REDES DE COMUNICACIONES I

Enunciado Práctica 1, sesión 2: Libpcap y Wireshark

Volver a: Práctica 1, ses...

Objetivos de la práctica

- Familiarizarse con la biblioteca libpcap: esta biblioteca nos permite capturar paquetes, filtrarlos, modificarlos y estudiarlos desde un programa C. 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.

Veamos primero las funciones básicas que nos ofrece libpcap.

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

Biblioteca Libpcap

Abrir un archivo pcap

Para abrir un archivo (traza) previamente capturado:

*pcap_t *pcap_open_offline(const char *fname, char *errbuf);*

Donde

- *fname* es el nombre del archivo .pcap que se desea abrir.
- En *errbuf* se guarda el mensaje de error, si procede.
- La función nos devuelve el puntero al descriptor de archivo .pcap.

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

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

Capturar de un interfaz

Para abrir un interfaz para captura:

*pcap_t *pcap_open_live(const char *device, int snaplen,int promisc, int to_ms, char *errbuf);*

Donde

- *device* es 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*).
- En *errbuf* se guarda el mensaje de error, si procede.

- La función nos devuelve el puntero al descriptor de archivo .pcap.

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

Ejemplo: descr=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 la cadena errbuf.

Leer tráfico de archivo o interfaz

*int pcap_next_ex(pcap_t *descr, struct pcap_pkthdr **pkt_header, const u_char **pkt_data);*

Donde

- *descr* es el puntero a descriptor de interface/fichero del que queramos leer (que anteriormente hemos abierto con open_pcap_live o open_pcap_offline).
- *pkt_header* es la cabecera pcap del paquete a ser rellenada. Esta cabecera es una estructura con cuatro campos:
  - *pkt_header->ts.tv_se*c, timestamp del paquete en segundos.
  - *pkt_header->ts.tv_usec*, timestamp del paquete en microsegundos.
  - *pkt_header->len*: longitud real del paquete.
  - *pkt_header->caplen*: longitud capturada del paquete. Esto es, el puntero que nos devuelve sólo contiene h->caplen bytes.
- *pkt_data* es el puntero al inicio del paquete leído en caso de éxito.
- La función nos devuelve:
  - 0 si no se leyó ningún paquete durante el tiempo definido como timeout.
  - 1 si se leyó un paquete correctamente
  - -1 Si hubo errores
  - -2 Si en el caso de leer de archivo, se leyeron todos los paquetes.
- Es responsabilidad de *pcap_next_ex(·)* reservar y liberar la memoria para devolver la cabecera y datos del paquete.

Ejemplo: *ret = pcap_next_ex(descr,&pkt_header,(const u_char **)&pkt_data);*

Hay otras funciones para leer paquetes basadas en bucles del tipo *pcap_loop(·)* u otras aquí no explicadas. **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_t *pcap_open_dead(int linktype, int 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*, el puntero al 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_dumper_t * pcap_dump_open(pcap_t *descr, const char *fname);*

Donde

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

Devuelve el puntero al archivo pcap donde se van a volcar los 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 el puntero a archivo de volcado de paquetes que vamos a utilizar para guardar los paquetes. Para guardar paquetes en el archivo creado con pcap_dump_open usamos la función: *void pcap_dump(u_char *user, struct pcap_pkthdr *h,u_char *sp);*

- *user* es el puntero devuelto por pcap_dump_open.
- *h* es un puntero a la cabecera pcap del paquete que vamos a guardar.
- *sp* es el puntero al 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(·)*.

*void pcap_close(pcap_t *descr);*

Donde

- *descr* es el descriptor a cerrar

*void pcap_dump_close(pcap_dumper_t *descr);*

- *descr* es el descriptor a cerrar

Definición de tipos

Es muy importante trabajar con tipos sin signo con tamaño definido (*uint8_t*, *uint16_t*, *uint32_t*, etc... de *stdint.h*) con objeto de controlar su tamaño de forma precisa al realizar operaciones binarias.

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 C (*.c y *.h) y **makefile** 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 **un** argumento, consideramos que queremos capturar de interfaz:
   - El programa debe mostrar el número de paquetes recibidos por la interfaz de red eth0 tras pulsar Control-C.
   - El programa debe almacenar los paquetes capturados **enteros** en una traza con nombre eth0.FECHA.pcap (donde FECHA será el tiempo actual UNIX en segundos).
   - Al almacenar la traza queremos modificar la fecha de cada paquete capturado. La modificación consistirá en **sumar dos días** a la fecha de captura. Ejemplo: si capturamos el día 20 de octubre a las 10:53, deberíamos observar en la traza almacenada los paquetes con fecha del 22 de octubre a las 10:53.
3. Si se ejecuta con ***dos*** argumentos (el segundo será la traza a analizar), 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** (N es el primer argumento de ejecución) 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. El programa solo debe distinguir la fuente de entrada a la hora de abrir el descriptor: NO se debe por tanto hacer dos "hilos" 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".

Entrega

Respecto al ejercicio **Libpcap**, denomine a los archivos de entrega **practica1.c y practica1.h**. 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. **No olvide el makefile** ni la captura de pantalla solicitada.

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. En caso de dudas pueden subir ambos miembros la práctica.

Criterios de evaluación

Ejercicios: Entrega antes de las 23:55 del 28 de septiembre.

- Normativa de entrega cumplida en su totalidad: 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: 20%
- Imprimir los N primeros bytes de un paquete (pruebe para N>10) 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 29 de septiembre. No olvide ser puntual, el control empezará a "y 5".

Última modificación: viernes, 22 de septiembre de 2017, 12:02

Volver a: Práctica 1, ses...