de funcione nativas de la librería. Al final de cada secuencia y a medida que vayan siendo capturados, los paquetes serán colocados en un buffer para un procesamiento posterior. Todos los detalles de la captura de paquetes son manejados por la librería `pcap`. El libro de SEED ofrece un código de ejemplo, que muestra como escribir un sniffer simple usando `pcap`. A continuación el código de ejemplo (Vea el libro para una explicación más detallada):

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


/* This function will be invoked by pcap for each captured packet.
   We can process each packet inside the function.
 */
void got_packet(u_char *args, const struct pcap_pkthdr *header,
        const u_char *packet)
```

```
{
   printf("Got a packet\n");
}

int main()
{
  pcap_t *handle;
  char errbuf[PCAP_ERRBUF_SIZE];
  struct bpf_program fp;
  char filter_exp[] = "icmp";
  bpf_u_int32 net;

  // Step 1: Open live pcap session on NIC with name eth3
  //         Students needs to change "eth3" to the name
  //         found on their own machines (using ifconfig).
  handle = pcap_open_live("eth3", BUFSIZ, 1, 1000, errbuf);

  // Step 2: Compile filter_exp into BPF psuedo-code
  pcap_compile(handle, &fp, filter_exp, 0, net);
  if (pcap_setfilter(handle, &fp) !=0) {
      pcap_perror(handle, "Error:");
      exit(EXIT_FAILURE);
  }

  // Step 3: Capture packets
  pcap_loop(handle, -1, got_packet, NULL);

  pcap_close(handle);   //Close the handle
  return 0;
}


// Note: don't forget to add "-lpcap" to the compilation command.
// For example: gcc -o sniff sniff.c -lpcap
```

Tim Carstens ha escrito un tutorial que explica como usar la librería `pcap` para escribir un sniffer. El tutorial puede ser accedido a tráves de este link `http://www.tcpdump.org/pcap.htm`

Tarea 2.1A: Entendiendo como funciona un Sniffer En esta Tarea, los estudiantes deberán escribir un sniffer para mostrar la dirección IP de origen y destino de cada paquete capturado. Los estudiantes pueden descargarse el código de ejemplo del libro de SEED (`https://www.handsonsecurity.net/figurecode.html`). Los estudiantes deberán de proveer screenshots que muestren que su programa sniffer corre bien y este produciendo los resultados esperados. Por favor responda las siguientes preguntas:

- **Pregunta 1.** Por favor explique con sus propias palabras la secuencia de llamados que realiza la librería y que son esenciales para el sniffer. Esta explicación no debe ser tan detallada como la del libro o el tutorial.
- **Pregunta 2.** ¿Por qué se necesitan privilegios de root para correr el sniffer? ¿En dónde falla el programa si es ejecutado sin privilegios de root?

- **Pregunta 3.** Por favor active y desactive el modo promiscuo en su sniffer. ¿Puede mostrar la diferencia cuando este modo está activado y desactivado? Por favor describa como puede demostrar esto.

Tarea 2.1B: Escribiendo Filtros. Por favor escriba filtros en su sniffer. Puede usar los manuales que están en la red para obtener más información sobre los filtros en `pcap`. En su informe de laboratorio, incluya screenshots que muestren los resultados después de aplicar cada uno de los filtros que se piden a continuación.

- Capturar paquetes ICMP entre dos hosts diferentes.
- Capturar paquetes TCP cuyo puerto destino este en el rango de 10 a 100.

Tarea 2.1C: Sniffing Passwords. Por favor muestre como puede usar sus sniffer para capturar passwords cuando alguien use `telnet` en la red que está monitoreando. Es posible que necesite modificar el código de su sniffer para mostrar la parte de los datos que son capturados en el paquete TCP (`telnet` usa TCP). Puede imprimir toda la parte de los datos y marcar manualmente donde se encuentra el password (o parte de este).

4.2 Tarea 2.2: Spoofing

Cuando un usuario no privilegiado envía un paquete, los sistemas operativos usualmente no le permiten establecer los valores para todos los campos que componen los encabezados de un paquete que son parte de un protocolo (como TCP, UDP y IP). Los sistemas operativos se encargarán de llenar estos campos, permitiendo que este tipo de usuarios puedan establecer valores sólamente para unos pocos campos (dirección IP de destino, puerto de destino, etc). Sin embargo, si el usuario tiene privilegios de root, puede establecer valores para cualquier campo en los encabezados del paquete. Esto es llamado packet spoofing y se puede hacer usando *raw sockets*.

Los Raw sockets le dan al programador absoluto control sobre la construcción de un paquete, permitiéndole crear cualquier tipo de paquete arbitrario, esto incluye establecer los valores de su encabezados y su payload. El uso de los raw sockets es bastante simple; involucra cuatro pasos: (1) crear el raw socket, (2) configurar sus opciones, (3) construir el paquete, (4) enviar el paquete a través del raw socket. Existen muchos tutoriales online que enseñan como programar raw sockets en C. Hemos agregado alguno de ellos en la página oficial del laboratorio. Por favor leálos y aprenda a escribir un programa para hacer spoofing de paquetes. A continuación se brinda un código genérico de un spoofer que deberá de ser completado:

```
int sd;
struct sockaddr_in sin;
char buffer[1024]; // You can change the buffer size

/* Create a raw socket with IP protocol. The IPPROTO_RAW parameter
 * tells the sytem that the IP header is already included;
 * this prevents the OS from adding another IP header.  */
sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
if(sd < 0) {
    perror("socket() error"); exit(-1);
}

/* This data structure is needed when sending the packets
 * using sockets. Normally, we need to fill out several
 * fields, but for raw sockets, we only need to fill out
```

```
 * this one field */
sin.sin_family = AF_INET;

// Here you can construct the IP packet using buffer[]
//    - construct the IP header ...
//    - construct the TCP/UDP/ICMP header ...
//    - fill in the data part if needed ...
// Note: you should pay attention to the network/host byte order.


/* Send out the IP packet.
 * ip_len is the actual size of the packet. */
if(sendto(sd, buffer, ip_len, 0, (struct sockaddr *)&sin,
              sizeof(sin)) < 0) {
      perror("sendto() error"); exit(-1);
}
```

Tarea 2.2A: Escriba un programa de spoofing. Por favor escriba su propio programa de spoofing en C. Para demostrar que su programa funciona y realiza el spoofing de paquetes IP, deberá de proveer evidencias (Por ejemplo, un packet trace usando Wireshark).

Tarea 2.2B: Haga Spoof de un ICMP Echo Request. Haga spoof de un paquete ICMP echo request en nombre de otra máquina (es decir, use la dirección IP de otra máquina que no sea la suya como dirección IP origen). Este paquete debería ser enviado a una máquina remota en Internet (debe estar online). Abra el Wireshark y si su spoofing es exitoso debería de recibir un paquete ICMP echo reply como respuesta de la máquina remota.

Preguntas. Por favor responda las siguientes preguntas.

- **Pregunta 4.** ¿Es posible modificar el campo length de un paquete IP usando cualquier valor, sin importar cuan grande sea el paquete en cuestión?
- **Pregunta 5.** Al programar con raw sockets, ¿Débe de calcular el checksum para un encabezado IP?
- **Pregunta 6.** ¿Por qué necesita privilegios de root para correr un programa que usa raw sockets? ¿Donde falla el programa si no se ejecuta con privilegios de root?

4.3 Tarea 2.3: Sniff y luego Spoof

En esta Tarea, va a combinar las técnicas de sniffing y spoofing para implementar un programa de sniff-and-then-spoof. Para lograr esto, necesita tener dos Máquinas Virtuales en la misma LAN. Desde la máquina A hará `ping` hacia una IP X. Esto va a generar un paquete ICMP echo request. Si X está viva, el programa de `ping` recibirá una echo reply y mostrará la respuesta. Su programa de sniff-and-then-spoof correrá en la Máquina B que va a monitorear la LAN usando packet sniffing. Cada vez que vea un paquete ICMP echo request, sin importar cual es la dirección IP destino, el programa deberá de enviar inmediatamente un paquete ICMP echo reply usando la técnica de packet spoofing. Además, sin importar si la máquina X este viva o no, el programa de `ping` siempre recibirá una reply, indicando que X está viva. Su tarea es escribir un programa de estas características usando C y incluir screenshots en su informe de laboratorio que muestren que su programa funciona. Por favor también incluya el código de su programa.

5 Guías

5.1 Llenado de Datos en los Raw Packets

Cuando envía un paquete usando raw sockets, básicamente lo que se está haciendo es construir un paquete dentro de un buffer, de forma tal que cuando es necesario enviar este paquete, se le da al sistema operativo el buffer y el tamaño del paquete. Trabajar directamente sobre este buffer no es fácil, una forma habitual de trabajar con este tipo de datos es hacer typecasting del buffer (o parte del mismo) dentro de estructuras, como IP header, así puede acceder a los elementos de ese buffer usando los campos de esas estructuras. Puede definir diferentes estructuras dentro de su programa como pueden ser los headers para IP, ICMP, TCP, UDP y otras más. El siguiente ejemplo muestra como se puede construir un paquete UDP:

```
struct ipheader {
   type  field;
   .....
}

struct udpheader {
   type field;
   ......
}

// This buffer will be used to construct raw packet.
char buffer[1024];

// Typecasting the buffer to the IP header structure
struct ipheader *ip = (struct ipheader *) buffer;

// Typecasting the buffer to the UDP header structure
struct udpheader *udp = (struct udpheader *) (buffer
                            + sizeof(struct ipheader));

// Assign value to the IP and UDP header fields.
ip->field = ...;
udp->field = ...;
```

5.2 Network/Host Byte Order y Conversiones

Es necesario prestar atención a la red y su host byte order. Si usa una CPU con una arquitectura x86, el host byte order de esta arquitectura es llamado *Little Endian*, mientras que en las redes se usa *Big Endian*. Esto quiere decir que los datos que se envíen dentro de un buffer de un paquete usarán ese network byte order; si esto no sucede, el paquete no estará bien formado. Actualmente no necesita preocuparse sobre el Endianess que su máquina usa, ni sobre la portabilidad de su programa.

Lo que debe recordar siempre es convertir los datos que son colocados en el buffer al network byte order en cuestión y convertirlos al host byte order cuando son copiados del buffer a la estructura de datos en su computadora. Si el dato tiene un sólo byte, no necesita realizar la conversión, pero si el dato es `short`, `int`, `long` o es un tipo de dato que tiene más de un byte, necesitará llamar a algunas de las siguientes funciones para realizar la conversión:

```
htonl(): convert unsigned int from host to network byte order.
ntohl(): reverse of htonl().
```

```
htons(): convert unsigned short int from host to network byte order.
ntohs(): reverse of htons().
```

También puede usar `inet_addr()`, `inet_network()`, `inet_ntoa()`, `inet_aton()` para convertir direcciones IP con notación decimal con puntos (cadena) en un entero de 32-bits network/host byte order. Puede consultar manuales en internet sobre estas funciones.

6 Informe del Laboratorio

Debe enviar un informe de laboratorio detallado, con capturas de pantalla, para describir lo que ha hecho y lo que ha observado. También debe proporcionar una explicación a las observaciones que sean interesantes o sorprendentes. Enumere también los fragmentos de código más importantes seguidos de una explicación. No recibirán créditos aquellos fragmentos de códigos que no sean explicados.

Agradecimientos

Este documento ha sido traducido al Español por Facundo Fontana