快速通道

文档分类
- 红盟公告
- 新手上路
- 漏洞分析
- 工具介绍
- 安全配置
- **编程技术 <=**
- 安全技术
- 业界动态

本类热门
- 初探数据包分析程序设计
- 杀毒软件的简单实现
- readasm的使用范例
- 半开扫描程序
- 注册表操作（RingZ_RgeEdit）
- proftpd 1.2.7/1.2.9rc2 remote root exploit

初探数据包分析程序设计

提交文章： maigan
提交日期： 2003-10-22
文章属性： 原创
文档类别： 编程技术
浏览次数： 今12次/总8848次

初探数据包分析程序设计
Author ：maigan
From : 第八军团－信息安全小组（www.cnhacking.com www.juntuan.org)
Mail : maigan@maigan.com
Warning: 转载本文请注明作者及出处

整天在网上转，也看到许多不错的文章，但我发现大多文章要么只停留在理论上，要么就是太高深。对问题详细分析介绍的很少。今天，我就想以数据包分析程序为主题和大家讨论一下网络编程的的相关问题，我也是新手，有不到之处，还望大家不吝指正。
通过对数据包的分析，我们可以判断通信双方的操作系统、网络信息流量、经过的路由、数据包的大小，以及数据包的内容等等。对于喜欢网络安全的人来说，掌握这方面的知识是相当重要的。现在的网络通信中，大部分数据都没有加密，我们可以轻易地从数据包中提取账号、密码之类我们关心的数据.大家在看本文时如有困难，可先读一读计算机网络及C程序设计还有协议分析方面的书。下面我将分TCP/IP族协议结构、程序部分函数及数据结构说明、案例程序剖析三个部分与大家共同学习数据包分析程序的设计方法。

一、TCP/IP族协议结构
在说TCP/IP之前，先让我们来认识一下以太网，因为我们现在接触最多的就是以太网，并且研究数据包又是离不开以太网的帧的。在以太网中，数据是以被称为帧的数据结构本为单位进行交换的。以太网中常用的协议是CSMA/CD(carrier sense multiple access with collision detection)即载波监听多点接入/碰撞检测，在这里，我们关注的是帧的格式。常用的以太网帧的格式有两种标准，一种是DIX Ethernet V2标准，另一种是IEEE的802.3标准。现在最常用的MAC帧是V2格式，这也是我们所要研究的格式，至于802.3帧我们不再讨论。以太网V2帧的格式如下：
(插入8字节）目的地址（6字节）->源地址(6字节)->类型(2字节)->数据(46-1500)->FCS(4字节)
以太网的地址由48位的二进制来表示,也就是我们常说的MAC地址及硬件地址。在MAC帧前还有8字节的前同步码和帧的开始定界符，之后才是地址等报头信息。接收端和发送端的地址之后是2字节的类型字段，存放帧中传送数据的上层协议类型，RFC1700号文档规定了这些，如下：
ETHER TYPES(十六进制) PROTOCOIS
800 IP
806 ARP
8035 Revese ARP
809B Apple Talk
8137/8138 Novel
814c SNMP
帧的数据部分长度为46－1500字节，当小于46时，会在后面加入一个整数字节的填

充字段。FCS(Frame Check Sequence)在以太网常用循环冗余校检(CRC:cyclic redandancy check)。
IP协议为网络层协议，网络层的数据结构体被称为IP数据报。IP地址及域名这两个概念我们就不说了，下面我们来看一看IP数据报的结构：
成员名 字节数 说明
version 1/2 IP的版本，现在为IPV4
IHL(报送长度) 1/2 最常用为20，取5－15之前的值，最大60字节
Type Of Service 1 优先和可靠性服务要求的数值
Total Lenth 2 IP数据报的全长
Identification 2 识别IP数据报的编号
Flags 3/8 1位为0表示有碎块，2位为0表示是最后的碎块，为1表示接收中。
Fragment Offset 13/8 分片在原分组中的位置
TTL 1 数据报寿命，建议值为32秒
Protocol 1 上层协议
Headerchecksum 2 报头检验码
Source Address 4 发送端IP地址
Destination Address 4 接收端IP地址
Options And Padding 4 选项及填充位
其中协议字段的值对我们分析数据包是很重要的，下面列出来给大家看看：
值 协议 意义
1 ICMP Internet Control Message Protocol
6 TCP Tranfer Control Protocol
8 EGP Exterior Gateway Protocol
9 IGP Interior Gateway Protocol
17 UDP User Datagram Protocol
下面这些协议的值在后面的程序中我们可以见到，请大家留心记一下。接着我们介绍地址解析协议(ARP/RARP):
成员名 字节数 说明
Hardware address 2 硬件类型，以太网为1
Protocol address 2 上层协议类型，IP为800
Byte length of each hardware 1 查询物理地址的字节长度，以太网为6
Byte length of each protocol address 1 查询上层协议的字节长度，IPv4时为4
Opcode 2 1为ARP请求，2为响应；3为RARP请求，4为响应
Hardware address of sender of this packet 6 发送端硬件地址
protocol address of sender of this packet 4 发送端IP地址
Hardware address of target of this packet 6 查询对象硬件地址
Protocol address of target of this packet 4 查询对象IP地址
ARP/RARP协议用来查询IP对应的硬件地址或反过来查询IP地址，这在我们分析数据包时也会见到。下面介绍ICMP协议。我们常用的PING命令就是用的这个协议，这个协议比较简单，由类型(1字节)、代码(1字节)、检验和(2字节)、还有四个字节的与类型相关的可变部分及数据构成。
数据包在运输层还有两个重要的协议，即TCP/UDP，TCP/UDP中使用端口的概念，以区别计算机上不同的程序。下面我们先来看看TCP数据报的首部构成：
成员名 字节数 说明
Source Port 2 发送端端口号
Destination Port 2 接收端端口号
Sequence NO 4 本报文段所发送的第一个字节的序号
ACk Number 4 期望收到的下一个报文段的序号
DAta Offset 1/2 首部的长度
Reserved 3/4 保留今后用
Contol Bits 3/4 控制位
Window 2 滑动窗口的大小
Checksum 2 检验和
Urgent Pointer 2 紧急指针
Options And Padding 4 可选，真充项
Tcp被使用在跨越路由器进行网络服务的网络应用程序中，如WWW、电子邮件、新闻、FTP等。UDP则是在IP的基础上加入了端口的概念，其结构很简单，只有八个字节首部如下：
源端口(2字节)->目的端口(2字节)->长度(2字节)->检验和(2字节)

二、程序部分函数及数据结构说明
在此部分我们将介绍后面程序中用到的部分函数及数据结构。在程序中我们使用了PCAP程序库，大家可以从
ftp://ftp.ee.lbl.gov/libpcap.tar.z下载。我们主要在Redhat Linux下测试程序，这里简单介绍一下程序库的安装方法，其它环境请大家自行解决。我的目的是给大家编写数据包分析程序提供思路，至于实用程序的实现这里不做介绍，第三部分给出的程序也不具实用性，为了演示，程序中实现的功能较多而有些地方又不够详细，编写实用程序时请适当取舍并加入你所需要的功能实现部分。PCAP程序库的安装方法如下：

1、解压文件
2、进入文件目录执行./configure 及make
3、使用Make命令，设定手册和Include文件(要有Root权限)，执行以下命令：
make install -man
make install -incl
4、如出现不存在Include及Include/net目录，则建立此目录并重新执行 make install -incl
5、检查/usr/include/netinet/目录是否存在Protocols.h文件，不存在则拷贝过去。至此程序库安装完毕。
下面介绍程序中出现的部分函数及数据结构：
1、PCAP_t *pd;
此型数据结构称为数据包捕捉描述符。
2、Pcap_Open_Live(argv[1],DEFAUT_SNALEN,1,1000,ebuf)
此函数对Pcap程序库进行初始化并返回指向Pcap_t型数据的指针，其参数列表如下：
char * 指定网络接口
int 取得数据的最大字节数
int 指定网络接口卡，一般用1
int 读出暂停时间
char * 错误消息用缓冲区
3、Pcap_loop(pd,-1,packet_proce,NUll)
此函数程序的核心，反复执行，利用Pcap取得数据包，返回的是读入数据包的个数，错误时返回－1，其参数列表如下：
Pcap_t * 指定取得数据包的数据包捕捉描述符
int 取得数据包的个数，－1为无限
返回指向函数的指针 指定数据包处理的函数
U_char * 指向赋给数据包处理函数字符串的指针
4、struct ether_header * eth
此结构体存储以太网报头信息，其成员如下：
ether_dhost[6] 接收端的MAC地址
ether_shost[6] 发送端的MAC地址
ether_type 上层协议的种类
5、fflush(stdout)
此函数完成的是强制输出，参数Stdout，强制进行标准输出。
6、noths(((struct ether_header *P)->ether_type))
此函数将短整型网络字节顺序转换成主机字节顺序。此类函数还有：
ntohl 长整型 功能同上
htons 短整型 将主机字节顺序转换成网络字节顺序
htons 长整型 同上
7、struct IP *iph
ip型结构体在IPh文件中定义，其成员和第一部分讲到的IP数据报结构对应，如下：
成员名 类型 说明
ip_hl 4位无符号整数 报头长度
ip_v 同上 版本，现为4
ip_tos 8位无符号整数 Type of service
ip_len 16位无符号整数 数据报长度
ip_id 同上 标识
ip_off 同上 数据块偏移和标志
ip_ttl 8位无符号整数 TTL值
ip_p 同上 上层协议
ip_sum 16位无符号整数 检验和
ip_src in_addr结构体 发送端IP
ip_dst 同上 接收端IP
8、struct ether_arp *arph
ether_arp型结构体成员如下：
成员名 类型 说明
ea_hdr arphdr型结构体 报头中地址以外的部分
arp_sha 8位无符号整数数组 发送端MAC地址
arp_spa 同上 发送端IP地址
arp_tha 同上 目标MAC地址
arp_tpa 同上 目标IP地址
9、struct icmphdr * icmp
icmphdr型结构体中包含共用体根据数据报类型的不同而表现不同性质，这里不再列出，只列能通用的三个成员
成员名 说明
type 类型字段
code 代码
checksum 检验和

三、案例程序剖析

```
//example.c
//使用方法：example〈网络接口名〉 > 〈输出文件名〉
//例如：example etho > temp.txe
//结束方法：ctrl+c
//程序开始，读入头文件
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netinet/in_systm.h>
#include<netinet/ip.h>
#include<netinet/if_ether.h>
#include<pcap.h> //pcap程序库
#include<netdb.h> //DNS检索使用
#define MAXSTRINGSIZE 256 //字符串长度
#define MAXSIZE 1024 //主机高速缓存中的最大记录条数
#fefine DEFAULT_SNAPLEN 68 /数据包数据的长度
typedef struct
{
   unsigned long int ipaddr; //IP地址
   char hostname[MAXSTRINGSIZE]; //主机名
}dnstable; //高速缓存数据结构
typedef struct
{
   dnstable table[MAXSIZE];
   int front;
   int rear;
}sequeue;
sequeue *sq; //定义缓存队列
sq->rear=sq->front=0; //初始化队列
//输出MAC地址函数
void print_hwadd(u_char * hwadd)
{
   for(int i=0,i<5;++i)
      printf("%2x:",hwadd[i]);
   printf("%2x",hwadd[i]);
}
//输出IP地址的函数
void print_ipadd(u_char *ipadd)
{
   for(int i=0;i<3;++i)
      printf("%d.",ipadd[i]);
   printf("%d",ipadd[i]);
}
//查询端口函数
void getportname(int portno,char portna[],char* proto)
{
   if(getservbyport(htons(portno),proto)!=NULL)
   {
      strcpy(portna,getservbyport(htons(portno),proto)->s_name);
   }
   else
      sprintf(portna,"%d",portno);
}
//将IP转化为DNS名
void iptohost(unsigned long int ipad,char* hostn)
{
   struct hostent * shostname;
   int m,n,i;
   m=sq->rear;
   n=sq->front;
   for(i=n%MAXSIZE;i=m%MAXSIZE;i=(++n)%MAXSIZE)
   {
      //检查IP是否第一次出现
      if(sq->table[i].ipaddr==ipad)
      {
         strcpy(hostn,sq->table[i].hostname);
         break;
      }
```

```
    }
    if(i=m%MAXSIZE)
    {//不存在则从域名服务器查询并把结果放入高速缓存
        if((sq->rear+1)%MAXSIZE=sq->front) //判队满
            sq->front=(sq->front+1)%MAXSIZE; //出队列
        sq->table[i].ipaddr=ipad;
        shostname=gethostbyaddr((char*)&ipad,sizeof(ipad),AF_INET);
        if(shostname!=NULL)
            strcpy(sq->table[i].hostname,shostname->h_name);
        else
            strcpy(sq->table[i].hostname,"");
        sq->rear=(sq->rear+1)%MAXSIZE;
    }
}
void print_hostname(u_char* ipadd)
{
    unsigned long int ipad;
    char hostn[MAXSTRINTSIZE];
    ipad=*((unsigned long int *)ipadd);
    iptohost(ipad,hostn)
        if(strlen(hostn)>0)
            printf("%s",hostn);
        else
            print_ipadd(ipadd);
}
//处理数据包的函数
void packet_proce(u_char* packets,const struct pcap_pkthdr * header,const
u_char *pp)
{
    struct ether_header * eth; //以太网帧报头指针
    struct ether_arp * arth; //ARP报头
    struct ip * iph; //IP报头
    struct tcphdr * tcph;
    struct udphdr * udph;
    u_short srcport,dstport; //端口号
    char protocol[MAXSTRINGSIZE]; //协议类型名
    char srcp[MAXSTRINGSIZE],dstp[MAXSTRINGSIZE]; //端口名
    unsigned int ptype; //协议类型变量
    u_char * data; //数据包数据指针
    u_char tcpudpdata[MAXSTRINGSIZE]; //数据包数据
    int i;
    eth=(struct ether_header *)pp;
    ptype=ntohs(((struct ether_header *)pp)->ether_type);
    if((ptype==ETHERTYPE_ARP)||(ptype==ETHERTYPE_RARP))
    {
        arph=(struct ether_arp *)(pp+sizeof(struct ether_header));
        if(ptype==ETHERTYPE_ARP)
            printf("arp ");
        else
            printf("rarp "); //输出协议类型
        print_hwadd((u_char *)&(arph->arp_sha));
        printf("(");
        print_hostname((u_char *)&(arph->arp_spa));
        printf(")->");
        print_hwadd((u_char *)&(arph->arp_tha));
        printf("(");
        print_hostname((u_char *)&(arph->arp_tpa));
        printf(")\tpacketlen:%d",header->len);
    }
    else if(ptype==ETHERTYPE_IP) //IP数据报
    {
        iph=(struct ip *)(pp+sizeof(struct ether_header));
        if(iph->ip_p==1) //ICMP报文
        {
            strcpy(protocol,"icmp");
            srcport=dstport=0;
        }
        else if(iph->ip_p==6) //TCP报文
        {
            strcpy(protocol,"tcp");
            tcph=(struct tcphdr *)(pp+sizeof(struct ether_header)+4*iph->ip_hl);
```

```
            srcport=ntohs(tcph->source);
            dstport=ntohs(tcph->dest);
            data=(u_char *)(pp+sizeof(struct ether_header)+4*iph->ip_hl+4*tcph-
>doff);
            for(i=0;i<MAXSTRINGSIZE-1;++i)
            {
               if(i>=header->len-sizeof(struct ether_header)-4*iph->ip_hl-4*tcph->doff);
               break;
               else
                  tcpudpdata[i]=data[i];
            }
         } //TCP数据处理完毕
         else if(iph->ip_p=17) //UDP报文
         {
            strcpy(protocol,"udp");
            udph=(struct udphdr *)(pp+sizeof(struct ether_header)+4*iph->ip_hl);
            srcport=ntohs(udph->source);
            dstport=ntohs(udph->dest);
            data=(u_char *)(pp+sizeof(struct ether_header)+4*iph->ip_hl+8);
            for(i=0;i<MAXSTRINGSIZE-1;++i)
            {
               if(i>=header->len-sizeof(struct ether_header)-4*iph->ip_hl-8);
               break;
               else
                  tcpudpdata[i]=data[i];
            }
         }
         tcpudpdata[i]='\0';
         getportname(srcport,srcp,protocol);
         getportname(dstport,dstp,protocol);
         printf("ip ");
         print_hwadd(eth->ether_shost);
         printf("(");
         print_hostname((u_char *)&(iph->ip_src));
         printf(")[%s:%s]->",protocol,srcp);
         print_hwadd(eth->ether_dhost);
         printf("(");
         print_hostname((u_char *)&(iph->ip_dst));
         printf(")[%s:%s]",protocol,dstp);
         printf("\tttl:%d packetlen:%d,iph->ttl,header->len);
       printf("\n");
         printf("%s",tcpudpdata);
         printf("==endpacket==");
      }
      printf("\n");
}
//Main函数取数据包并初始化程序环境
int main(int argc,char ** argv)
{
   char ebuf[pcap_ERRBUF_SIZE];
   pcap * pd;
   if(argc<=1) //参数检查
   {
      printf("usage:%s<network interface>\n",argv[0]);
      exit(0);
   }
   //设置PCAP程序库
   if((pd=pcap_open_live(argv[1],DEFAULT_SNAPLEN,1,1000,ebuf))=NULL)
   {
      (void)fprintf(stderr,"%s",ebuf);
      exit(1);
   }
   //循环取数据包
   //改变参数－1为其它值，可确定取数据包的个数，这里为无限个
   if(pcap_loop(pd,-1,packet_proce,NULL)<0)
   {
      (void)fprintf(stderr,"pcap_loop:%s\n",pcap_geterr(pd));
      exit(1);
   }
   pcap_colse(pd);
   exit(0);
}
```

//程序结束
到此为止，我的这篇文章就写完了，希望能给大家以启发和帮助。平时和网友交流中发现大多数朋友在网上见到长的文章往往都没有信心看下去，其实以前我也是这样，但在这里我想告诉大家，不踏踏实实的深入进去学习，是得不到收获的。一分耕耘，一分收获。

返回上页

Copyright © 2000-2004 **HUC** All Rights Reserved.
中国红客网络技术联盟 www.cnhonker.com
webmaster@cnhonker.net