# 网络攻防技术与实践课程

## 课程4. 网络嗅探与协议分析

**诸葛建伟**

**zhugejw@gmail.com**

# 内容

1. 网络嗅探技术
2. 课堂实践：使用Tcpdump
3. 网络协议分析技术
4. 课堂实践：使用Wireshark
5. 作业4—解码网络扫描/网络扫描攻防对抗作业

# 网络嗅探

- 网络嗅探(Sniff)
  - 网络监听、网络窃听
  - 类似于传统的电话线窃听
- 网络嗅探技术定义
  - 利用计算机网络接口截获目的地为其他计算机的数据报文
  - 监听网络流中所包含的用户账户密码或私密信息等
- 网络嗅探器(Sniffer)
  - 实现嗅探的软件或硬件设备
  - 嗅探获得数据→二进制格式数据报文
  - 解析和理解二进制数据，获取各层协议字段和应用层传输数据→ 网络协议分析

# 网络嗅探的危害与作用

- 攻击者：内网渗透技术
  - 窃取机密信息
  - 为发起进一步攻击收集信息
- 防御者
  - 管理员可以用来监听网络的流量情况，定位网络故障
  - 为网络入侵检测系统提供底层数据来源基础
- 其他作用
  - 开发网络应用的程序员可以监视程序的网络情况

# 网络嗅探技术与工具分类

- 链路层网络进行分类
  - 以太网嗅探
  - **WiFi嗅探**
  - **…**
  - 目前一些著名嗅探器支持多种链路层网络嗅探，**wireshark, Sniffer Pro…**
- 工具形态
  - 软件嗅探器
  - 硬件嗅探器(协议分析仪): 专用设备, 速度快, 额外功能(如流量记录与重放等), 价格昂贵

# 以太网络的工作原理

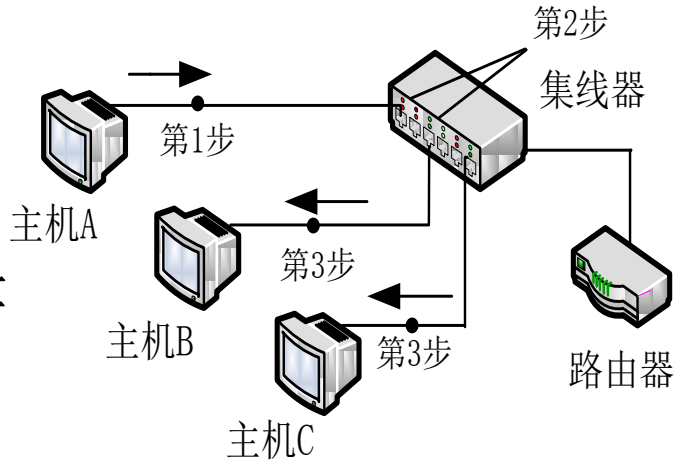
- 载波侦听/冲突检测(**CSMA/CD: 802.3, carrier sense multiple access with collision detection**)技术
  - 载波侦听：是指在网络中的每个站点都具有同等的权利，在传输自己的数据时，首先监听信道是否空闲
    - 如果空闲，就传输自己的数据
    - 如果信道被占用，就等待信道空闲
  - 而冲突检测则是为了防止发生两个站点同时监测到网络没有被使用时而产生冲突
- 以太网采用了**CSMA/CD**技术，由于使用了广播机制，所以，所有在同一媒介信道上连接的工作站都可以看到网络上传递的数据

# 以太网卡的工作模式

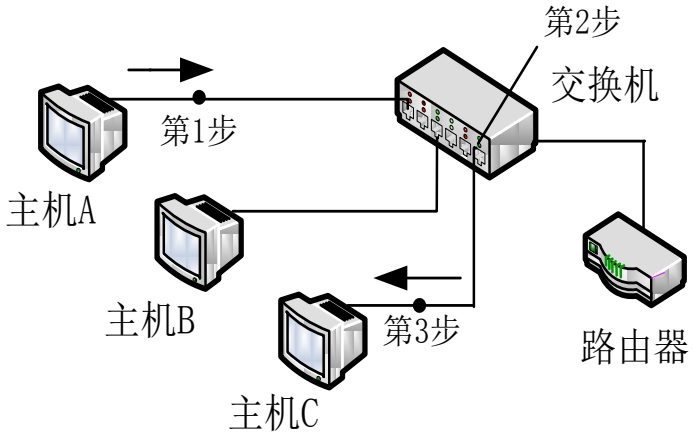
- 网卡的MAC地址(48位)
  - 通过ARP来解析MAC与IP地址的转换
  - 用ipconfig/ifconfig可以查看MAC地址
- 正常情况下，网卡应该只接收这样的包
  - MAC地址与自己相匹配的数据帧
  - 广播包
- 网卡完成收发数据包的工作，两种接收模式
  - 混杂模式：不管数据帧中的目的地址是否与自己的地址匹配，都接收下来
  - 非混杂模式：只接收目的地址相匹配的数据帧，以及广播数据包(和组播数据包)
- 为了监听网络上的流量，必须设置为混杂模式

# 共享式网络和交换式网络

- 共享式网络
  - 通过**Hub(集线器)**连接
  - 总线方式**:** 通过网络的所有数据包发往每一个主机
  - 能够嗅探整个**Hub**上全部网络流量




- 交换式网络
  - 通过**Switch(交换机)**连接
  - 由交换机构造一个“**MAC**地址**-**端口”映射表
  - 发送包的时候，只发到特定端口上
  - 只能监听同一端口上流量
  - 可通过流量映像口监听**(SPAN)**



# 交换式网络中的嗅探攻击

- **MAC地址洪泛攻击**
  - 向交换机发送大量虚构**MAC**地址和**IP**地址数据包
  - 致使交换机“**MAC**地址**-**端口映射表”溢出
  - 交换机切换入所谓的“打开失效”模式**-**“共享式”
- **MAC欺骗**
  - 假冒所要监听的主机网卡，将源**MAC**地址伪造成目标主机的**MAC**地址
  - 交换机不断地更新它的“**MAC**地址**-**端口映射表”
  - 交换机就会将本应发送给目标主机的数据包发送给攻击者
- **ARP欺骗**
  - 利用**IP**地址与**MAC**地址之间进行转换时的协议漏洞
  - 第五章详细介绍

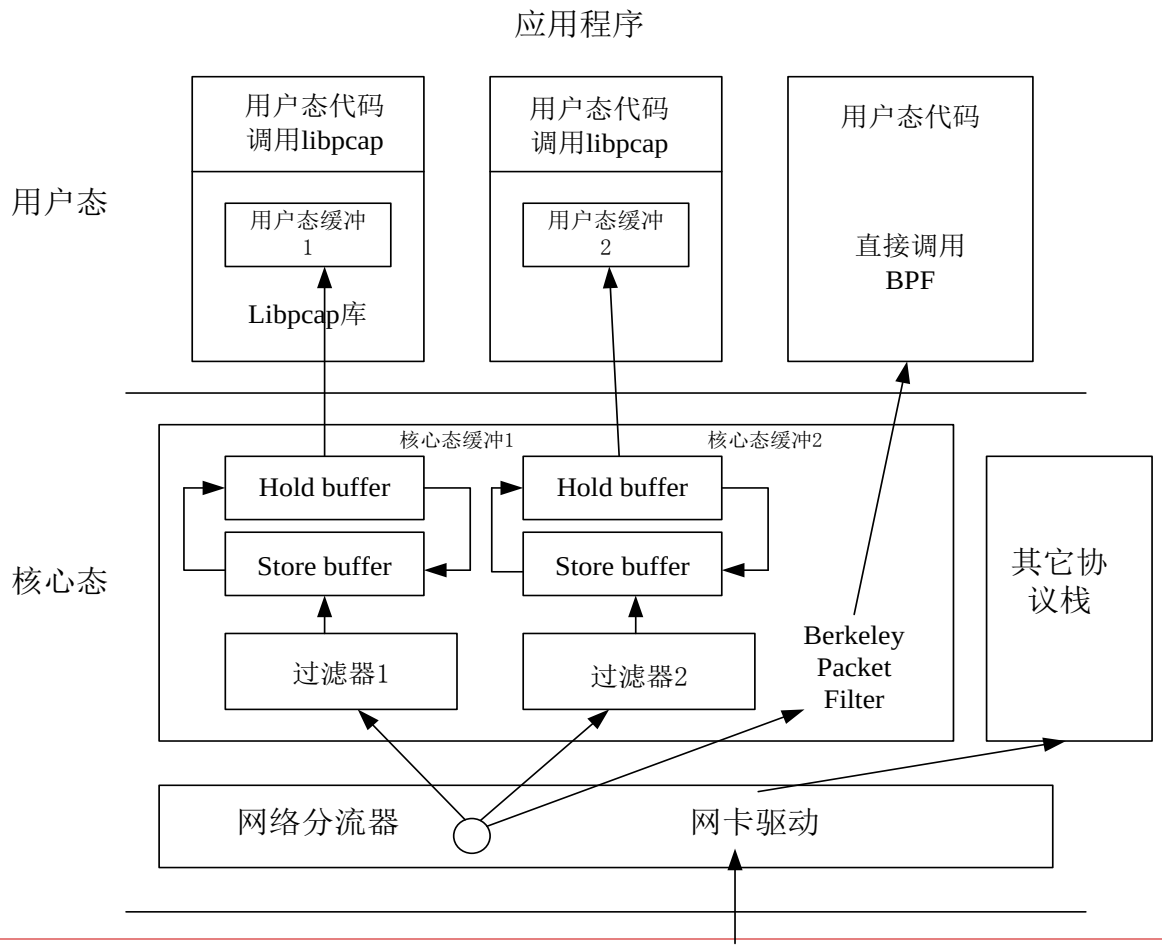
# 应用程序抓包的技术

- 类Unix平台提供了标准的API支持
  - 内核态: BPF(Berkeley Packet Filter)
  - 用户态函数库：libpcap
  - 用户态嗅探程序：tcpdump等
- Windows平台通过驱动程序来抓取数据包
  - 驱动程序: NPF(NetGroup Packet Filter)
  - 用户态函数库：winpcap
  - 用户态嗅探程序：windump等

# BPF(Berkeley Packet Filter)

- BSD数据包捕获
  - BPF是一个核心态的组件，支持数据包“过滤”抓取
  - Network Tap接收所有的数据包
  - BPF虚拟机机器语言的解释器，比较/算术等操作
  - Kernel Buffer，保存过滤器送过来的数据包
  - User buffer，用户态上的数据包缓冲区
- Libpcap(一个抓包工具库)支持BPF
  - Libpcap是用户态的一个抓包工具
  - Libpcap几乎是系统无关的
- BPF是一种比较理想的抓包方案
  - 在核心态，所以效率比较高
  - 目前类UNIX系统的标准抓包内核模块

# BPF和libpcap



# 关于libpcap抓包库

- 用户态下的抓包库
- 系统独立的接口，C语言接口
  - 多种其他高级编程语言包装接口: **Perl, Python, Ruby, Tcl, Java, …**
- 广泛应用于
  - 网络统计软件
  - 入侵检测系统
  - 网络调试
- 支持过滤机制，**BPF**

# BPF过滤器

- **Man tcpdump (http://www.tcpdump.org/tcpdump_man.html)**
  - 善用过滤器规则是网络报文分析的关键
- **Examples:**
  - 达到某主机或从某主机发出的流量:
    **tcpdump host HOST**
  - 每个**TCP**会话的首包和尾包**(SYN**包**/FIN**包**):**
    **tcpdump 'tcp[tcpflags] & (tcp-syn|tcp-fin) != 0‘**
  - 除**echo requests/replies**之外的所有**ICMP**包**(i.e.,** 非**ping**包的**ICMP**报文**):**
    **tcpdump 'icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply'**

# libpcap/tcpdump文件格式

标准格式:tcpdump/libpcap的pcap文件格式


```
struct pcap_file_header {    //pcap文件头结构
        bpf_u_int32 magic;     // pcap文件magic标识，0xa1b2c3d4
        u_short version_major;  // pcap文件版本号
        u_short version_minor;  // pcap文件小版本号
        bpf_int32 thiszone;     // 时区修正(未使用)
        bpf_u_int32 sigfigs;    // 精确时间戳(未使用)
        bpf_u_int32  snaplen;  //  抓包最大长度(抓全数据包长度:
0x0000ffff，tcpdump缺省68)
        bpf_u_int32 linktype; // 链路层类型：一般都是1(以太网)
};
```

```
struct pcap_pkthdr { // 数据包捕获信息结构
      struct timeval ts;  // 捕获的时间戳
      bpf_u_int32 caplen; // 捕获数据长度
      bpf_u_int32 len;   // 数据包长度
};
```

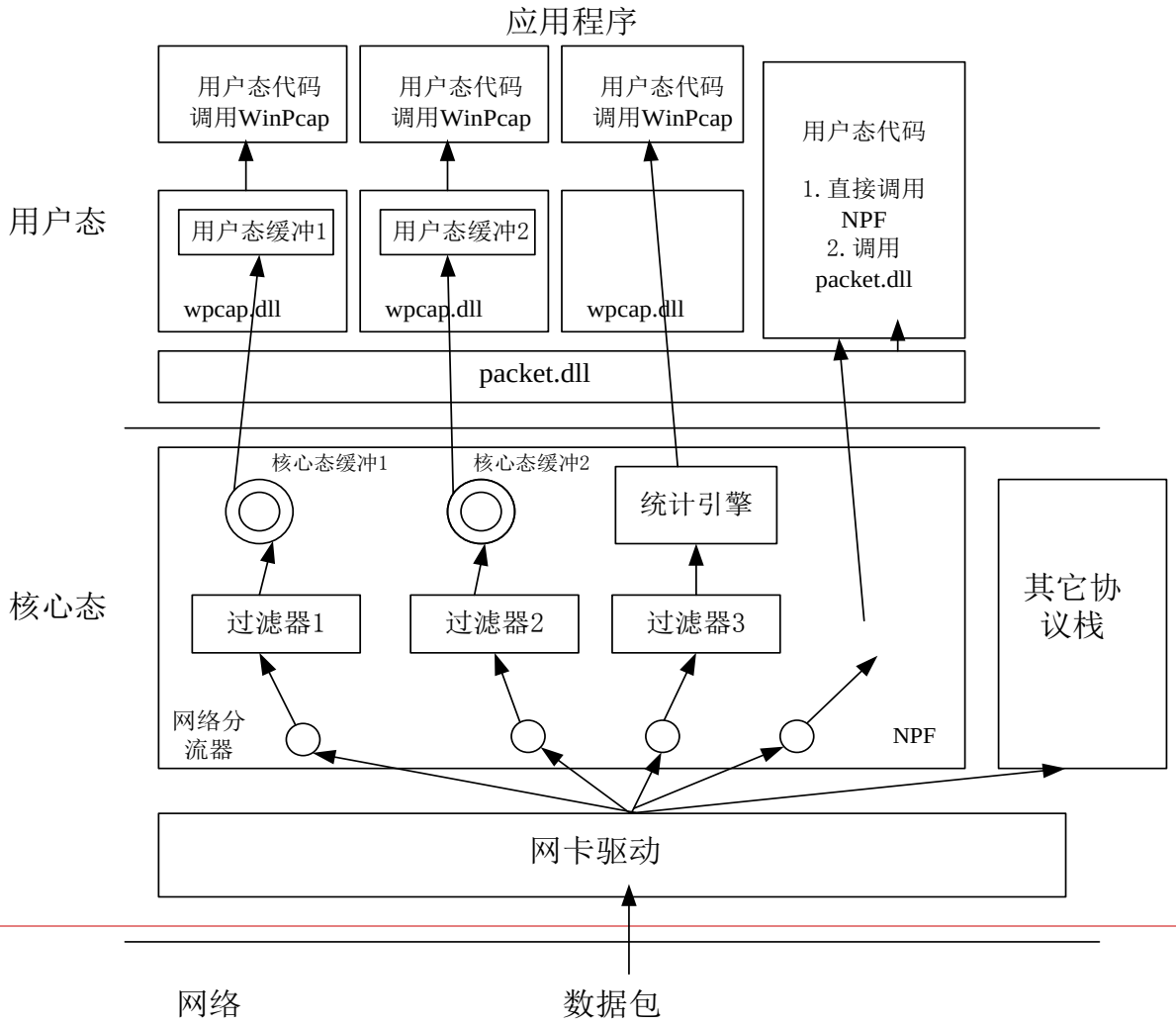
# Windows平台下的抓包技术

- 内核本身没有提供标准的接口
- 通过增加一个驱动程序或者网络组件来访问内核网卡驱动提供的数据包
  - 在**Windows**不同操作系统平台下有所不同
- 不同**sniffer**采用的技术不同
  - **WinPcap**是一个重要的抓包工具，它是**libpcap**的**Windows**版本

# WinPcap

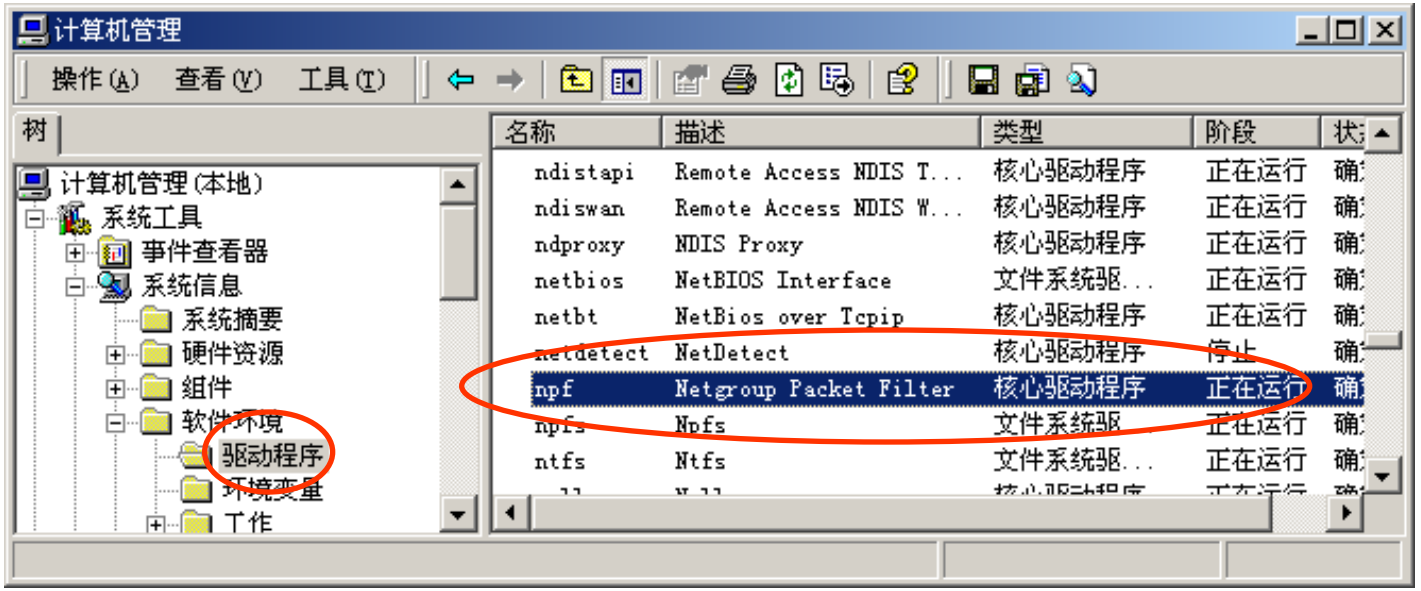
- **WinPcap包括三个部分**
  - 第一个模块**NPF(Netgroup Packet Filter)**，是一个虚拟设备驱动程序文件。它的功能是过滤数据包，并把这些数据包原封不动地传给用户态模块，这个过程中包括了一些操作系统特有的代码
  - 第二个模块**packet.dll**为**win32**平台提供了一个公共的接口。不同版本的**Windows**系统都有自己的内核模块和用户层模块。**Packet.dll**用于解决这些不同。调用**Packet.dll**的程序可以运行在不同版本的**Windows**平台上，而无需重新编译
  - 第三个模块 **Wpcap.dll**是不依赖于操作系统的。它提供了更加高层、抽象的函数。
- **packet.dll和Wpcap.dll**
  - **packet.dll**直接映射了内核的调用
  - **Wpcap.dll**提供了更加友好、功能更加强大的函数调用

# WinPcap和NPF



# NPF在Windows网络结构中位置

- **NDIS(Network Driver Interface Specification，网络驱动接口规范)描述了网络驱动与底层网卡之间的接口规范，以及它与上层协议之间的规范**
- **NPF作为一个核心驱动程序而提供的**



# WinPcap的优势

- 提供了一套标准的抓包接口
  - 与**libpcap**兼容，可使得原来许多类**UNIX**平台下的网络分析工具快速移植过来
  - 便于开发各种网络分析工具
- 除了与**libpcap**兼容的功能之外，还有
  - 充分考虑了各种性能和效率的优化，包括对于**NPF**内核层次上的过滤器支持
  - 支持内核态的统计模式
  - 提供了发送数据包的能力

# 网络嗅探器软件

- 类Unix平台网络嗅探器软件
  - Libpcap*抓包开发函数库
  - Tcpdump*以及wireshark*嗅探器软件
  - Snort*、dsniff、sniffit和linux_sniffer…
- Windows平台网络嗅探器软件
  - NPF/winpcap/windump
  - SnifferPro
  - Buttsniffer、NetMon、Network Associates Sniffer

# Libpcap开发函数库

| libpcap主要库函数 | 函数功能描述 |
| --- | --- |
| char *pcap_lookupdev(char *errbuf) | 获得一个网络设备名的指针 |
| int pcap_lookupnet(const char *device, bpf_u_int32 *netp, bpf_u_int32 *maskp, char *errbuf) | 获得指定网络设备的IP地址和掩码 |
| pcap_t *pcap_open_live(const char *device, int snaplen, int promisc, int to_ms, char *errbuf) | 打开指定设备的网络捕获 |
| int pcap_compile(pcap_t *p, struct bpf_program *fp, char *str, int optimize, bpf_u_int32 netmask) | 编译过滤规则 |
| int pcap_setfilter(pcap_t *p, struct bpf_program *fp) | 设置一个过滤规则 |
| int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user) | 捕获主循环，用来捕获数据包时用的,当捕获了符合要求数据包之后就交给回调函数callback来进行下一步处理. |

# 基于Libpcap实现简单的嗅探器

```
#include "netinet/in.h"
#include "arpa/inet.h"
#include "netinet/if_ether.h"
void my_callback(u_char  *useless,const  struct pcap_pkthdr *pkthdr,const u_char  *packet) /*  回调函数*/
{
      printf("I have got a packet \n");
}
dev = pcap_lookupdev(errbuf);
if (dev == NULL)
{
      fprintf(stderr,"%s\n",errbuf);
      exit(1);
}
pcap_lookupnet(dev,&netp,&maskp,errbuf);
descr = pcap_open_live(dev,BUFSIZ,1,-1,errbuf);
if (descr == NULL)
{
      printf("pcap_open_live():%s\n",errbuf);
      exit(1);
}
if (pcap_compile(descr,&fp,argv[1],0,netp)  == -1)
{
      fprintf(stderr,"Error calling pcap_compile\n");
      exit(1);
}
if (pcap_setfilter(descr,&fp)  == -1)
{
      fprintf(stderr,"Error setting filter\n");
      exit(1);
}
pcap_loop(descr,5,my_callback,NULL);
return 0;
```

# Tcpdump嗅探器软件

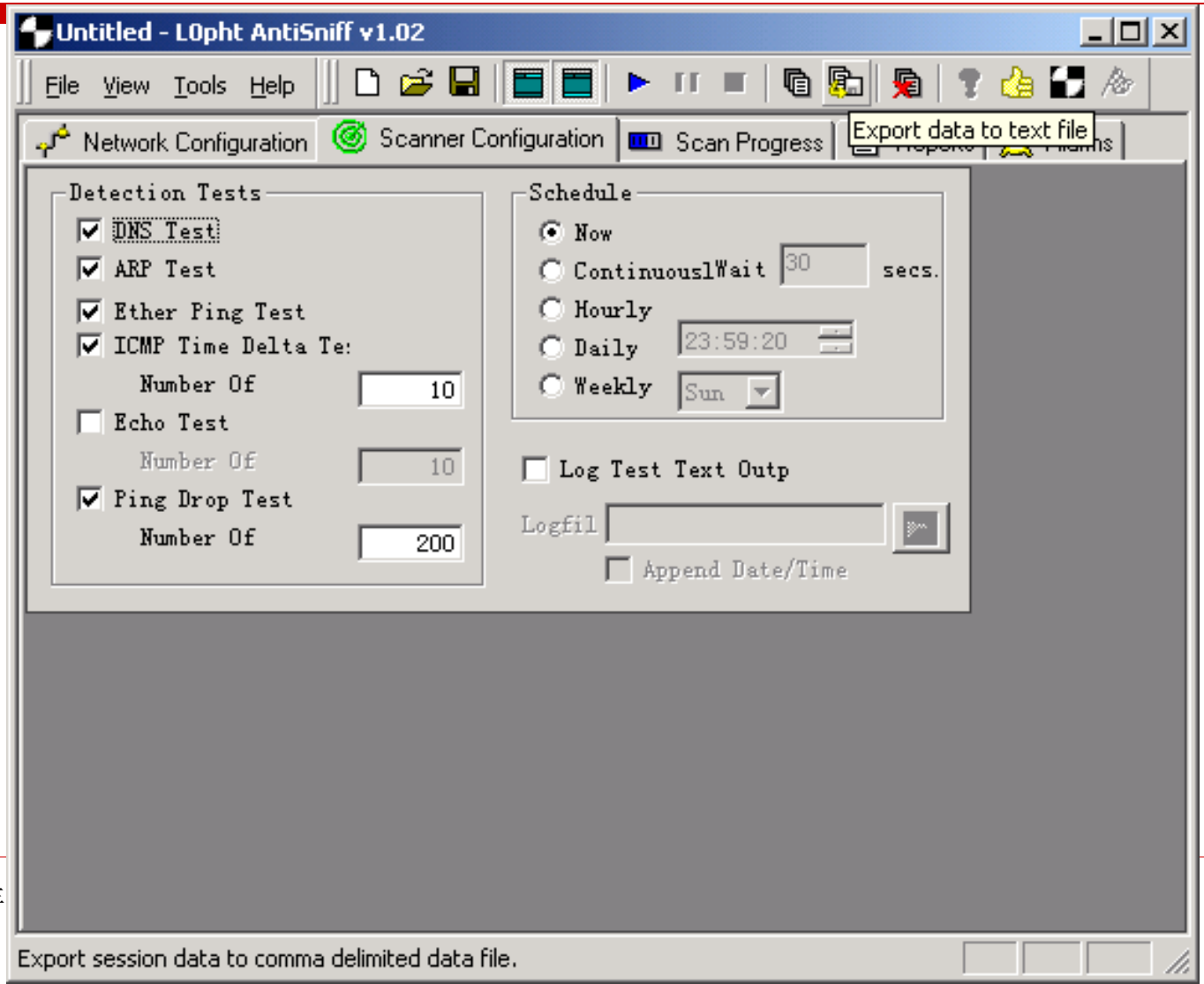
- 通用的命令行网络嗅探与数据包分析程序
- 绝大多数类Unix平台缺省安装
- 善用BPF过滤器规则 – 用好tcpdump的关键
- 实例：从源主机172.24.7.44向外连接的HTTP网络流量情况

```
root@ubuntu:~$ sudo tcpdump src 172.24.7.44 and tcp dst port 80
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode listening on eth0,
link-type EN10MB (Ethernet), capture size 96 bytes
    19:55:05.445585 IP ubuntu.local.34964 > 61.135.133.26.www: Flags [.], ack 3492103045, win 5840, length 0
    19:55:05.445726 IP ubuntu.local.34964 > 61.135.133.26.www: Flags [P.], seq 0:362, ack 1, win 5840, length 362
    19:55:05.571886 IP ubuntu.local.34964 > 61.135.133.26.www: Flags [.], ack 520, win 6432,
    length 0
.....
13 packets captured
59 packets received by filter
```

# 网络嗅探的检测技术

- 网卡和操作系统对于是否处于混杂模式会有一些不同的行为，利用这些特征可以判断一个机器是否运行在混杂模式下
- 一些检测手段
  - 根据操作系统的特征
    - **Linux**内核的特性：正常情况下，只处理本机**MAC**地址或者以太广播地址的包。在混杂模式下，许多版本的Linux内核只检查数据包中的IP地址以确定是否送到IP堆栈。因此，可以构造无效以太地址而IP地址有效的ICMP ECHO请求，看机器是否返回应答包(混杂模式)，或忽略(非混杂模式)。
    - **Windows 9x/NT**：在混杂模式下，检查一个包是否为以太广播包时，只看**MAC**地址前八位是否为**0xff**。
  - 根据网络和主机的性能
    - 根据响应时间：向本地网络发送大量的伪造数据包，然后，看目标主机的响应时间，首先要测得一个响应时间基准和平均值

# L0pht的AntiSniff产品



# 网络嗅探技术的防范措施

- 采用安全的网络拓扑
  - 共享式网络→交换式网络
  - 交换机上设置**VLAN**等技术手段，对网络进行合理的分段
- 共享式以太网→交换式以太网拓扑
  - 性能提升: 广播冲突域→每台主机单独冲突域
  - 安全性提升: 较难被网络监听
  - 交换式网络提供安全性仍可能被挫败: **ARP**欺骗
- 静态**ARP**或者**MAC-**端口映射表代替动态机制
- 重视网络数据传输的集中位置点的安全防范
- 避免使用明文传输口令/敏感信息网络协议, 使用加密协议
  - **telnet → ssh**
  - **IPSEC/TLS**

# 内容

1. 网络嗅探技术
2. 课堂实践：使用Tcpdump
3. 网络协议分析技术
4. 课堂实践：使用Wireshark
5. 作业4—解码网络扫描/网络扫描攻防对抗作业

# 课堂实践4.1-使用tcpdump

- 使用tcpdump开源软件对在本机上访问www.tianya.cn网站过程进行嗅探，回答问题：你在访问www.tianya.cn网站首页时，浏览器将访问多少个Web服务器？他们的IP地址都是什么？

# 内容

1. 网络嗅探技术
2. 课堂实践：使用Tcpdump
3. 网络协议分析技术
4. 课堂实践：使用Wireshark
5. 作业4—解码网络扫描/网络扫描攻防对抗作业

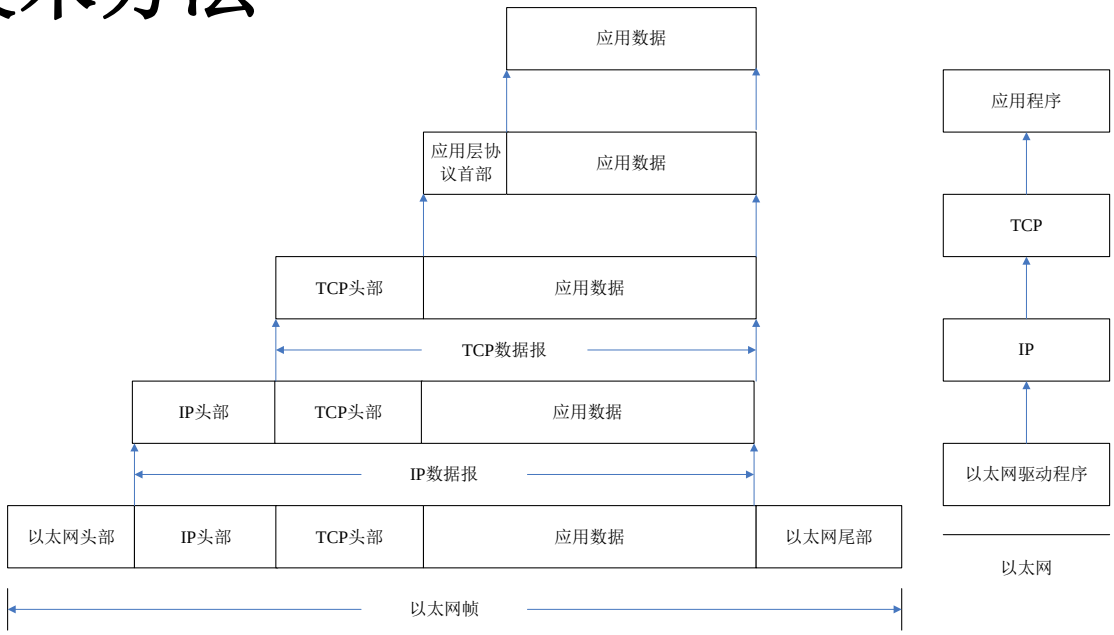
# 网络协议分析

- 网络协议分析的粒度和层次
  - 原始数据包: 最细粒度、最低层次
  - 网络流(/会话): 通过5元组进行流(/会话)重组
    - 5元组: sip, sport, dip, dport, ipproto
  - 网络流高层统计
    - IP会话列表<sip, sport>
    - 目标端口流统计<dport>

- 网络报文分析工具
  - 集成工具: Wireshark
  - 网络流重组: nstreams, snort
  - 高层统计和摘要分析: Netflow, RRDTools

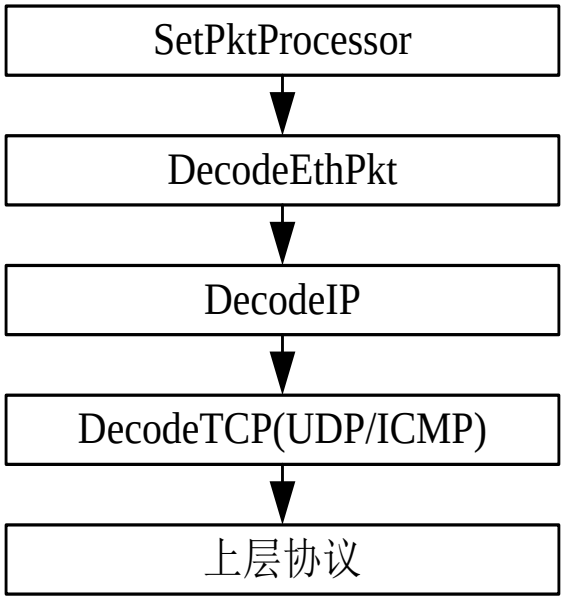
# 原始数据包粒度网络协议分析

- 对网络上传输的二进制格式数据包进行解析，以恢复出各层网络协议信息以及传输内容的技术方法



# 网络协议分析技术实现

- 实现参考源码
  - **Snort**中的网络解码器模块
  - **decode.c/decode.h**
- 解析以太网数据帧 **DecodeEthPkt**
  - 预处理：拆包前进行一些前期处理
  - 拆包：将当前得到的包内存位置赋给**Packet**数据结构中相应的指针**eh(EtherHdr)**型的指针即可
  - 解析上层协议： **switch**语句，根据**ether_type**分别调用相应的上层协议解析例程




```
typedef struct _EtherHdr
{
    u_int8_t  ether_dst[6] ;
    u_int8_t  ether_src[6] ;
    u_int16_t ether_type   ;
} EtherHdr ;
```

# DecodeEthPkt() in snort decode.c

```
1  void DecodeEthPkt(Packet * p, struct pcap_pkthdr * pkthdr, u_int8_t * pkt)
2  {
3      u_int32_t pkt_len;      /* suprisingly, the length of the packet */
4      u_int32_t cap_len;      /* caplen value */
5
6      bzero((char *) p, sizeof(Packet));
7
8      p->pkth = pkthdr;
9      p->pkt = pkt;
10
11     /* set the lengths we need */
12     pkt_len = pkthdr->len;  /* total packet length */
13     cap_len = pkthdr->caplen;   /* captured packet length */
14
15     if(snaplen < pkt_len)
16         pkt_len = cap_len;
17
18     /* lay the ethernet structure over the packet data */
19     p->eh = (EtherHdr *) pkt;
20
```

```
/*
 * Ethernet header
 */

typedef struct _EtherHdr
{
    u_int8_t ether_dst[6];
    u_int8_t ether_src[6];
    u_int16_t ether_type;

}          EtherHdr;
```

# DecodeEthPkt() in snort decode.c – con'd

```
21      /* grab out the network type */
22      switch(ntohs(p->eh->ether_type))
23      {
24          case ETHERNET_TYPE_PPPoE_DISC:
25          case ETHERNET_TYPE_PPPoE_SESS:
26              DecodePPPoEPkt(p, pkthdr, pkt);
27              return;
28
29          case ETHERNET_TYPE_IP:
30              DecodeIP(p->pkt + ETHERNET_HEADER_LEN, cap_len - ETHERNET_HEADER_LEN, p);
31              return;
32
33          case ETHERNET_TYPE_ARP:
34          case ETHERNET_TYPE_REVARP:
35              pc.arp++;
36              DecodeARP(p->pkt + ETHERNET_HEADER_LEN, cap_len - ETHERNET_HEADER_LEN, p);
37              return;
38
39          case ETHERNET_TYPE_IPV6:
40              pc.ipv6++;
41              if(pv.showipv6_flag)
42                  DecodeIPV6(p->pkt + ETHERNET_HEADER_LEN, (cap_len - ETHERNET_HEADER_LEN));
43              return;
44
45          case ETHERNET_TYPE_IPX:
46              pc.ipx++;
47              if(pv.showipx_flag)
48                  DecodeIPX(p->pkt + ETHERNET_HEADER_LEN, (cap_len - ETHERNET_HEADER_LEN));
49              return;
50
51          case ETHERNET_TYPE_8021Q:
52             DecodeVlan(p->pkt + ETHERNET_HEADER_LEN, cap_len - ETHERNET_HEADER_LEN, p);
53             return;
54
55          default:
```

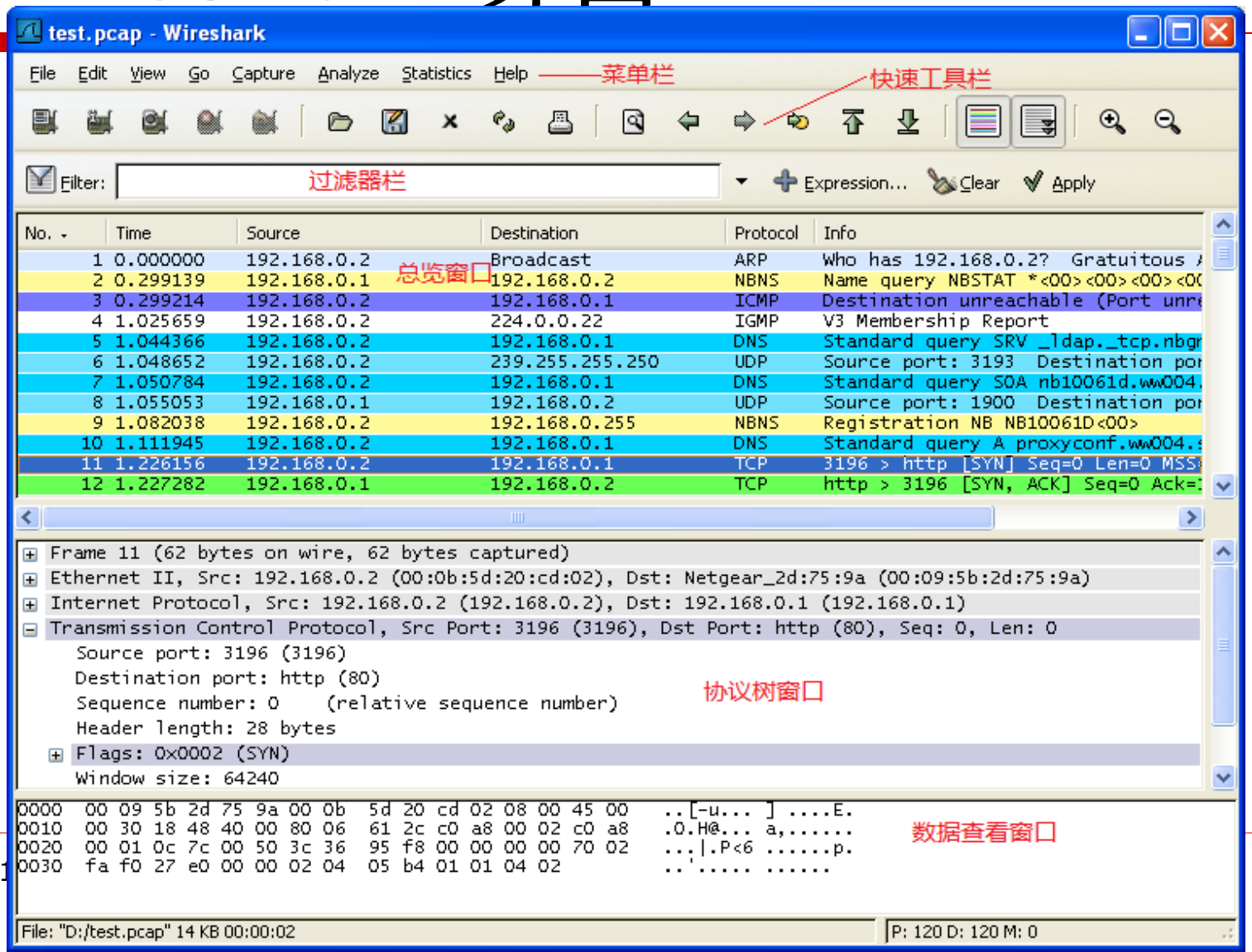
- PPPoE
- IP
- ARP, RARP
- IPv6
- IPX
- 802.1Q

```
case ETHERNET_TYPE_IP:
    DecodeIP(p->pkt + ETHERNET_HEADER_LEN, cap_len - ETHERNET_HEADER_LEN, p);
    return;
```

# Wireshark* (ethereal)

- Wireshark (ethereal)
  - 1998-2006: Ethereal
    - Gerald Combs, University of Missouri-Kansas City
  - 2006-now: wireshark
    - Renamed from Ethereal due to trademark issues
    - eWEEK Labs named Wireshark one of "The Most Important Open-Source Apps of All Time“
- Wireshark特性
  - 图形化界面/命令行(tshark)
  - 在线/离线抓包(支持标准pcap二进制日志文件)
  - 支持BPF过滤器
  - 支持分析几百种常见网络协议
  - 跨平台：类UNIX、Win32(依赖libpcap/WinPcap)

# Wireshark界面

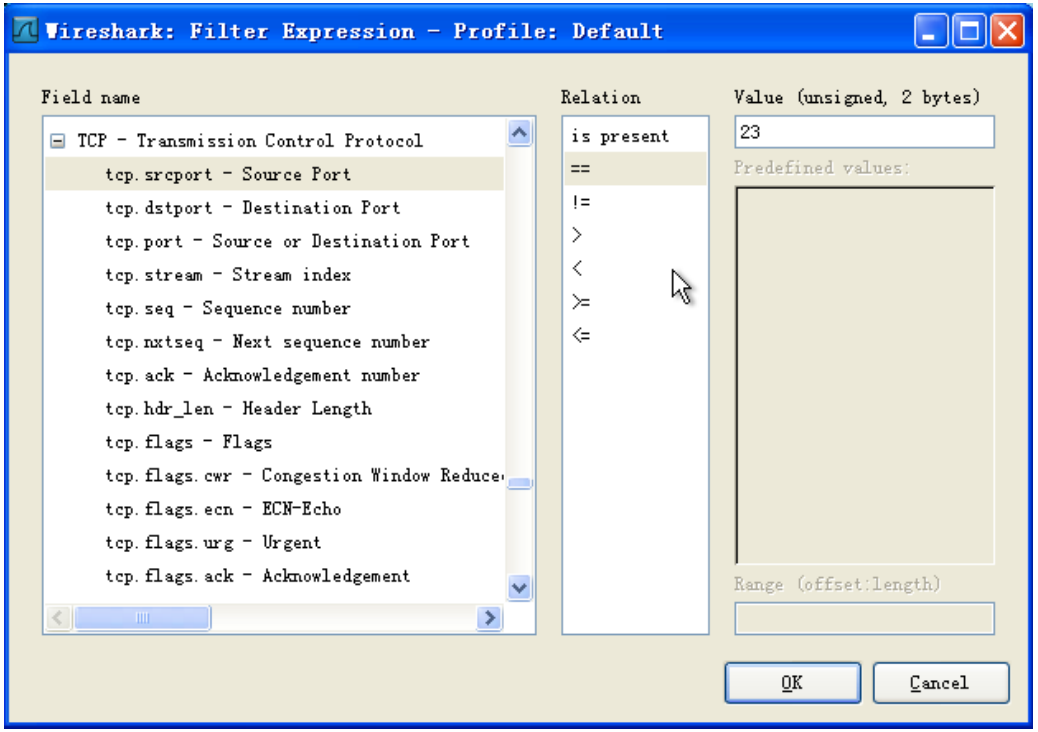


# Wireshark基本功能

- 抓包(Capture)
  - Capture Filter: BPF过滤器
- 分析(Analyze)
  - 自动协议解码: 支持数百种协议, 显示各层包头和内容字段
  - 灵活选择协议对网络流进行解码 Decode As…
- 统计(Statistics)
  - 协议分类(Protocol Hierarchy)
  - 会话列表(Conversations)
    - 2层(以太网)/3层(IP)/4层(TCP,UDP)
  - 会话终端(EndPoints)
  - I/O Graph: 随时间统计的流量曲线
  - 会话重组(Follow TCP/UDP Stream)和会话图(Flow Graph)

# Wireshark中的两类过滤规则

- 嗅探过滤规则
  - 支持BPF规则
  - 用于嗅探抓包时的过滤
- 显示过滤规则
  - 用于在界面中选择显示哪些数据包
  - 与BPF规则有所不同



# 流重组/会话重组

- 流重组/会话重组
  - **TCP/UDP会话发送字节数可能很大**
  - **IP包最大长度(64K-20≈64K)**
  - 以太网帧最大长度**(1500-20=1480)**
  - 协议栈发送大量**TCP/UDP**报文时，必然分组传送
  - 流重组**:** 将同属于一个**TCP/UDP**会话的**IP**包负载按序重新组装，还原应用层数据的过程
- 流重组工具
  - **Wireshark: Follow TCP/UDP Stream**
  - **nstreams:**
    - **nstreams -f pcap_file > nstreams.txt**
  - **Snort:**
    - **Log规则(snort.conf): log tcp any any <> any any (sid:1000001; session: printable;)**
    - **snort -r pcap_file –l ./log -c snort.conf**

# 网络流记录和高层统计分析

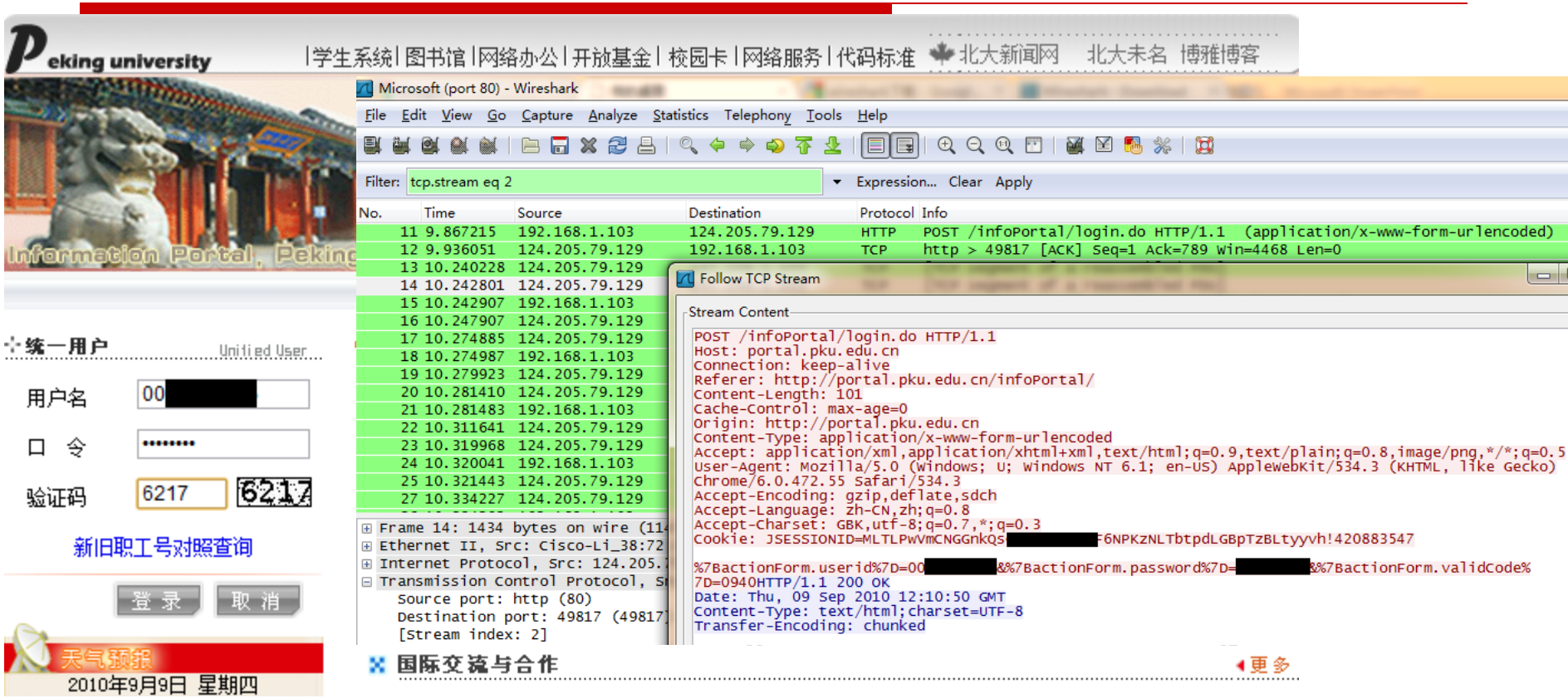
- **Netflow**
  - 定义了网络会话流记录的业界标准-**Cisco**
    - **RFC 3334/3954/3955**
  - **IP Flow Information Export (netflow v10)-IETF**

- 网络流记录
  - 商业路由器、交换机支持**Netflow**日志输出
  - 开源软件: **nfdump(**支持**Netflow**标准**), Argus**

- 网络流分析
  - 基于**pcap**文件上的流重组和统计分析
    - **Wireshark:** 协议分类/会话列表…
    - **SnifferPro**
  - **Netflow Analyzer: HP openview/cacti/nfsen**

# Wireshark工具的演示

# 对校园门户网站的监听和协议分析




- %7BactionForm.userid%7D=00XXXXXXXX
- &%7BactionForm.password%7D=********

# 内容

1. 网络嗅探技术
2. 课堂实践：使用Tcpdump
3. 网络协议分析技术
4. 课堂实践：使用Wireshark
5. 作业4—解码网络扫描/网络扫描攻防对抗作业

# 课堂实践4.2-使用Wireshark

- 任务：使用**wireshark**开源软件对在本机上以**TELNET**方式登录**BBS**进行嗅探与协议分析，回答如下问题并给出操作过程：
  - 你所登录的**BBS**服务器的**IP**地址与端口各是什么？
  - **TELNET**协议是如何向服务器传送你输入的用户名及登录口令？
  - 如何利用**wireshark**分析嗅探的数据包，并从中获取你的用户名及登录口令？

# 内容

1. 网络嗅探技术
2. 课堂实践：使用Tcpdump
3. 网络协议分析技术
4. 课堂实践：使用Wireshark
5. 作业4—解码网络扫描/网络扫描攻防对抗作业

# 作业4—解码网络扫描/网络扫描攻防对抗作业

- 团队作业(合作完成)
- **4.1 取证分析实践作业 – 解码网络扫描(5分+1分bonus)**
- **4.2 攻防对抗实践作业 – 网络扫描攻防(5分)**
- 提交
  - **Deadline: 11月3日**

# 作业4.1－解码网络扫描(5分)

- 难度等级：入门级
- 挑战内容：这次案例分析挑战是完全为刚入门的安全分析师准备的，目标是分析由人为构造的到一台蜜罐主机的**5**次不同类型端口扫描。需要指出的是，这次案例分析中的端口扫描流量并不是从“野外”捕获的，而是特意构造的，这次入门级的案例分析挑战的目的完全是为了提供学习和训练的机会。
- 网络入侵检测器**-snort**捕获了每次扫描的流量并存入了**tcpdump**格式二进制网络日志文件中。这次挑战的任务每组从这**5**次扫描的日志文件中随机选择**2**个，并分析这两个文件，回答所列的问题，并撰写详细的实验分析报告。通过这次挑战，你能够学习到数据包抓取技术的使用方法，以及使用数据包解码工具**tcpdump**或**wireshark**分析网络数据包的技能。

# 作业4.1—解码网络扫描(问题)

- 1．攻击主机的IP地址是什么？(0.5)
- 2．网络扫描的目标IP地址是什么？ (0.5)
- 3．本次案例中是使用了哪个扫描工具发起这些端口扫描？你是如何确定的？(1)
- 4．你所分析的日志文件中，攻击者使用了那种扫描方法，扫描的目标端口是什么，并描述其工作原理。(2)
- 5．在蜜罐主机上哪些端口被发现是开放的？(1)
- 6．额外奖励问题(1)：攻击主机的操作系统是什么？

# 作业4.1提示

- 使用工具
  - **wireshark(ethereal)**为主
  - **snort**可为辅－检测扫描工具的特征
  - 结合使用命令行与图形界面：高手倾向于使用命令行，但不意味着完全排斥**GUI**
- 分析关键点
  - 确认发起扫描使用的工具，然后对照它的使用说明和你看到的数据包特征
  - 注意每次扫描的开始和结尾区域，看是否有不一样的数据包
  - 善用**tcpdump filter**选择性的关注数据包

# 作业4.2
# —网络扫描攻防对抗实践(5分)

- 难度等级：入门级
- 基于作业2中构建的网络攻防实验环境
- 攻击方用nmap扫描（达到特定目的），防守方tcpdump嗅探，用wireshark分析，并分析出攻击方的扫描目的以及每次扫描使用的nmap命令。
- 撰写实验分析报告。

# Thanks

诸葛建伟

zhugejw@gmail.com