REDES DE COMUNICACIONES I

Introducción a libpcap y Wireshark

Volver a: Prácticas

Objetivos de la práctica

- Familiarizarse con la biblioteca libpcap: esta biblioteca nos permite capturar paquetes, filtrarlos, modificarlos y estudiarlos desde un programa C o un script de Python. Puede interpretarse que Wireshark sirve como front-end de esta biblioteca*.
- Demostrar que se ha realizado el tutorial de Wireshark con provecho.

Introducción

Existen multitud de recursos en Internet para aprender a usar libpcap en detalle:

- Packet Capture With libpcap and other Low Level Network Tricks de NAU's Computer Systems Engineering.
- Aprendiendo a programar con Libpcap de Alejandro Lopez Monge.

Las referencias proporcionadas abordan el uso de la libpcap desde programas en lenguaje C. En las prácticas usaremos un *wrapper* para Python llamado RC1-libpcap que nos encapsulará el acceso a las funciones de la biblioteca libpcap que está escrita en C. Este *wrapper* respeta los nombres de funciones, argumentos, retornos y tipos de datos de la biblioteca original y puede usarse de manera equivalente.

Veamos primero las funciones básicas que nos ofrece RC1-libpcap.

(**IMPORTANTE** Por favor, lea el enunciado entero antes de empezar.)

Biblioteca RC1-Libpcap

Abrir un archivo pcap

Para abrir un archivo (traza) previamente capturado:

pcap_open_offline(fname,errbuf)

Donde

- *fname* es una cadena con el nombre del archivo .pcap que se desea abrir.
- *errbuf* es un *bytearray* donde se guardará un mensaje de error en caso de que algo haya fallado al abrir el fichero
- La función nos devuelve un descriptor al archivo .pcap o None en caso de que haya algún error

Ejemplo: p = pcap_open_offline('traza.pcap', errbuf)

Abre para lectura el archivo traza.pcap. En caso de error, guarda el mensaje en el *bytearray* errbuf.

Capturar de un interfaz

Para abrir un interfaz para captura:

pcap_open_live(device,snaplen,promisc,to_ms, errbuf)

Donde

- *device* es una cadena con el nombre de la interfaz que se quiere abrir (eth0, eth1...).
- *snaplen* es la cantidad de bytes que se quieren guardar por cada paquete. Es útil cuando no nos interesa la carga útil del paquete y así reduciríamos el tamaño de la captura.

- *promisc* indica si queremos abrirla en modo promiscuo (promisc=1) o no (promisc=0).
- *to_ms* duración del *timeout* de lectura. Tiempo que se espera para leer varios paquetes en una misma transacción (*polling*).
- e*rrbuf* es un *bytearray* donde se guardará un mensaje de error en caso de que algo haya fallado al abrir la interfaz.
- La función nos devuelve un descriptor al archivo .pcap o None en caso de que haya algún error

Para abrir una interfaz es necesario tener permisos de superusuario. En la VM facilitada simplemente debemos usar sudo seguido de la instrucción.

Ejemplo: p = pcap_open_live('eth0',BUFSIZ,0, 100, errbuf)

Abre la interfaz eth0 en modo no promiscuo, capturando el paquete en su totalidad, con un timeout de lectura de 100 ms. (puede que la VM le alerte sobre la imposibilidad de capturar tráfico de modo promiscuo en caso de modificar el tercer argumento, no es importante para la realización de las prácticas, acepte y continúe). En caso de error, guarda el mensaje en el *bytearray* errbuf.

Leer tráfico de archivo o interfaz

pcap_loop(descr, cnt, callback, user)

Donde:

- *descr* es el descriptor PCAP del que queramos leer (que anteriormente hemos abierto con open_pcap_live o open_pcap_offline).
- *cnt* es el número de paquetes a analizar (-1 para ilimitados).
- *callback* es una función de atención al paquete. Esta función se ejecutará por cada paquete leído o capturado.
- user es una variable auxiliar que sirve para pasar datos a la función de atención.
- *pcap_loop* nos devuelve:
  - 0 si se leyó la traza entera o se supero el límite cnt.
  - -1 Si hubo errores
  - -2 Si fue interrumpido por pcap_breakloop() (u otras).
  - Otros valores si se capturó un paquete.

Ejemplo: *ret = pcap_loop (descr,-1,procesa_paquete,None)*

Y a su vez: *callback (user,pkt_header,pkt_data)*

Donde:

- *user: son los datos auxiliares de usuario que se han pasado a pcap_loop*
- *pkt_header:* es un objeto de tipo pcap_pkthdr que contiene la cabecera pcap del paquete leído o capturado.Este objeto tiene tres campos:
  - *pkt_header.ts : objeto timestamp que contiene el tiempo de captura del paquete. A su vez este objeto tiene 2 campos:*
    - *ts.tv_s*ec: timestamp del paquete en segundos.
    - *ts.tv_usec:* microsegundos dentro del segundo actual de timestamp. **OJO: Este valor nunca debe ser superior a 1,000,000**. **Esta variable NO almacena el mismo tiempo que tv_sec pero en microsegundos sino la parte fraccional del tiempo de captura.**
  - *pkt_header.len*: longitud real del paquete.
  - *pkt_header.caplen*: longitud capturada del paquete. Esto es, pkt_data solo contendrá pkt_header.caplen bytes.
- *pkt_data* es un *bytearray* que contiene los datos del paquete en caso de éxito.

- Es responsabilidad de *pcap_loop(·)*, no de la función de atención, reservar y liberar la memoria para devolver la cabecera y datos de cada paquete.

Hay otras funciones para leer paquetes no basadas en bucles, del tipo *pcap_next_ex(·)/pcap_next(·)*. **No las use de ninguna manera en las prácticas. Los motivos son de tipo docente. Su uso conllevará una evaluación nula.**

Guardar archivo pcap

Para guardar un archivo pcap necesitamos primero crear el archivo donde vamos a ir volcando los paquetes. Para ello se usan la funciones *pcap_open_dead* y *pcap_dump_open*:

pcap_open_dead(linktype, snaplen)

Donde

- *linktype* es el tipo de enlace de los paquetes que vamos a guardar. Típicamente, redes Ethernet: DLT_EN10MB
- *snaplen* es el tamaño máximo que queramos guardar de cada paquete.

Devuelve, como las otras funciones *pcap_open*, un descriptor de archivo pcap.

Ejemplo: *descr2 = pcap_open_dead(DLT_EN10MB,1514)* Abre un descriptor de archivo pcap para paquetes Ethernet, guardando como máximo 1514 Bytes de cada paquete.

pcap_dump_open(descr, fname)

Donde

- *descr* es el descriptor de archivo pcap previamente abierto con pcap_open_dead.
- *fname* es una cadena con el nombre del archivo pcap en el que queramos guardar los paquetes.

Devuelve un objeto *dumper* que se usará para guardar paquetes.

Ejemplos: *pdumper = pcap_dump_open(descr2,'salida.pcap')*

Crea un archivo llamado salida.pcap con las características (tipo de enlace, y tamaño máximo de paquete) de descr2 (que indicamos en el pcap_open_dead). Nos devuelve un objeto dumper. Para guardar paquetes en el archivo creado con pcap_dump_open usamos la función:

pcap_dump(dumper,h,sp)

- *dumper* es el *dumper* devuelto por pcap_dump_open.
- *h* es un objeto de tipo pcap_pkthdr con la cabecera pcap del paquete que vamos a guardar.
- *sp* es un bytearray con el contenido del paquete. Se van a guardar tantos bytes como indiquemos en el campo caplen del parámetro h.

Ejemplo: *pcap_dump(pdumper,h,pkt_data)*

Guardamos en pdumper el paquete apuntado por packet con cabecera h.

Cerrar archivo

Los descriptores abiertos con *pcap_open_live*, *pcap_open_offline* y *pcap_open_dead* se cierran con *pcap_close(·)*. Mientras que los archivos abiertos con *pcap_dump_open* se cierran con *pcap_dump_close(·)*.

pcap_close(descr)

Donde

- *descr* es el descriptor a cerrar

pcap_dump_close(descr)

- *descr* es el *dumper* a cerrar

Ejercicios

Se pide la entrega de dos ejercicios:

Primer ejercicio: libpcap

Se facilita un programa ejemplo en el Moodle. Descárguelo, analícelo y modifíquelo para que cumpla los requisitos definidos a continuación.

Entregue los fuentes Python (*.py) usados para implementar un programa basado en libpcap que:

1. Si se ejecuta **sin** argumentos, debe devolver ayuda de ejecución.
2. Si se ejecuta con el argumento --**itf**, consideramos que queremos capturar de interfaz:
    - El programa debe mostrar el número de paquetes recibidos por la interfaz de red especificada en el argumento tras pulsar Control-C.

- El programa debe almacenar los paquetes capturados **enteros** en una traza con nombre captura.nombreitf.FECHA.pcap (donde FECHA será el tiempo actual UNIX en segundos y nombreitf el nombre de la interfaz especificada).
- Al almacenar la traza queremos modificar la fecha de cada paquete capturado. La modificación consistirá en **sumar 30 minutos** a la fecha de captura. Ejemplo: si capturamos el día 20 de octubre a las 10:23, deberíamos observar en la traza almacenada los paquetes con fecha del 20 de octubre a las 10:53.

3. Si se ejecuta con el argumento **--file**, consideramos que queremos analizar una traza pcap. El programa debe mostrar el número de paquetes de la traza al finalizar su ejecución.
4. En ambos casos (traza o captura de interfaz/en vivo) el programa debe **mostrar** los **N** (siendo N especificado en el parámetro **--nbytes**) primeros bytes de cada paquete capturado/analizado en hexadecimal con 2 dígitos por Byte (**y separando cada Byte por espacios en blanco**).
   - Prestad atención a los límites de bytes capturados, y a paquetes más pequeños (¿los hay?).
   - Para demostrar la corrección de este tercer apartado use Wireshark: compare visualmente si la salida de su programa coincide con la salida que da Wireshark en su ventana inferior. Se espera que no haya diferencias.
   - Haga una captura de pantalla que muestre ambas salidas para una captura en vivo (*online*). Llame a esta captura de pantalla practica1captura.*, e inclúyala en la entrega.

NOTA IMPORTANTE. El programa solo debe distinguir la fuente de entrada a la hora de abrir el descriptor: NO se debe por tanto hacer dos "flujos" distintos para cada tipo de operación sino tan solo un flujo que al principio distinga de donde "sale" el tráfico sobre el que trabajar pero, a partir de ese punto, debe haber un único flujo con las mínimas variaciones posibles.

Segundo ejercicio: Wireshark

Responda al listado de preguntas en el documento "Ejercicios de captura de tráfico".

Criterios de evaluación

Ejercicios: Entrega antes de las 23:55 del 18 de octubre.

- Normativa de entrega cumplida en su totalidad: 5%
- Fichero leeme.txt bien explicado: 5%
- Contar paquetes de una traza (independientemente del número de paquetes): 10%
- Contar paquetes de la interfaz de red: 5%
- Uso de un único "flujo" para traza e interfaz: 10%
- Almacenar correctamente el tráfico capturado en vivo una traza: 10%
- Modificar fecha correctamente: 15%
- Imprimir los N primeros bytes de un paquete (pruebe para N>15) y validarlo con Wireshark (captura de pantalla): 20%
- Ejercicios de captura de tráfico: 20%

Control individual: Cuestionario sobre manejo básico de Wireshark y libpcap el día 19 de octubre. No olvide ser puntual, el control empezará a "y 5".

Entrega

Respecto al ejercicio **Libpcap**, denomine al archivo de entrega **practica1.py**.

- Añada un archivo **leeme.txt** que incluya los nombres de los autores, comentarios que se quieran transmitir al profesor y, en caso de entregar algún archivo más, la descripción y/o explicación del mismo. **Además este fichero debe contener una sección donde se determine si se ha dado respuesta (Realizado/Parcialmente-Realizado/No-Realizado, y en caso afirmativo la explicación de cómo se ha validado) a cada criterio de evaluación solicitado. Ejemplo:**
  - **Normativa de entrega cumplida en su totalidad: Realizado: Varias relecturas del enunciado.**
  - **Contar paquetes de una traza: Realizado: Se ha comprobado que el número de paquete que muestra nuestro programa coincide con el que indica Wireshark.**
  - **Contar paquetes de la interfaz de red: No-Realizado.**

- **Almacenar en una traza el tráfico capturado en vivo: Realizado: Se ha comprobado que todos los bytes de la traza capturado coincide con lo que indica Wireshark en un conjunto diverso de paquetes.**
- **Modificar fecha correctamente: No-Realizado.**
- **Imprimir los N primeros bytes de un paquete y validarlo con Wireshark (captura de pantalla): Parcialmente-Realizado: Se imprimen correctamente solamente los 5 primeros bytes.**
- **Cuestionario "Ejercicios de captura de tráfico": No-Realizado.**

Respecto al ejercicio **Wireshark**, entregue un pdf con sus respuesta con nombre **practica1.pdf**. Debe ser un pdf, **no se aceptará ningún otro formato.** Adicionalmente añada la traza que haya generado para dar respuesta a las cuestiones, y llámela practica1.pcap.

Comprima en un zip **TODO** lo que vaya a entregar y llámelo practica1_YYYY_PXX.zip, donde YYYY es el grupo al que pertenece (1301,1302,etc), y XX (y solo XX) es el número de pareja (**con dos dígitos**).

Por ejemplo, para la pareja 5 del grupo 1301: **$ zip practica1_1301_P05.zip ***

Solo es necesario que suba la entrega un miembro de la pareja.

Última modificación: jueves, 19 de septiembre de 2019, 17:12

◄ Ejemplos Python

Material P1 ►

Volver a: Prácticas