Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Сетевой анализатор трафика на языке программирования Perl в ОС Windows

Курсовая работа студентки 345 группы

Филипповой Анастасии Валерьевны

Научный руководитель Баклановский Максим Викторович

Санкт-Петербург

2010

Оглавление

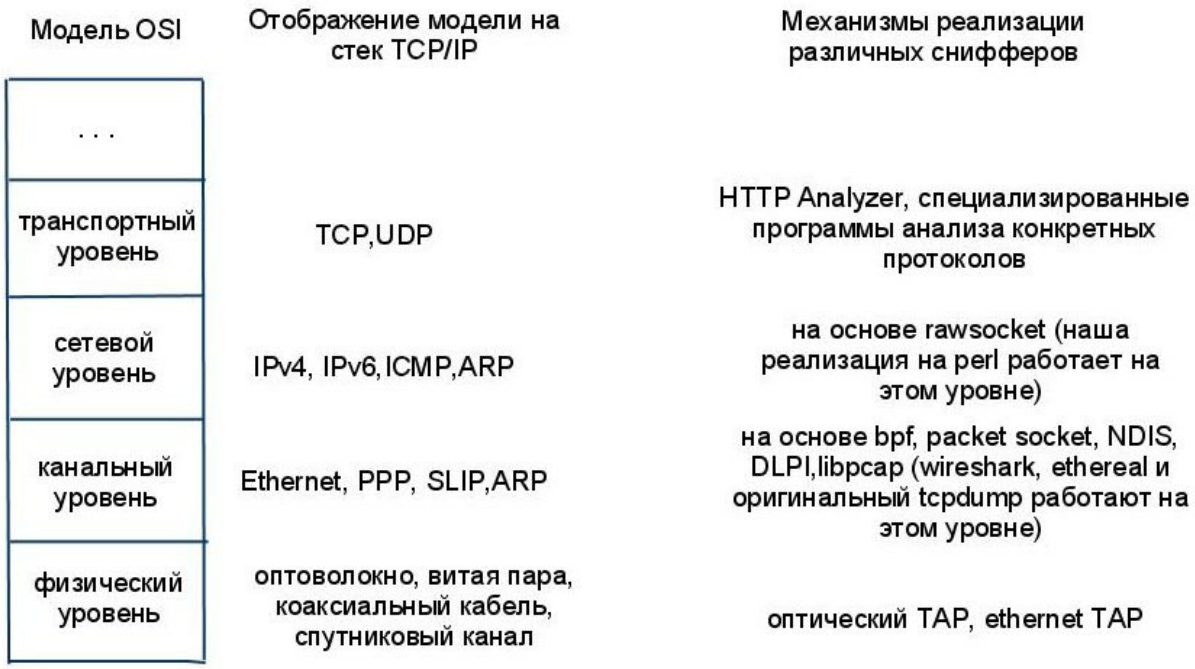
Введение

Широкое распространение сетей, гигантские объемы информации, ежесекундно находящиеся "в полете" в сети, сама сеть как сложная, многоуровневая структура взаимодействия узлов-участников явились объективными причинами появления снифферов - программ для мониторинга пакетного обмена в сети.

Даже такой, казалось бы, простой и привычный процесс, как просмотр веб-странички в браузере, при ближайшем рассмотрении оказывается подобен матрешке:

- Браузер загружает с сервера и отображает на экране информацию, которая кодируется в соответствии с протоколом HTTP. Ограничения на размер страницы нет.
- HTTP – это текстовый протокол, который использует в качестве транспорта протокол TCP из стека TCP/IP. Размер отдельного TCP пакета ограничен, поэтому страница разбивается на множество фрагментов. Каждый из фрагментов получает заголовок, содержащий служебную информацию протокола TCP(номера портов, номер пакета по порядку, флаги, контрольные суммы,…).
- TCP – это протокол гарантированной передачи данных, использующий в качестве транспорта протокол IP. На этом уровне каждый TCP пакет может быть разбит (фрагментирован) на несколько фрагментов меньшей длины. Каждый фрагмент получает заголовок уровня IP, содержащий информацию об IP адресах сервера и машины с браузером, о необходимом уровне качества обслуживания, общей длине пакета и смещении фрагмента данных относительно начала TCP пакета.
- Протокол IP использует в качестве транспорта множество других протоколов в зависимости от аппаратуры передачи данных. В случае использования адаптеров Ethernet каждый IP пакет получит заголовок, содержащий MAC адреса отправителя и получателя и «хвост», содержащий контрольную сумму данных.
- Допустим, что компьютер с браузером и веб-сервер находятся на разных континентах в пределах общей виртуальной частной сети. Тогда, с большой долей вероятности, Ethernet пакеты получат сначала заголовок 802.1Q виртуальной локальной сети, а затем, на аппаратуре провайдера, один или более заголовков с меткой MPLS.
- И на этом матрешка может не закончиться.

Такая множественная инкапсуляция данных позволяет разделить сложный процесс передачи веб-странички на несколько стадий с отдельными «простыми» правилами. На каждой стадии абсолютно неважно, что было завернуто на предыдущей. Все эти уровни хорошо иллюстрируются моделью OSI.



СНИФФЕР(от англ. to sniff — нюхать) – это программа, позволяющая перехватывать и анализировать сетевой трафик. Снифферы часто используют для диагностики проблем, возникающих в сети. Различных снифферов существует очень много, их можно классифицировать по назначению и уровню взаимодействия с ОС. Несколько примеров:

- HTTPScan [1] - онлайновый сниффер, позволяющий просматривать ответ веб-сервера в протоколе HTTP.
- Wireshark [2] - мощная утилита, позволяющая анализировать сетевой трафик, начиная с канального уровня, и декодировать все уровни вложенных данных.
- CommView [3] - сетевой анализатор и декодер, позволяющий анализировать трафик наиболее популярных сигнальных протоколов SIP и H.323, включая RTP трафик. Позволяет проигрывать перехваченные VoIP звонки.
- Dsniff [4] - коллекция утилит, официально заявленная, как средство тестирования сети на устойчивость к взлому. Одна из утилит, в частности, служит для сбора паролей из сетевого трафика.

Очевидно, что снифферы являются программами двойного назначения - могут быть использованы как во благо (в основном для проверки работоспособности сети, поиска неисправностей, анализа обмена пакетами), так и во вред (доступ к пакету на достаточно низком уровне дает возможность несанкционированного доступа к данным в пакете, легко могут быть перехвачены пароли протоколов, не имеющих криптографической защиты).

Одной из первых утилит этого класса была разработанная в Lawrence Berkeley National Laboratory программа tcpdump [5], входящая в любой дистрибутив ОС UNIX начала 90-х годов XX-го века.

Большинство снифферов получают доступ к данным на канальном уровне сетевой

модели. В примерах выше три из четырех программ работают на этом уровне. Альтернативным решением является доступ к сетевым данным на сетевом уровне стека протоколов TCP/IP с помощью raw-socket-ов.

Терминология

Анализатор трафика, или **сниффер** (от англ. to sniff — нюхать) — сетевой анализатор трафика, программа или программно-аппаратное устройство, предназначенное для перехвата и последующего анализа либо только анализа сетевого трафика, предназначенного для других узлов.

TCP/IP - Transmission Control Protocol/Internet Protocol - набор протоколов передачи данных, на которых основано функционирование Интернет.

libpcap - библиотека Pcap (Packet Capture), позволяет получить доступ к сетевым данным, поступающим на сетевой адаптер компьютера.

NDIS - Network Driver Interface Specification – спецификация, разработанная Microsoft и 3Com для сопряжения драйверов сетевых адаптеров с ОС.

tcpdump – программа, позволяющая перехватывать и анализировать сетевой трафик, проходящий через компьютер, на котором запущена.

bpf - berkeley packet filter , это архитектура ядра для захвата пакетов, созданная для работы с ядрами ОС UNIX

DLPI - Data Link Provider Interface - аналогично в HP-UX, Solaris, AIX.

mac-адрес - уникальный идентификатор, сопоставляемый с сетевыми адаптерами в широковещательных сетях (ethernet).

promiscuous режим - режим сетевого адаптера, в котором он принимает все пакеты вне зависимости от наличия собственного, multicast или broadcast, mac-адреса в поле получателя.

socket - программный интерфейс для обеспечения обмена данными между процессами. Процессы могут исполняться на одной ЭВМ или на разных, соединенных сетью.

raw socket - позволяет процессам получить доступ к служебным данным сетевого уровня (заголовки пакетов).

packet socket - позволяет процессам получить доступ к данным канального уровня (ethernet заголовки)

icmp - Internet Control Message Protocol - один из протоколов стека TCP/IP.

семафор – объект, позволяющий войти в заданный участок кода не более чем n-потокам (в частности реализован в IPC SystemV unix).

multicast - многоадресная передача (имеет выделенный диапазон mac-адресов)

broadcast - широковещательная передача (имеет свой mac-адрес)

TAP – устройство, позволяющее считывать данные со среды передачи без участия в передаче.

DDoS – Distributed Denial of Service – распределенная атака, преследующая своей целью потерю работоспособности ПО или аппаратуры атакуемого.

Spoofed адреса – поддельные IP адреса отправителя специально сгенерированных пакетов, использующихся в DDoS.

802.1Q – сетевой стандарт, позволяющий нескольким Ethernet сетям разделять общий физический канал.

MPLS – MultiProtocol Label Switching – мультипротокольная коммутация по меткам. Работает между канальным и сетевым уровнями модели OSI.

SIP – Session Initiation Protocol - стандарт на способ установления и завершения пользовательского сетевого сеанса, включающего обмен мультимедийным содержимым (audio,video,…).

H.323 – стандарт мультимедийных приложений по сетям с коммутацией пакетов с негарантированной пропускной способностью.

RTP – Real-time Transport Protocol- работает на транспортном уровне и используется для передачи трафика реального времени.

VoIP – Voice over IP – система связи , обеспечивающая передачу речевого сигнала по сети. В качестве сигнализации использует SIP или H.323 . Голосовой трафик передается с помощью RTP.

Цель работы

Реализация классической командно-строчной утилиты tcpdump на языке perl в среде ОС Windows без использования специализированного ПО для низкоуровневого доступа к сетевым адаптерам (libpcap, NDIS драйвера,...).

Оригинальная утилита tcpdump работает с помощью библиотеки libpcap, что позволяет ей получить данные на канальном уровне. Работа нашей программы основана на raw-socket-ax, что ограничивает нас сетевым уровнем.

Все функциональные возможности оригинальной утилиты можно разбить на следующие группы:

1. Те, что могут быть реализованы в программе на языке perl в ОС Windows.

2. Те, что могут быть реализованы с ограниченным функционалом.

3. Те, что не могут быть реализованы.

К третьей группе относятся следующие опции оригинального tcpdump-a :

[-e] – вывод информации об адресах канального уровня. Реализация невозможна из-за того, что приложение работает на сетевом уровне.

Ко второй группе в первую очередь относятся выражения-фильтры, которые позволяют выводить информацию, отсеивая ее по адресам, протоколам или их логическим комбинациям. Функционал может быть реализован только частично, поскольку оригинальная программа может включать выражения адреса и протоколы канального уровня, что для нашего приложения недоступно.

[-O] – не выполнять оптимизацию фильтров-выражений, поскольку эта опция очень сильно опирается на оригинальную реализацию.

Мы ограничились реализацией только небольшого набора принципиальных функциональных возможностей прототипа.

tcpdump [-n] [-p] [-h] [-v] [-D] [-t] [-c count] [-i (autolinterface)

- -n use numeric IP-addresses and port numbers
- -v little bit more verbose output
- -h this help
- -p don't put interface into promiscuous mode
- -i iface1:iface2:... IP-address(es) of \"sniffed\" interface or \"auto\" for autodetect
- -D print the list of the network interfaces available on the system
- -t don't print a timestamp on each dump line
- -c exit after reciving count packets

Остальные функции прототипа разбили на группы по сложности возможной реализации в будущем.

Обзор

Большинство снифферов получают доступ к данным на канальном уровне модели TCP/IP. С одной стороны это позволяет получить информацию об адресах канального уровня (mac-адреса), с другой - реализация драйверов канального уровня в различных ОС сильно отличается. Эти отличия заставляют использовать для доступа к сетевым данным специализированное ПО, тесно взаимодействующее с ОС.

В ОС Unix семейства BSD (Free/Net/OpenBSD, BSDI [6][7][8][9]) используется драйвер ядра (bpf).

В ОС HP-UX[10] и Solaris[11] используется интерфейс DLPI.

В ОС Linux[12] доступ к сетевым данным обеспечивается через packet-socket.

В Windows через NDIS-драйвер.

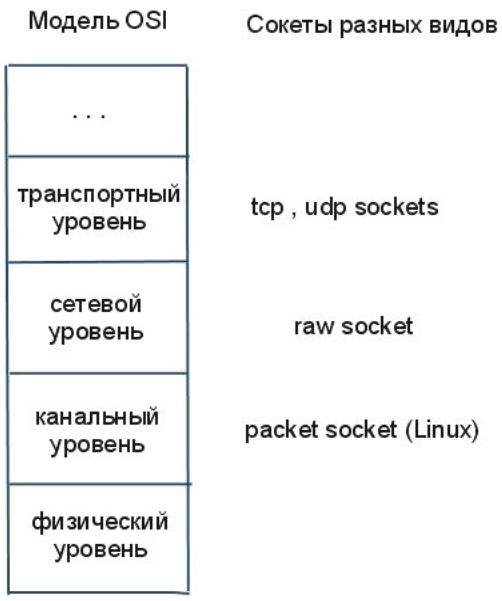
Существует библиотека libpcap, которая, опираясь на эти механизмы, реализует общий для всех типов ОС API для доступа к сетевым данным.[5]

Цели, которые ставятся при создании сниффера, определяют верхний интересующий авторов уровень модели OSI, а возможные механизмы доступа к данным определяют низший уровень модели, доступный снифферу. Сниффер должен уметь декодировать остальные уровни, находящиеся между ними.

Вне зависимости от используемой ОС широковещательный сетевой адаптер (ethernet) может работать в двух режимах: обычном и promiscuous. В обычном режиме он принимает из сети и передает драйверу только те пакеты, которые содержат его собственный mac-адрес в поле получателя (либо пакеты с multicast/broadcast адресами). В promiscuous режиме принимаются все пакеты. Как правило, снифферы переводят "прослушиваемые" адаптеры во второй режим.

Очевидно, что в promiscuous режиме приемный буфер Raw-Socket-а в ОС используется сильнее, и, как следствие, при большом объеме анализируемого трафика невозможно обойтись без потери части информации о сетевом трафике из-за переполнения буфера.

Для доступа к данным на сетевом уровне стека протоколов TCP/IP используются программные интерфейсы socket различных видов.



С точки зрения реализации сниффера наиболее подходящим из них является raw-socket. Socket этого типа позволяет получить доступ к информации в пакете, начиная с уровня IP. Этот подход и был выбран для наших целей.

Описание архитектуры

Функционально описание реализации можно разбить на три основных части:

1. "Сетевая часть" - инициализация и чтение raw-socket.

Raw-socket "привязывается" (bind) к конкретному интерфейсному IP-адресу, либо ко всем IP-адресам, используемым на интерфейсах в ОС [13]

```
socket(S,AF_INET,SOCK_RAW,0) || die "Can't open raw socket...\n";
if ($interf != INADDR_ANY) {
        $ia = inet_aton($interf);
} else {
        $ia = INADDR_ANY;  # то же самое что $ia = inet_aton("0.0.0.0");
}

$pa = sockaddr_in(0,$ia);
bind(S,$pa) || die "Can't bind socket...\n";
```

В полной мере raw-socket-ы реализованы в OS Windows, начиная с Windows2000. До этого Windows95/98 ограниченно поддерживали “сырые” socket-ы только для ICMP[14]. Полное отсутствие ограничений на посылку пакетов с программно сформированными заголовками в WindowsXP до уровня ServicePack1 сделало эту ОС идеальной для организации DDoS атак с поддельными (spoofed) адресами отправителя.

Начиная с SP2, фирма Microsoft ограничила функциональность raw-socket-ов. В частности, в обычном режиме сетевого адаптера, читается только входящий icmp-трафик, адресованный на интерфейсные IP-адреса машины. Для полноценного функционирования необходим перевод сетевого адаптера в promiscuous-режим. [15] Это возможно только при "связывании" raw-socket с КОНКРЕТНЫМ интерфейсным адресом. Если socket привязан к 0.0.0.0 (INADDR_ANY), режим не включится.

Перевод интерфейса в promiscuous режим выполняется системным вызовом ioctl с параметрами, описанными в спецификации интерфейса Windows Winsock2[15] и в файлах заголовков .Net 2.0 SDK:

```
# взято из mstcpip.h
#define SIO_RCVALL _WSAIOW(IOC_VENDOR,1)
# взято из winsock2.h
#define _WSAIOW(x,y) (IOC_IN|(x)|(y))
#define IOC_IN 0x80000000
#define IOC_VENDOR 0x18000000
```

```
use constant SIO_RCVALL => 0x80000000|0x18000000|1;


if(!ioctl(S,SIO_RCVALL,pack("I",1))) {
        print "Turning promiscuous mode failed ...\n";
}
```

Необходимость анализировать трафик нескольких интерфейсов одновременно с учетом ограничений, описанных выше, вынуждает нас открывать несколько raw-socket-ов и читать их одновременно. Следовательно, необходимо организовать группы параллельных процессов и передать в них соответствующие адреса интерфейсов.

```
        foreach my $x (@iface) {
        $ifacethreads[$i] = threads->create(\&chld,$x);
        $ifacethreads[$i]->detach();
        $i++;
    }
    …

    sub chld {
        my $interf = shift;
    …
}
```

Используем «нити» интерпретатора perl (ithreads).

2. "Интерфейсная часть" - передача параметров в программу.

Параметры передаем в командной строке - описание можно получить, вызвав tcpdump.pl с ключом -h.

В программе организовано чтение клавиатуры (ждем символ 'q') и обработка сигналов SIGINT, SIGBREAK,SIGTERM.

3. "Презентационная часть" - вывод информации о "перехваченных" пакетах.

Информация о каждом пакете, считанном из raw-socket, помещается в очередь печати, организованной с помощью Thread ::Queue. Очередь разбирается с помощью отдельной thread – печати. Информация о пакете выводится в подробном или кратком виде в зависимости от наличия параметра -v (verbose) в командной строке.

Для отображения информации о пакетах разных протоколов стека TCP/IP используются разные подпрограммы, учитывающие специфику протоколов.

Реализованы специфические подпрограммы для IP-протоколов icmp,udp,tcp.

Добавив подпрограмму с именем, совпадающим с названием протокола, возвращенным командой *lc(getprotobynumber($proto),* получаем отдельный обработчик для протокола с номером $proto.

В Windows функция getprotobynumber() возвращает имя протокола по его номеру, опираясь на информацию из файла \WINDOWS\system32\drivers\etc\protocols. Этот файл содержит следующие записи :

- ip 0 IP # Internet protocol
- icmp 1 ICMP # Internet control message protocol
- ggp 3 GGP # Gateway-gateway protocol
- tcp 6 TCP # Transmission control protocol
- egp 8 EGP # Exterior gateway protocol
- pup 12 PUP # PARC universal packet protocol
- udp 17 UDP # User datagram protocol
- hmp 20 HMP # Host monitoring protocol
- xns-idp 22 XNS-IDP # Xerox NS IDP
- rdp 27 RDP # "reliable datagram" protocol
- rvd 66 RVD # MIT remote virtual disk

Для протоколов с другими номерами имя протокола возвращено не будет. В случае если есть необходимость написания специфической подпрограммы для протокола, не входящего в этот список, его следует добавить в этот файл. Вторая из ниже приведенных строчек генерируется программой после добавления протокола gre в файл protocols. При этом отображается имя протокола вместо его номера.

23:40:41.71875 IP 193.124.254.19 > 193.124.81.4 **47** ttl=255 length=48

23:41:05.71875 IP 193.124.254.19 > 193.124.81.4 **gre** ttl=255 length=48

Номера и имена протоколов распределяются IANA - Internet Assigned Numbers Authority[17] в координации с IETF – Internet Engineering Task Force[18].

```
    ...
    my $printer = lc(getprotobynumber($proto));

   if(defined &$printer) {

    &$printer($stamp,\$buf);
   } else {
      ...

     sub icmp {
  ...
 }
 sub tcp { ... }
```

Использование Thread::Queue позволяет нам избежать проблемы наложения вывода от разных thread-ов:

```
my $printqueue = Thread::Queue->new();
…
sub prt {
…
$printqueue->enqueue($str);
…
}

sub printsub {
  my $c = shift;
  while(1){
          my $s = $printqueue->dequeue();
          print $s;
  }
}
```

- так как операция enqueue использует внутреннюю локировку.

Заключение

Реализован аналог утилиты tcpdump на языке perl в ОС Windows. Синтаксис командной строки по возможности сохранен. В отличие от других реализаций снифферов на языке perl(Net::RawIP[12]) данная программа не использует дополнительных низкоуровневых библиотек.

Первоначально для анализа трафика с нескольких сетевых интерфейсов компьютера предполагалось использовать распараллеливание процессов с помощью вызова fork и использование семафоров и локировок для конкурентного вывода на экран.

В процессе разработки было выяснено, что в реализации perl в ОС Windows семафоры отсутствуют. К тому же была заявлена необходимость реализовать функцию подсчета общего количества перехваченных пакетов. С использованием fork модели это было затруднительно из-за необходимости реализовывать межпроцессное взаимодействие.

В связи с этим было принято решение отказаться от модели fork и перейти к использованию «нитей»(threads). Процесс вывода на экран был выделен в отдельный thread, что позволило подсчитывать пакеты. Передача информации между «нитями» с помощью очередей сняла необходимость использовать локировки и семафоры.

Приложения:

1. Оценка сложности реализации функций оригинальной программы.

tcpdump [-AdDefIKlLnNOpqRStuUvxX] [-B buffer_size] [-c count]
[-C file_size] [-G rotate_seconds] [-F file]
[-i interface] [-m module] [-M secret]
[-r file] [-s snaplen] [-T type] [-w file]
[-W filecount]
[-E spi@ipaddr algo:secret,...]
[-y datalinktype] [-z postrotate-command] [-Z user]
[expression]

OPTIONS

-A **5**

Print each packet (minus its link level header) in ASCII. Handy for capturing web pages.

-B **impossible**

Set the operating system capture buffer size to buffer_size.

-c **done**

Exit after receiving count packets.

-C **4**

Before writing a raw packet to a savefile, check whether the file is currently larger than file_size and, if so, close the current savefile and open a new one. Savefiles after the first savefile will have the name specified with the -w flag, with a number after it, starting at 1 and continuing upward. The units of file_size are millions of bytes (1,000,000 bytes, not 1,048,576 bytes).

-d **10**

Dump the compiled packet-matching code in a human readable form to standard output and stop.

-dd **10**

Dump packet-matching code as a C program fragment.

-ddd **10**

Dump packet-matching code as decimal numbers (preceded with a count).

-D **done**

Print the list of the network interfaces available on the system and on which tcpdump can capture packets. For each network interface, a number and an interface name, possibly followed by a text description of the interface, is printed. The interface name or the number can be supplied to the -i flag to specify an interface on which to capture.

This can be useful on systems that don't have a command to list them (e.g., Windows systems, or UNIX systems lacking ifconfig -a); the number can be useful on Windows 2000 and later systems, where the interface name is a somewhat complex string.

The -D flag will not be supported if tcpdump was built with an older version of libpcap that lacks the pcap_findalldevs() function.

-e **impossible**

Print the link-level header on each dump line.

-E **10**

Use spi@ipaddr algo:secret for decrypting IPsec ESP packets that are addressed to addr and contain Security Parameter Index value spi. This combination may be repeated with comma or newline seperation.

Note that setting the secret for IPv4 ESP packets is supported at this time.

Algorithms may be des-cbc, 3des-cbc, blowfish-cbc, rc3-cbc, cast128-cbc, or none. The default is des-cbc. The ability to decrypt packets is only present if tcpdump was compiled with cryptography enabled.

secret is the ASCII text for ESP secret key. If preceeded by 0x, then a hex value will be read.

The option assumes RFC2406 ESP, not RFC1827 ESP. The option is only for debugging purposes, and the use of this option with a true `secret' key is discouraged. By presenting IPsec secret key onto command line you make it visible to others, via ps(1) and other occasions.

In addition to the above syntax, the syntax file name may be used to have tcpdump read the provided file in. The file is opened upon receiving the first ESP packet, so any special permissions that tcpdump may have been given should already have been given up.

-f **3**

Print `foreign' IPv4 addresses numerically rather than symbolically (this option is intended to get around serious brain damage in Sun's NIS server --- usually it hangs forever translating non-local internet numbers).

The test for `foreign' IPv4 addresses is done using the IPv4 address and netmask of the interface on which capture is being done. If that address or netmask are not available, available, either because the interface on which capture is being done has no address or netmask or because the capture is being done on the Linux "any" interface, which can capture on more than one interface, this option will not work correctly.

-F **10**

Use file as input for the filter expression. An additional expression given on the command line is ignored.

-G **4**

If specified, rotates the dump file specified with the -w option every rotate_seconds seconds. Savefiles will have the name specified by -w which should include a time format as defined by strftime(3). If no time format is specified, each new file will overwrite the previous.

If used in conjunction with the -C option, filenames will take the form of `file<count>'.

-i **done**

Listen on interface. If unspecified, tcpdump searches the system interface list for the lowest numbered, configured up interface (excluding loopback). Ties are broken by choosing the earliest match.

On Linux systems with 2.2 or later kernels, an interface argument of ``any'' can be used to capture packets from all interfaces. Note that captures on the ``any'' device will not be done in promiscuous mode.

If the -D flag is supported, an interface number as printed by that flag can be used as the interface argument.

-I **impossible**

Put the interface in "monitor mode"; this is supported only on IEEE 802.11 Wi-Fi interfaces, and supported only on some operating systems.

Note that in monitor mode the adapter might disassociate from the network with which it's associated, so that you will not be able to use any wireless networks with that adapter. This could prevent accessing files on a network server, or resolving host names or network addresses, if you are capturing in monitor mode and are not connected to another network with another adapter.

This flag will affect the output of the -L flag. If -I isn't specified, only those link-layer types available when not in monitor mode will be shown; if -I is specified, only those link-layer types available when in monitor mode will be shown.

-K **6**

Don't attempt to verify IP, TCP, or UDP checksums. This is useful for interfaces that perform some or all of those checksum calculation in hardware; otherwise, all outgoing TCP checksums will be flagged as bad.

-l **done**

Make stdout line buffered. Useful if you want to see the data while capturing it. E.g., ``tcpdump -l | tee dat" or ``tcpdump -l > dat & tail -f dat".

-L **impossible**

List the known data link types for the interface, in the specified mode, and exit. The list of known data link types may be dependent on the specified mode; for example, on some platforms, a Wi-Fi interface might support one set of data link types when not in monitor mode (for example, it might support only fake Ethernet headers, or might support 802.11 headers but not support 802.11 headers with radio information) and another set of data link types when in monitor mode (for example, it might support 802.11 headers, or 802.11 headers with radio information, only in monitor mode).

-m **10**

Load SMI MIB module definitions from file module. This option can be used several times to load several MIB modules into tcpdump.

-M **10**

Use secret as a shared secret for validating the digests found in TCP segments with the TCP-MD5 option (RFC 2385), if present.

-n **done**

Don't convert addresses (i.e., host addresses, port numbers, etc.) to names.

-N 3

Don't print domain name qualification of host names. E.g., if you give this flag then tcpdump will print ``nic'' instead of ``nic.ddn.mil''.

-O impossible

Do not run the packet-matching code optimizer. This is useful only if you suspect a bug in the optimizer.

-p done

Don't put the interface into promiscuous mode. Note that the interface might be in promiscuous mode for some other reason; hence, `-p' cannot be used as an abbreviation for `ether host {local-hw-addr} or ether broadcast'.

-q 2

Quick (quiet?) output. Print less protocol information so output lines are shorter.

-R 10

Assume ESP/AH packets to be based on old specification (RFC1825 to RFC1829). If specified, tcpdump will not print replay prevention field. Since there is no protocol version field in ESP/AH specification, tcpdump cannot deduce the version of ESP/AH protocol.

-r 4

Read packets from file (which was created with the -w option). Standard input is used if file is ``-''.

-S 5

Print absolute, rather than relative, TCP sequence numbers.

-s 3

Snarf snaplen bytes of data from each packet rather than the default of 65535 bytes. Packets truncated because of a limited snapshot are indicated in the output with ``[lproto]'', where proto is the name of the protocol level at which the truncation has occurred. Note that taking larger snapshots both increases the amount of time it takes to process packets and, effectively, decreases the amount of packet buffering. This may cause packets to be lost. You should limit snaplen to the smallest number that will capture the protocol information you're interested in. Setting snaplen to 0 sets it to the default of 65535, for backwards compatibility with recent older versions of tcpdump.

-T 10

Force packets selected by "expression" to be interpreted the specified type. Currently known types are aodv (Ad-hoc On-demand Distance Vector protocol), cnfp (Cisco NetFlow protocol), rpc (Remote Procedure Call), rtp (Real-Time Applications protocol), rtcp (Real-Time Applications control protocol), snmp (Simple Network Management Protocol), tftp (Trivial File Transfer Protocol), vat (Visual Audio Tool), and wb (distributed White Board).

-t done

Don't print a timestamp on each dump line.

-tt 3

Print an unformatted timestamp on each dump line.

-ttt 3

Print a delta (micro-second resolution) between current and previous line on each dump line.

-tttt 3

Print a timestamp in default format proceeded by date on each dump line.

-ttttt 3

Print a delta (micro-second resolution) between current and first line on each dump line.

-u 10

Print undecoded NFS handles.

-U 10

Make output saved via the -w option ``packet-buffered''; i.e., as each packet is saved, it will be written to the output file, rather than being written only when the output buffer fills.

The -U flag will not be supported if tcpdump was built with an older version of libpcap that lacks the pcap_dump_flush() function.

-v done

When parsing and printing, produce (slightly more) verbose output. For example, the time to live, identification, total length and options in an IP packet are printed. Also enables additional packet integrity checks such as verifying the IP and ICMP header checksum.

When writing to a file with the -w option, report, every 10 seconds, the number of packets captured.

-vv 3

Even more verbose output. For example, additional fields are printed from NFS reply packets, and SMB packets are fully decoded.

-vvv 3

Even more verbose output. For example, telnet SB ... SE options are printed in full. With -X Telnet options are printed in hex as well.

-w 4

Write the raw packets to file rather than parsing and printing them out. They can later be printed with the -r option. Standard output is used if file is ``-''. See pcap-savefile(5) for a description of the file format.

-W 4

Used in conjunction with the -C option, this will limit the number of files created to the specified number, and begin overwriting files from the beginning, thus creating a 'rotating'

buffer. In addition, it will name the files with enough leading 0s to support the maximum number of files, allowing them to sort correctly.

Used in conjunction with the -G option, this will limit the number of rotated dump files that get created, exiting with status 0 when reaching the limit. If used with -C as well, the behavior will result in cyclical files per timeslice.

-x **5**

When parsing and printing, in addition to printing the headers of each packet, print the data of each packet (minus its link level header) in hex. The smaller of the entire packet or snaplen bytes will be printed. Note that this is the entire link-layer packet, so for link layers that pad (e.g. Ethernet), the padding bytes will also be printed when the higher layer packet is shorter than the required padding.

-xx **5**

When parsing and printing, in addition to printing the headers of each packet, print the data of each packet, including its link level header, in hex.

-X **5**

When parsing and printing, in addition to printing the headers of each packet, print the data of each packet (minus its link level header) in hex and ASCII. This is very handy for analysing new protocols.

-XX **5**

When parsing and printing, in addition to printing the headers of each packet, print the data of each packet, including its link level header, in hex and ASCII.

-y **impossible**

Set the data link type to use while capturing packets to datalinktype.

-z **6**

Used in conjunction with the -C or -G options, this will make tcpdump run " command file " where file is the savefile being closed after each rotation. For example, specifying -z gzip or -z bzip2 will compress each savefile using gzip or bzip2.

Note that tcpdump will run the command in parallel to the capture, using the lowest priority so that this doesn't disturb the capture process.

And in case you would like to use a command that itself takes flags or different arguments, you can always write a shell script that will take the savefile name as the only argument, make the flags & arguments arrangements and execute the command that you want.

-Z **impossible**

Drops privileges (if root) and changes user ID to user and the group ID to the primary group of user.

This behavior can also be enabled by default at compile time.

Список использованных источников:

[0] Andrew S. Tanenbaum (Prentice Hall)
«Computer Networks» (ISBN-10: 0130661023)
«Operating Systems Design and Implementation»(ISBN-10: 0131429388)

[00] http://www.perl.org/camel.html
[1] Сайт сервиса httpscan
http://www.hackzone.ru/tools/httpscan/
[2] Официальный сайт утилиты WireShark
http://www.wireshark.org
[3] Официальный сайт утилиты CommView
http://www.tamos.com
[4] Официальный сайт коллекции утилит Dsniff
http://monkey.org/~dugsong/dsniff/
[5] Официальный сайт сниффера tcpdump
http://www.tcpdump.org
[6] Официальный сайт ОС FreeBSD
http://www.freebsd.org
[7] Официальный сайт ОС NetBSD
http://www.netbsd.org
[8] Официальный сайт ОС OpenBSD
http://www.openbsd.org
[9] Википедия – Свободная энциклопедия, статья про ОС BSD/OS
http://ru.wikipedia.org/wiki/BSD/OS
[10] Официальный сайт Hewlett-Packard, ОС HP-UX
http://www.hp.com/products1/unix/
[11] Официальный сайт Sun, ОС Solaris
http://www.sun.com/software/solaris/10/index.jsp
[12] Официальный сайт Linux Kernel
http://www.kernel.org/
[13] Perl Programming Documentation, описание модуля Socket
http://perldoc.perl.org/Socket.html
[14] История raw-socket в Windows.
http://www.insidepro.com/kk/230/230r.shtml
[15] Microsoft Developer Network
http://msdn.microsoft.com/en-us/library/ee309610(v=VS.85).aspx
[16] Net:RawIP CPAN-расширение
http://search.cpan.org/~skolychev/Net-RawIP-0.1/RawIP.pm
[17] IANA - Internet Assigned Numbers Authority
` http://www.iana.org/protocols/
[18] IETF – Internet Engineering Task Force
http://www.ietf.org/