Лекция

# Библиотека libpcap

#include <pcap.h>

*Версии libpcap реализованы для многих ОС,*
*в том числе для Windows.*
*Самая последняя версия доступна на сайте*
http://www.tcpdump.org

**Типичная последовательность шагов:**

1. Идентификация сетевого интерфейса
2. Открытие сетевого интерфейса и создание сессии перехвата
2. Создание фильтра
3. Захват и обработка пакетов
4. Закрытие сессии перехвата пакетов

# Основные функции библиотеки pcap

## Идентификация сетевого интерфейса

*Три основных способа идентификации сетевого интерфейса, на котором будет осуществляться прослушивание.*

***Первый способ*** (без использования библиотеки libpcap).
Имя интерфейса задается жестко в программе.

**#define DEVICE “eth0”**

*Или, например, передавать имя сетевого интерфейса через командную строку.*

***Второй способ*.**

**char *pcap_lookupdev(char *errbuf)**

Данная функция возвращает имя сетевого интерфейса, установленного по умолчанию; в случае ошибки возвращает NULL и в аргумент функции записывает причину ошибки.

Пример:

```
#include <pcap.h>
  char    err[PCAP_ERRBUF_SIZE]; /* буфер для записи ошибок */
  char    *device;              /* буфер, в котором будет находиться название интрефейса */
    device = pcap_lookupdev(err);
```

**int pcap_lookupnet(char *device, bpf_u_int32 *netp, bpf_u_int32 *maskp, char *errbuf)**

Функция позволяет получить номер/маску сети на сетевом интерфейсе заданным первым параметром, в случае ошибки возвращает -1 и записывает причину ошибки в последний аргумент.

(bpf_u_int32 это unsigned int)

```
 if ((pcap_lookupnet(device, &net, &mask, err)) < 0)
{ /* Обработка ошибки*/}
```

***Третий способ***

Предложить пользователю выбрать интерфейс из имеющего списка. Такой список формируется функцией

**int pcap_findalldevs(pcap_if_t **alldevsp, char *errbuf);**

— функция собирает информацию о доступных интерфейсах (сетевые адаптеры; в том числе, всевозможные инфракрасные устройства, параллельные порты, etc.), которые можно пытаться прослушивать.
Функция берет указатель на **pcap_if_t** и возвращает односвязный список об обнаруженных интерфейса . Функция pcap_findalldevs() сама выделяет нужную память под ****alldevsp**. Признак конца списка — значение NULL в очередном **pcap_if *next** из выбранного ****alldevsp**. В буфер **errbuf** будет передано описание ошибки, если таковая возникнет при работе.

Тип **pcap_if_t** (это тип, производный от **pcap_if**) представляет собой структуру, которая содержит множество информации, которая может быть полезна:

```
typedef struct pcap_if pcap_if_t;

struct pcap_if {
        struct pcap_if  *next; /* указатель на следующий элемент в
                                  списке */
        char *name;            /* имя интерфейса */
        char *description;     /* текстовое описание интерфейса или
                                  NULL */
struct pcap_addr *addresses; /* IP-адрес, маска сети,
                                широковещательный адрес и пр. */
        bpf_u_int32 flags;     /* равно PCAP_IF_LOOPBACK для
                                  интерфейса обратной связи
                                  loopback interface) */
};
```

Элемент ***addresses*** является указателем на структуру **pcap_addr**, которая содержит дополнительную информацию об интерфейсе:

```
struct pcap_addr {
        struct pcap_addr *next;        /* указатель на следующий
                                          элемент в списке */
        struct sockaddr *addr;         /* IP-адрес */
        struct sockaddr *netroask;     /* маска сети для этого IP-адреса*/
        struct sockaddr *broadaddr;    /* широковещательный адрес */
         struct sockaddr *dstaddr;     /* адрес приемной стороны для
                                          соединения типа точка-точка
                                           point-to-point ) или NULL */
};
```

Функция **pcap_findalldevs()** отсутствует в устаревших версиях библиотеки **libpcap**.

# Открытие сетевого интерфейса и создание сессии перехвата

**pcap_t *pcap_open_live(char *device, int snaplen, int promisc, int to_ms, char *ebuf)**

Функция возвращает дескриптор открытого сетевого интрефейса. Что делается с полученным? Происходит перехват пакетов с заданного первым аргументом, второй аргумент это максимальная длинна обрабатываемого пакета, обычно данное значение равняется MTU на открываемом интерфейсе.
Следующий аргумент задает: переводить ли интерфейс в PROMISC режим работы (1 - да, 0 - нет).
Идущий далее параметр задает таймаут чтения с сетевого интерфейса в миллисекундах.
Последний параметр - это буфер, куда, при возникновении ошибок, будут записываться причины (при ошибке функция возвращает NULL).

```
 if ((fd = pcap_open_live(device, MAXLEN, 1, TIMEOUT, err)) == NULL)
  {
    printf("Cant open device %s\n", err);
    exit(0);
  }
```

Тип данных **pcap_t** это структура:

```
struct pcap {
        int fd;                 /* видимо реальный дескриптор интерфейса */
        int snapshot;
        int linktype;       /* тип интерфейса, можно и не использовать, pcap_datalink,
                                а узнать тип интерфейса, просто pcap_t->linktype */
        int tzoff;              /* оффсет временой зоны */
        int offset;             /* оффсет для правильной регулировки (чего?) */
        struct pcap_sf sf;
        struct pcap_md md;
        int bufsize;
        u_char *buffer;
        u_char *bp;
        int cc;
        u_char *pkt;            /* указатель на следующий пакет */
        struct bpf_program fcode; /* структура для регулярного выражения
                                     bpf, если bpf не встроен в ядро */
        char errbuf[PCAP_ERRBUF_SIZE]; /* буфер для хранения информации об
ошибках */
};
```

**int pcap_stats(pcap_t *p, struct pcap_stat *ps)**

Данная функция используется для получения статистики сетевого интерфейса во время работы нашей программы, первый её аргумент - это дескриптор открытого сетевого интерфейса, второй это указатель на стуркутуру pcap_stat.

```
struct pcap_stat {
        u_int ps_recv;          /* кол-во принятых пакетов */
        u_int ps_drop;          /* кол-во отброшенных пакетов */
        u_int ps_ifdrop;        /* отброшено определенным интерфейсом (пока не работает) */
};
```

**int pcap_datalink(pcap_t *p)**

Эта функция отображает тип открытого сетевого интерфейса, заданного его дескриптором.
Например возвращает DLT_EN10MB, это означает, что сетевой интерфейс - 10мбит Ethernet,
или DLT_PPP - PPP интерфейс (данная функция удобна для получения информации об используемом канальном уровне на данном интерфейсе).

```
 if (pcap_datalink(fd) != DLT_EN10MB)
 {
    printf("Only ethernet supported\n");
    pcap_close(fd); // закрываем интерфейс
    exit(0);
  }
```

# Создание фильтра для перехватываемых пакетов

- **int pcap_compile(pcap_t *p, struct bpf_program *fp, char *str, int optimize, bpf_u_int32 netmask)**

  Используется для занесения в структуру bpf_program строки, заданной 3им аргументом, в которой
- записано регулярное выражение в стиле bpf (Berkley Packet Filter). Данная возможность позволяет достаточно легко задавать свои правила фильтрации захватываемых пакетов, без того, чтобы писать самому сложные условия обработчика пакетов.
- Пример третьего аргумента на следующем слайде

- Четвертый аргументом в функции выглядит так: 1, если оптимизировать код, 0, если нет. Следует добавить, что при оптимизации программа ест гараздо больше ресурсов.

- Например нам нужно, чтобы в нашу программу попадали только пакеты по протоколу TCP, идущие на порт 21:

  **ip proto TCP and port 21**

  Или например я хочу получать только пакеты с адреса 192.168.0.1 на адрес 192.168.0.2 по протоколу icmp

  **src 192.168.0.1 and dst 192.168.0.2 and ip proto ICMP**

  Хотим видеть весь траффик, исключая TCP-траффик на 22 порт: proto TCP and not port 22

  Показывать пакеты по протоколу ICMP или UDP пакеты на порт 31337: ip proto **ICMP or ip proto UDP and port 31337**

  Пакеты идущие с хоста www.gov.ru:
- **src host www.gov.ru**

  Для получения более подробной информации по регулярным выражениям bpf, читайте tcpdump(1).

- /* встриваем фильтр на IPv4 TCP порт 21 */
- **if ((pcap_compile(fd, &bpf_fil, "ip proto TCP and port 21", 1, mask)) < 0)**
- **{**
- **pcap_perror(fd, "pcap_compile ");**
- **pcap_close(fd);**
- **exit(0);**
- **}**

- **int pcap_setfilter(pcap_t *p, struct bpf_program *fp)**

  Данная функция используется для применения фильтра, встроенного функцией pcap_compile в структуру bpf_program.
  Функция возвращает -1 при ошибке, и 0 при удачном выполнении функции.

- struct  bpf_program bpf_fil;   /* структура для фильтра */
- /* применяем фильтр */
- **if ((pcap_setfilter(fd, &bpf_fil)) < 0)**
- **{**
- **pcap_perror(fd, "pcap_setfilter ");**
- **pcap_close(fd);**
- **exit(0);**
- **}**
-

  **char *pcap_geterr(pcap_t *p)**

  Вернёт строку ошибки и будет использована для получения информации об ошибках, в таких функциях как:

  pcap_setfilter(), pcap_compile(), pcap_loop() и др. Первым аргументом прописывается дескриптор открытого интерфейса.

# Перехват и обработка пакетов

**u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h)**
Функция возвращает указатель на следующий захваченный пакет. Вторым аргументом функции, задается указатель на достаточно полезную структуру pcap_pkthdr, которая будет заполнена данными после выполнения функции.

Так же рассматриваемая структура используется в обработчике пакетов, вызываемом функцией pcap_loop.

```
struct pcap_pkthdr {
    struct timeval ts;        /*  временная метка (время захвата пакета) */
      bpf_u_int32 caplen;    /* длина захваченных данных */
      bpf_u_int32 len;        /* размер текущего захваченного пакета */
};
```

# Перехват и обработка пакетов

**u_char *pcap_next_ex(pcap_t *p, struct pcap_pkthdr *h,**
**const u_char **pkt_data)**

Функция возвращает возвращает одно из следующих числовых значений: 1 – пакет был прочитан, 2 – время ожидания истекло, -1 – произошла ошибка, -2 – пакеты прочитаны из сохраненного файла и больше не доступны. Первый аргумент – дескриптор открытой сессии. Вторым аргументом функции, задается указатель на структуру **pcap_pkthdr**, которая будет заполнена данными после выполнения функции.

Если поместить эти функции в цикл, то можно организовать перехват заданного количества обрабатываемых пакетов. Однако, возможно использовать более удобный вариант, приведенный на следующем слайде.

**int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char** *user)

Данная функция используется для создания цикла для обработки захватываемых пакетов, первым параметром задается дескриптор открытого интерфейса, вторым - количество пакетов, которое необходимо обработать, если параметр отрицательный то число пакетов не ограничено и цикл будет бесконечным, если не возникнет ошибка или пройдет время чтения с интерфейса заданный в функции **pcap_open_live**.

Третий параметр функции это указатель на функцию которая будет вызвана для обработки нового пакета.

**u_char *user** — это буфер обмена между pcap_loop() и функцией обработчика сообщений (данные которые передаются в функцию обработкчика заданным первым аргументом).

# Callback

**int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user)**

Третий аргумент - имя callback-функции (только имя, без скобок)
Прототип нашей callback-функции должен быть таким:

**void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet);**

Первое - функция возвращает пустое значение (void).
Первый аргумент совпадает с последним аргументом pcap_loop().
Второй аргумент - заголовок pcap, который содержит информацию о том, когда был перехвачен пакет, его размер и т.д.
**u_char *pkt_data** — это сам кадр (перехваченный с помощью pcap_loop()), начиная с заголовка локального уровня.

**void handler(char *, struct pcap_pkthdr *, u_char *);** /* обработчик пакетов */

**process = (pcap_handler) handler;** // получаем указатель на функцию обработчик

```
if (pcap_loop(fd, -1, process, NULL))
{
   pcap_perror(fd, "pcap_loop ");
   pcap_close(fd);
   exit(0);
 }
```

# Функции ошибок

**char *pcap_geterr(pcap_t *p)**

Вернёт строку ошибки и будет использована для получения информации об ошибках, в таких функциях как: **pcap_setfilter(), pcap_compile(), pcap_loop()** и др.
Первым аргументом прописывается дескриптор открытого интерфейса.

**void pcap_perror(pcap_t *p, char *prefix)**

Выдаёт произошедшую ошибку на экран, используется для получения ошибок в функциях: **pcap_setfilter(), pcap_compile(), pcap_loop()** и др. Первым аргументом задается дескриптор открытого интерфейса.

# Закрытие сессии

**void pcap_close(pcap_t *p)**

Данная функция закрывает открытый интерфейс.

# Дополнение

Есть **возможность отправки данных в открытый интерфейс** посредством pcap существует лишь в WinPCap и реализована в виде функции

**int pcap_sendpacket(pcap_t *p, u_char *buf, int size)**

**u_char *buf** — это кадр данных, который необходимо отправить.
int size — его размер.

Контрольную сумму локального уровня высчитывать и вставлять не надо, это делает драйвер сетевого уровня автоматически.

# Пример-программа

Эта программа перехватывает пакеты на устройстве, которое возвращает pcap_lookupdev(), переводя его в pormiscuous режим. Она обнаруживает пакет, который идет через 21-й порт (ftp) и выводит его размер в байтах. Вызов pcap_close() закрывает открытую сессию перехвата.

```
#include <pcap.h>
#include <stdio.h>
int main()
{
pcap_t *handle;                          // хэндл сессии
char *dev;                               // устройство, на котором будем
                                         // перехватывать траффик
char errbuf[PCAP_ERRBUF_SIZE];           // строка с описанием ошибки
struct bpf_program filter;               // скомпилированный фильтр

char filter_app[] = “ip proto TCP and port 21";   // фильтр
bpf_u_int32 mask;                        // наша сетевая маска
bpf_u_int32 net;                         // наш ip адрес
struct pcap_pkthdr header;               // заголовок пакета,
                                         // который заполнит pcap
const u_char *packet;                    // сам пакет
```

# Пример-программа (продолжение)

```
                                        // определим интерфейс
dev = pcap_lookupdev(errbuf);

                                // получим сетевой адрес и маску интерфейса
pcap_lookupnet(dev, &net, &mask, errbuf);

                                // откроем сессию перехвата в promiscuous режиме
handle = pcap_open_live(dev, BUFSIZ, 1, 0, errbuf);

                                // скомпилируем и применим пакетный фильтр
pcap_compile(handle, &filter, filter_app, 0, net);
pcap_setfilter(handle, &filter);
                                        // перехватим пакет
packet = pcap_next(handle, &header);

                                        // выведем его длину в консоль
printf("Jacked a packet with length of [%d]\n", header.len);
                                        // закроем сессию
pcap_close(handle);
return(0);
}
```

# Лекция закончена!