黑客攻防

嗅探器

Unit06

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	知识讲解
	10:30 ~ 11:20	
	11:30 ~ 12:00	
下午	14:00 ~ 14:50	实训案例
	15:00 ~ 15:50	
	16:00 ~ 16:50	扩展提高
	17:00 ~ 17:30	总结和答疑

知识讲解

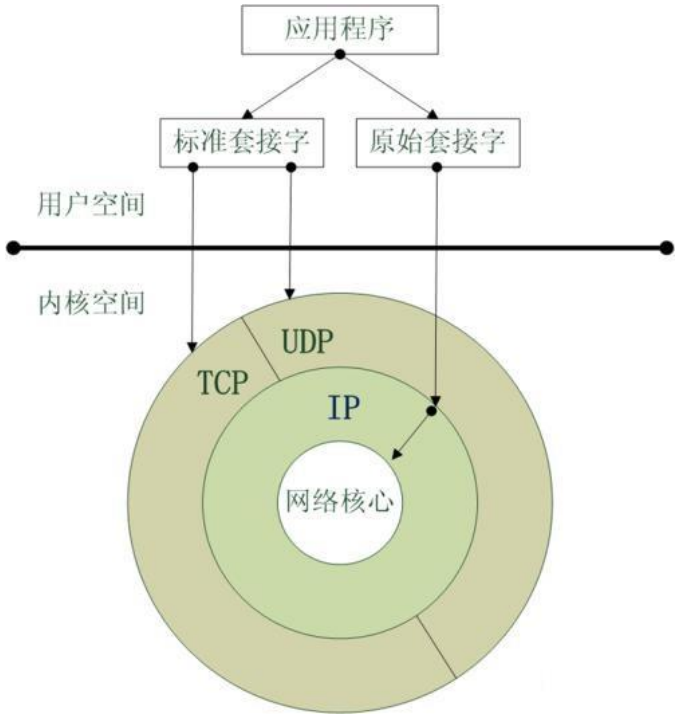

知识讲解
原始套接字
原始套接字的特点
原始套接字的相关操作
数据包的处理
TCP/IP协议栈
TCP/IP协议栈
数据封装与解析
数据封装与解析

原始套接字

原始套接字

- Linux网络协议栈中的套接字主要分为三类：
  - 流式套接字：工作在传输层TCP协议之上
  - 数据报套接字：工作在传输层UDP协议之上
  - 原始套接字：工作在网络层或物理层之上

知识讲解



原始套接字的特点

- 读写ICMP/IGMP报文
  - 对于ICMP、IGMP等封装在IP数据包内部但又在传输层以下的报文，内核总会为协议类型匹配的原始套接字传递一份拷贝。通过这种方式，可以在用户空间处理这些报文，而无需内核参与，从而减轻系统内核的负担
- 读写部分IP报文
  - 内核对于无法识别协议类型的IP报文，会首先查找有没有合适的原始套接字，如果有这样的套接字，内核会将该报文复制一份给它们，否则直接丢弃该报文

知识讲解

原始套接字的特点

- 直接通过物理层捕获数据包
  - 对于使用TCP、UDP等传输层协议的IP报文，内核可以识别，会交给相应的功能模块进行处理，而不会传递给原始套接字，除非创建套接字时指定了PF_PACKET协议族(socket函数的第一个参数)和ETH_P_IP或ETH_P_ALL通信协议(socket函数的第三个参数)。这样的原始套接字将直接通过物理层捕获数据包
- 自己构造IP包头
  - 原始套接字可以自己构造IP包头，但必须先通过setsockopt函数为该套接字设置IP_HDRINCL选项，以告诉内核IP包头由自己填充而不由内核填充
- 需要超级用户权限
  - 创建原始套接字需要超级用户权限

知识讲解

原始套接字的相关操作

- 创建原始套接字

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

知识讲解

  - 第一个参数domain表示协议族，一般情况下用PF_INET或AF_INET表示互联网协议族，二者可以认为是等价的。严格来讲PF_INET是协议族(Protocol Family)，而AF_INET则是地址族(Address Family)。按照最初的设计，一个协议族可以包含多个地址族，但迄今为止每个协议族下面仅有一个地址族。在有关套接字的头文件中有如下宏定义：

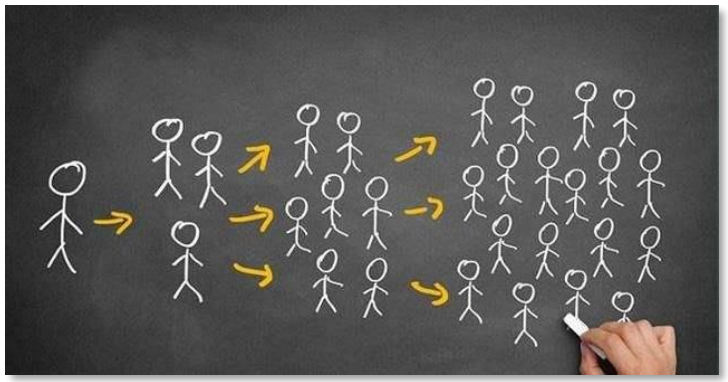
```
#define PF_INET AF_INET
```

  - 这两个宏在数值上相等，功能上亦无差别。另外，如果需要直接通过物理层捕获数据包，可将domain参数设置为PF_PACKET

原始套接字的相关操作

- 创建原始套接字
  - 第二个参数type表示套接字类型。SOCK_RAW表示原始套接字
  - 第三个参数protocol表示通信协议。如果想通过原始套接字捕获TCP或UDP包，需要将此参数设置为ETH_P_IP或ETH_P_ALL。出于兼容性的考虑，最好先通过htons函数将这两个宏转换为网络字节序短整型
  - 创建原始套接字需要超级用户权限，否则socket函数会返回-1，同时将errno设置为EACCES

知识讲解



原始套接字的相关操作

- 绑定和连接
  - 通常情况下，原始套接字并不需要绑定操作。如果绑定也只是绑定IP地址而不会涉及端口号：
    - 绑定到某个特定IP地址的原始套接字：
      - 接收数据时，只能接收以此IP为目的地址的数据包
      - 发送数据时，数据包的源地址会被内核设置为此IP
    - 不绑定任何特定IP地址的原始套接字：
      - 接收数据时，可接收以任意IP为目的地址的数据包
      - 发送数据时，数据包的源地址会被设置为接口主IP
  - 不同主机上的原始套接字可以通过connect函数彼此连接，同样只连接IP地址而不涉及端口号。连接成功以后可以通过write或send函数发送数据，read或recv函数接收数据，而发送的目的和接收的源即为连接的对方主机

知识讲解

原始套接字的相关操作

- 读写原始套接字
  - 通过原始套接字发送数据，默认情况下，由内核负责对IP包头的填充，但如果通过setsockopt函数为该套接字设置了IP_HDRINCL选项，则必须由用户程序填充IP包头，内核只负责计算并填充IP包头的校验和。当数据大于链路的最大传输单元(Maximum Transmission Unit, MTU)时，内核将自动对数据包进行分片
  - 读写原始套接字和读写普通套接字并没有显著的差别：
    - sendto/recvfrom：向/从任意主机发送/接收数据包
    - send/recv：向/从已连接(connect)的对方主机发送/接收数据包
    - write/read：与send/recv等价

知识讲解

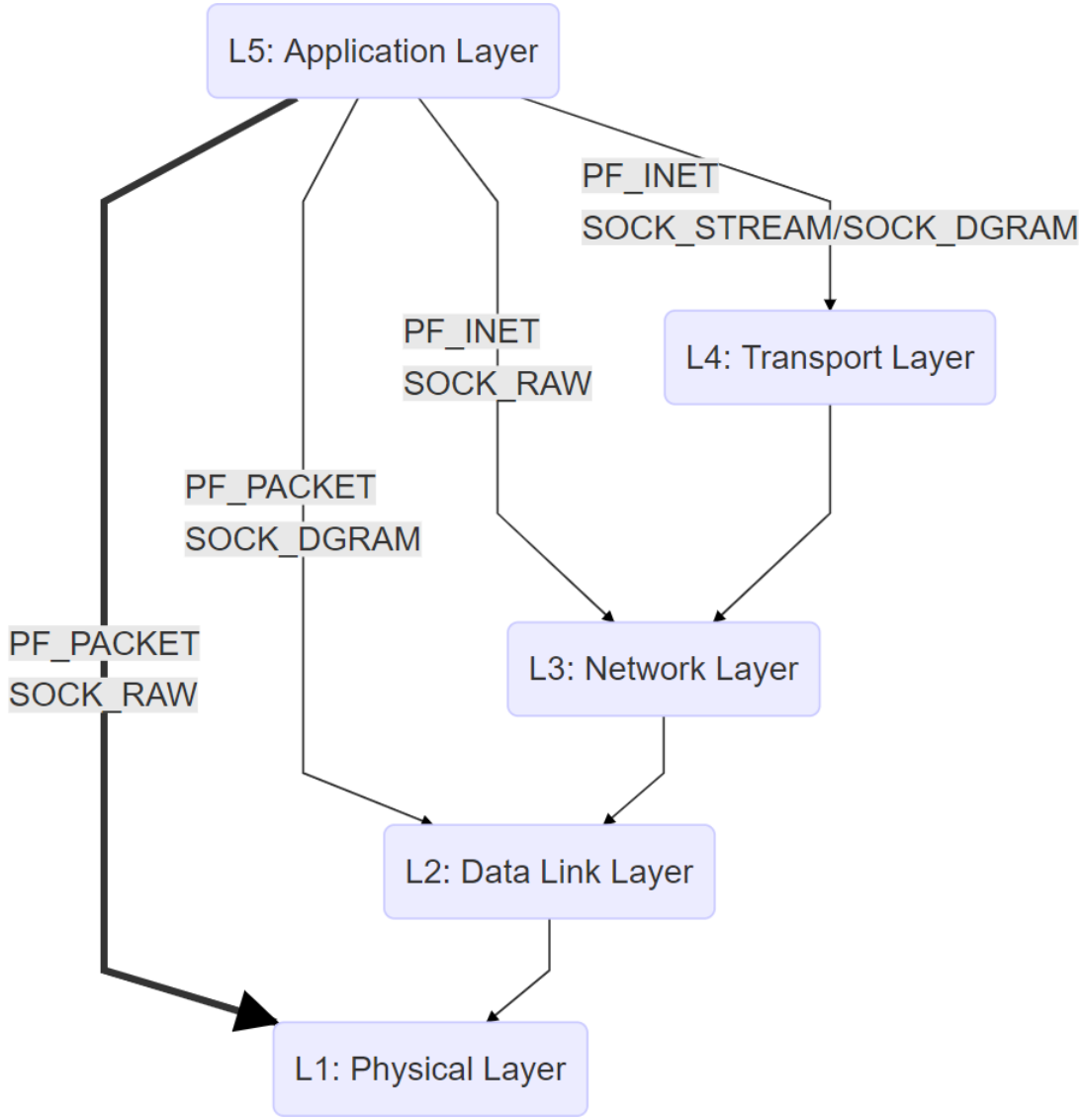
数据包的处理

- 当内核收到一个无法识别协议类型的IP报文时，会检查系统中所有进程的所有原始套接字，根据以下原则决定是否将该IP报文拷贝给相应的套接字：
  - 套接字的通信协议(socket函数的第三个参数)与该IP报文的通信协议完全匹配
  - 套接字的绑定地址(调用过bind函数)与该IP报文的目的地址完全匹配
  - 套接字的连接地址(调用过connect函数)与该IP报文的源地址完全匹配
- 如果一个原始套接字的通信协议(socket函数的第三个参数)为0，并且没有绑定(bind)和连接(connect)，那么该套接字将能捕获到内核收到的除TCP和UDP以外的所有IP报文
- 如果希望原始套接字也能捕获TCP和UDP数据包，可在创建该套接字时指定PF_PACKET协议族(socket函数的第一个参数)和ETH_P_IP或ETH_P_ALL通信协议(socket函数的第三个参数)。这样的原始套接字将从物理层捕获数据包

知识讲解

数据包的处理

- 协议族和套接字类型与网络协议栈的关系如下图所示：

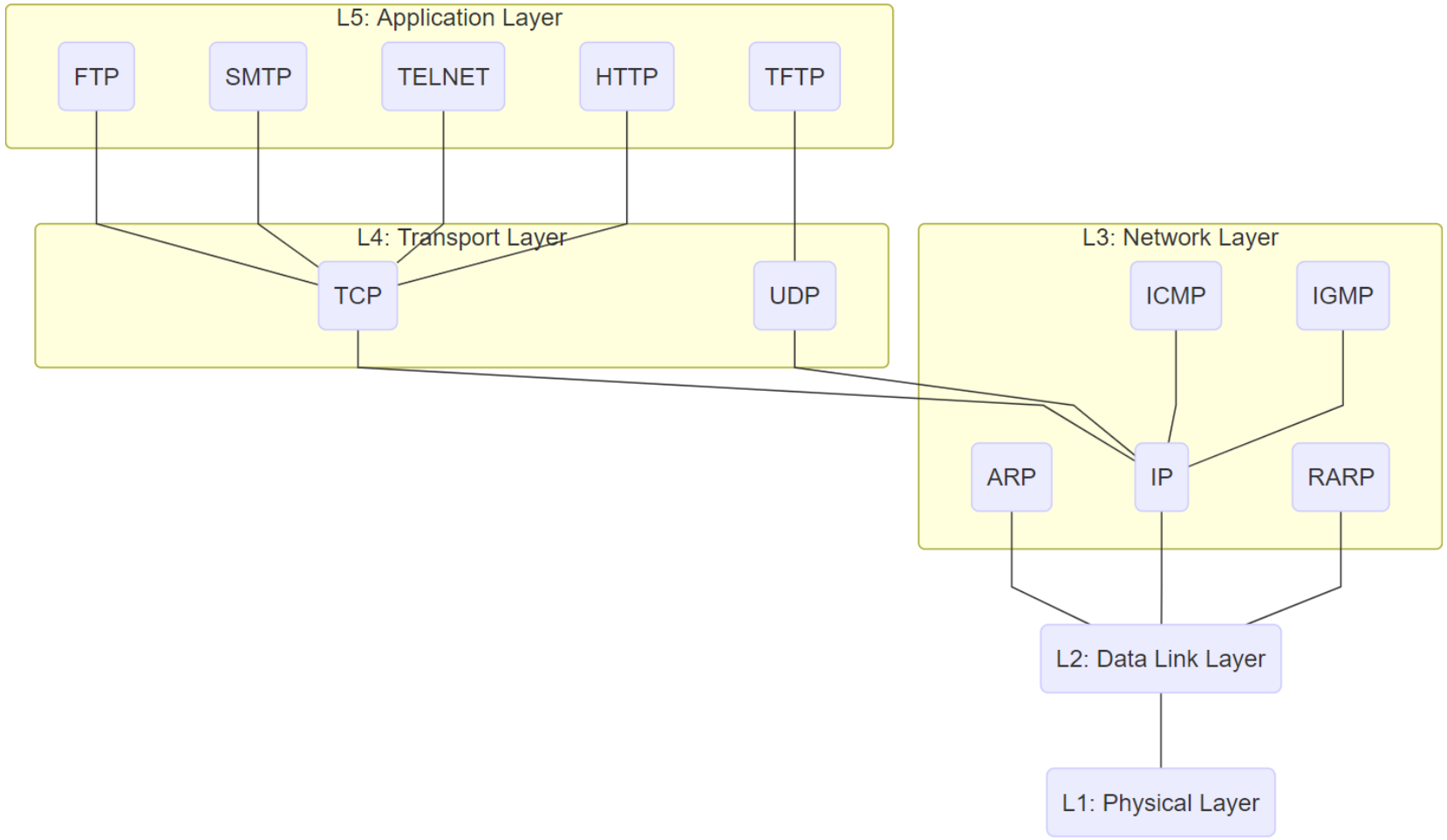


知识讲解

TCP/IP协议栈

TCP/IP协议栈

- TCP/IP协议栈模型中的常见协议及其位置如下图所示：



知识讲解

数据封装与解析

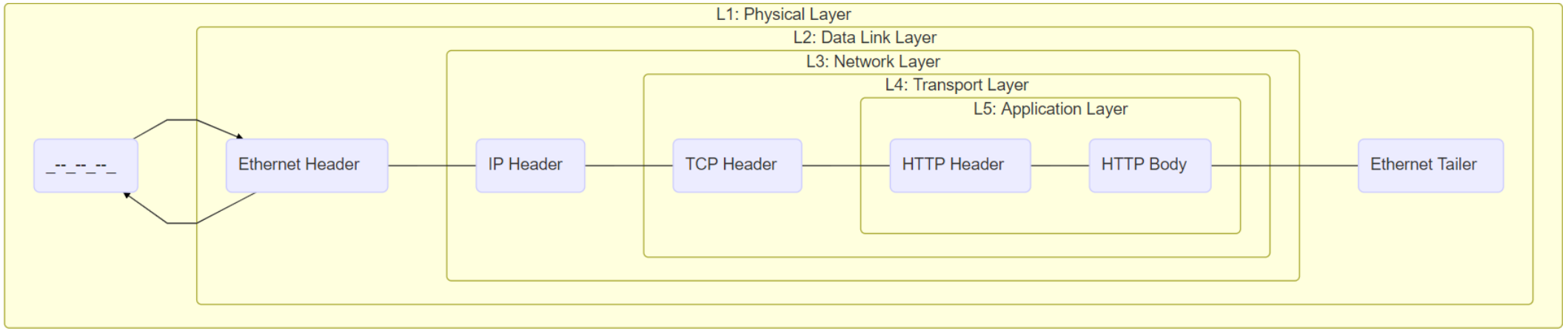
数据封装与解析

- 当应用程序通过协议栈向网络发送数据时，应用层的数据要依次经历传输层、网络层、数据链路层，最后进入物理层。每一层都要为数据添加该层的头部(有的层还有尾部)信息，以实现层次化控制。这个过程称为数据封装
- 当网络接口设备，如网卡，从网络接收数据时，物理层的数据要依次经历数据链路层、网络层、传输层，最后进入应用层。每一层都要对该层的头部(有的层还有尾部)信息进行解析，最后得到用户数据。这个过程称为数据解析

知识讲解

数据封装与解析

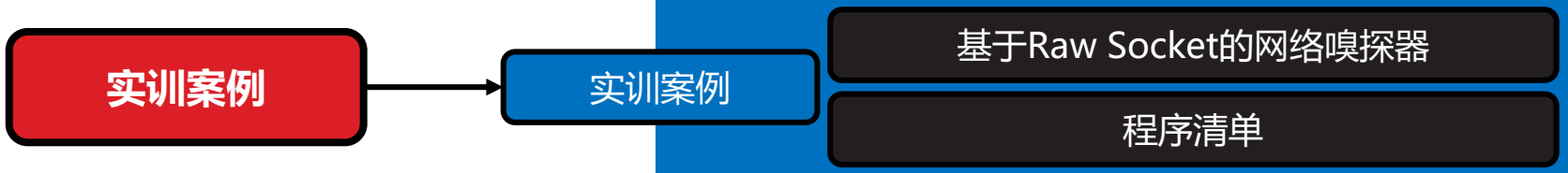
- 以HTTP报文为例，数据包的层次化结构与网络协议栈的关系如下图所示：



- 嗅探器(Sniffer)的工作原理就是利用原始套接字捕获流经主机网卡的数据包，并按上图所示封装结构，解析每一层的头(尾)部信息，将解析结果显示出来

知识讲解

实训案例



实训案例

基于Raw Socket的网络嗅探器

- 应用有关原始套接字的知识，编写一个网络嗅探器，捕获网络数据包并分析其基本信息，例如IP地址、端口号、协议类型、物理地址等
- 实现简单的过滤器功能，捕获指定的数据包，例如捕获指定IP地址、指定协议的数据包

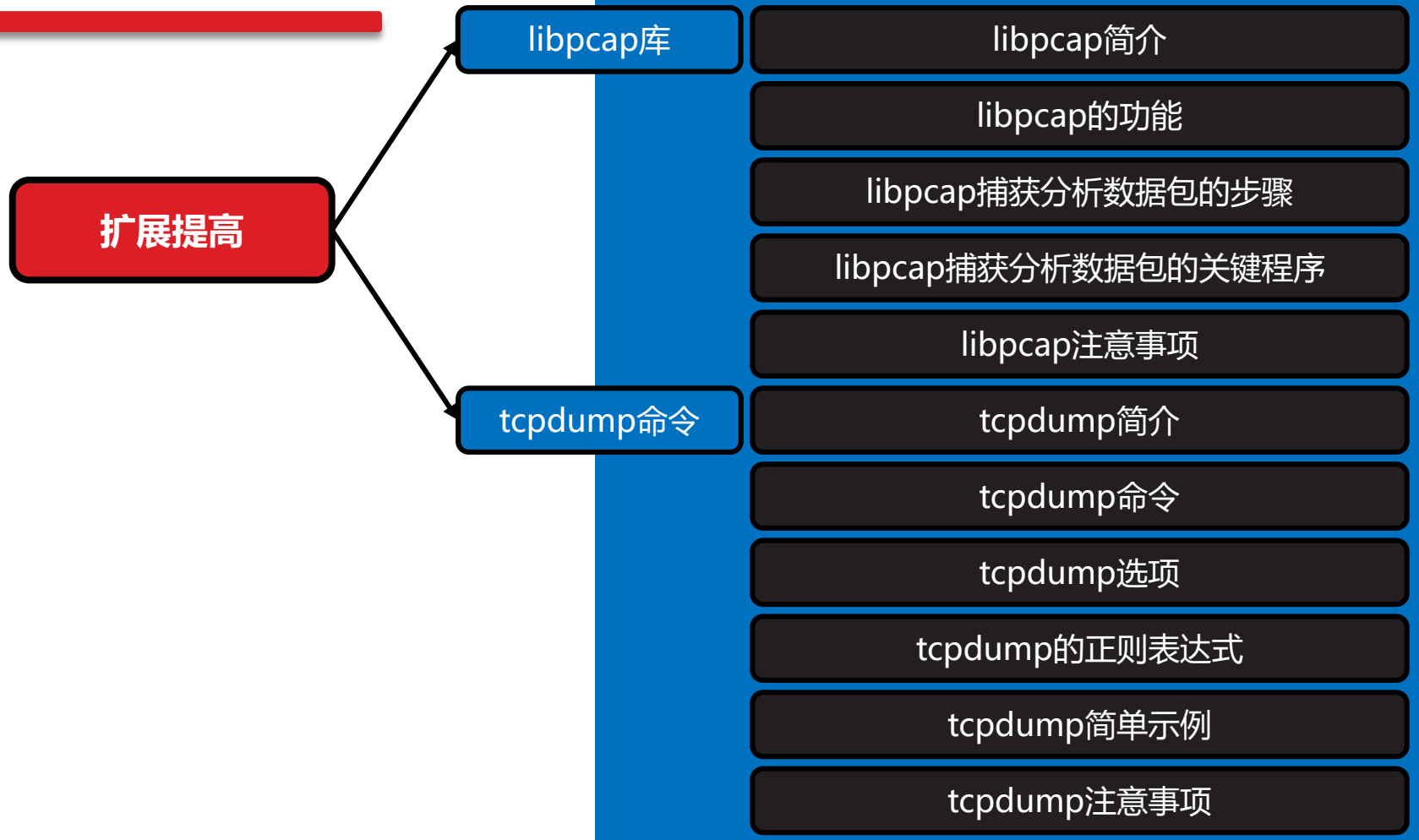
代码实践

程序清单

- 包结构
  - packet.h
- 声明RawSocket类
  - rawsocket.h
- 实现RawSocket类
  - rawsocket.cpp
- 声明Sniffer类
  - sniffer.h
- 实现Sniffer类
  - sniffer.cpp
- 测试Sniffer类
  - sniffer_test.cpp
- 测试Sniffer类构建脚本
  - sniffer_test.mak

代码实践

扩展提高


扩展提高
libpcap库
libpcap简介
libpcap的功能
libpcap捕获分析数据包的步骤
libpcap捕获分析数据包的关键程序
libpcap注意事项
tcpdump命令
tcpdump简介
tcpdump命令
tcpdump选项
tcpdump的正则表达式
tcpdump简单示例
tcpdump注意事项

通过libpcap库捕获数据包

libpcap简介

- 对数据包的捕获也可以借助Linux下的libpcap开发包来实现。该开发包由加州大学伯克利分校的Van Jacobson、Craig Leres和Steven McCanne合作开发，它提供了丰富的API函数，可以帮助程序员快速开发数据包捕获软件
- libpcap的官方网址：http://www.tcpdump.org
- libpcap具有如下特点：
  - 对数据包捕获功能进行封装，易于使用
  - 可以设置各种复杂的组合过滤规则，捕获用户感兴趣的数据包
  - 过滤模块在内核层次实现，效率非常高
- 很多著名的网络抓包工具，如tcpdump，和网络入侵检测工具，如snort，都是基于libpcap开发的

知识讲解

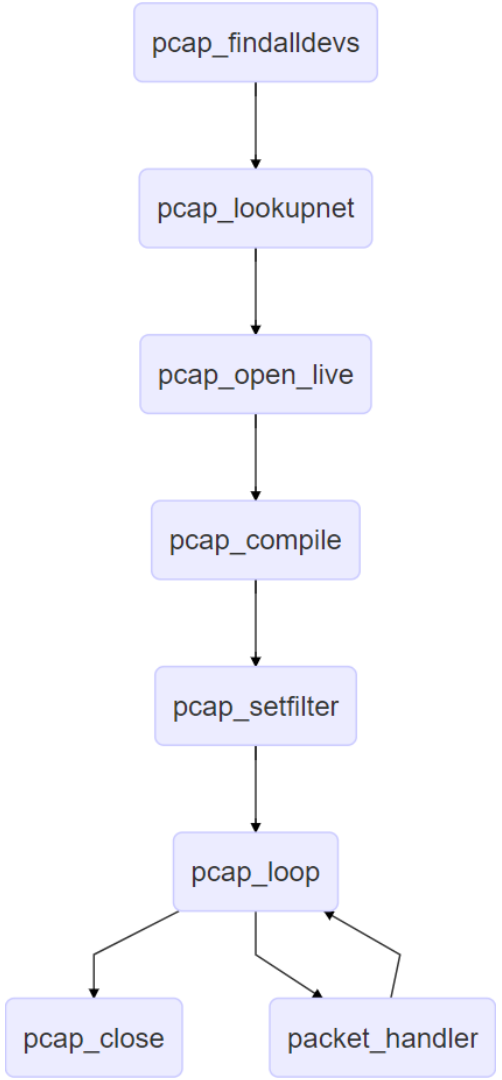
libpcap的功能

- 捕获数据包
  - 使用libpcap可以方便、高效地捕获网络数据包
- 过滤数据包
  - libpcap可以在内核层次对数据包进行过滤，不仅效率高而且过滤规则非常详细，可以进行各种复杂的组合，实现强大的过滤功能
- 分析数据包
  - libpcap在捕获数据包的同时还提供了一些辅助信息，如捕获时间、数据包长度等，可以帮助开发者更好地分析数据包
- 存储数据包
  - libpcap可将捕获到的数据包存储到本地，甚至可以离线方式对本地文件中的数据包进行分析

知识讲解

libpcap捕获分析数据包的步骤

- 使用libpcap对网络数据包进行捕获和分析的步骤如下：
  - 获取设备列表：调用pcap_findalldevs函数
  - 获取网络地址和子网掩码：调用pcap_lookupnet函数
  - 打开指定的设备：调用pcap_open_live函数
  - 编译过滤规则：调用pcap_compile函数
  - 设置过滤规则：调用pcap_setfilter函数
  - 捕获分析数据包：调用pcap_loop函数
  - 关闭设备：调用pcap_close函数
- 如下图所示：



知识讲解

libpcap捕获分析数据包的关键程序

- 获取设备列表

```
if (pcap_findalldevs(&devices, errbuf) == -1) {
    fprintf(stderr, "pcap_findalldevs: %s\n", errbuf);
    return -1;
}
```

- 获取网络地址和子网掩码

```
if (pcap_lookupnet(devname, &net_ip, &net_mask, errbuf) == -1) {
    fprintf(stderr, "pcap_lookupnet: %s\n", errbuf);
    return -1;
}
```

知识讲解

libpcap捕获分析数据包的关键程序

- 打开指定的设备

```
if ((dev_handle_pcap = pcap_open_live(devname, BUFSIZ, 1, 0, errbuf)) == NULL) {
    fprintf(stderr, "pcap_open_live: %s\n", errbuf);
    return -1;
}
```

- 编译过滤规则

```
if (pcap_compile(dev_handle_pcap, &bpf_filter, bpf_filter_string, 0, net_ip) == -1) {
    fprintf(stderr, "pcap_compile: %s\n", bpf_filter_string);
    return -1;
}
```

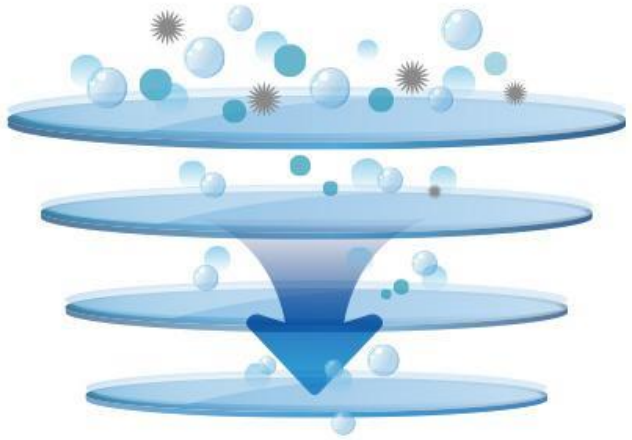
知识讲解

libpcap捕获分析数据包的关键程序

- 设置过滤规则

```
if (pcap_setfilter(dev_handle_pcap, &bpf_filter) == -1) {
    fprintf(stderr, "pcap_setfilter: %s\n", bpf_filter_string);
    return -1;
}
```

知识讲解



libpcap捕获分析数据包的关键程序

- 过滤表达式
  - 过滤表达式相当于一种微型语言，亦有其特定的语法：
    - 过滤表达式支持逻辑运算，如and、or、not等，也可以通过小括号人为规定优先级
    - 基于协议的过滤可以使用诸如ip、arp、rarp、tcp和udp等协议限定符
    - 基于MAC地址的过滤使用ether限定符：
      - 过滤源MAC地址：ether src 00:E0:4C:E0:38:88
      - 过滤目的MAC地址：ether dst 00:E0:4C:E0:38:88
      - 同时过滤源和目的MAC地址：ether host 00:E0:4C:E0:38:88
    - 基于IP地址的过滤使用host限定符：
      - 过滤源IP地址：src host 192.168.1.27
      - 过滤目的IP地址：dst host 192.168.1.27
      - 同时过滤源和目的IP地址：host 192.168.1.27
    - 基于端口的过滤使用port限定符：
      - 过滤特定端口：port 80

知识讲解

libpcap捕获分析数据包的关键程序

- 过滤表达式
  - 例如：捕获ARP和ICMP包
    - arp or icmp
  - 例如：捕获以192.168.1.27为源或目的地址的80端口TCP包
    - (ip and tcp) and (host 192.168.1.27) and (port 80)
  - 例如：捕获在主机192.168.1.27和192.168.1.28之间传递的所有UDP包
    - (ip and udp) and ((src host 192.168.1.27 and dst host 192.168.1.28) or (src host 192.168.1.28 and dst host 192.168.1.27))
  - 例如：捕获从MAC地址00:E0:4C:E0:38:88发往MAC地址00:50:56:C0:D2:F6的所有ARP包
    - arp and (ether src 00:E0:4C:E0:38:88 and ether dst 00:50:56:C0:D2:F6)

知识讲解

libpcap捕获分析数据包的关键程序

- 捕获分析数据包

```
pcap_loop(dev_handle_pcap, -1, packet_handler, NULL);
```

  - pcap_loop函数每捕获到一个数据包，就会将其作为参数传递给程序设计者自定义packet_handler函数，该函数负责对数据包的具体处理，其原型如下：

```
void packet_handler(u_char* argument, const struct pcap_pkthdr* packet_header,
    const u_char* packet_content);
```

  - 其中，packet_header参数为libpcap在捕获数据包时附加的辅助信息，如时间戳、包长度等，packet_content参数为捕获到的数据包内容

- 关闭设备

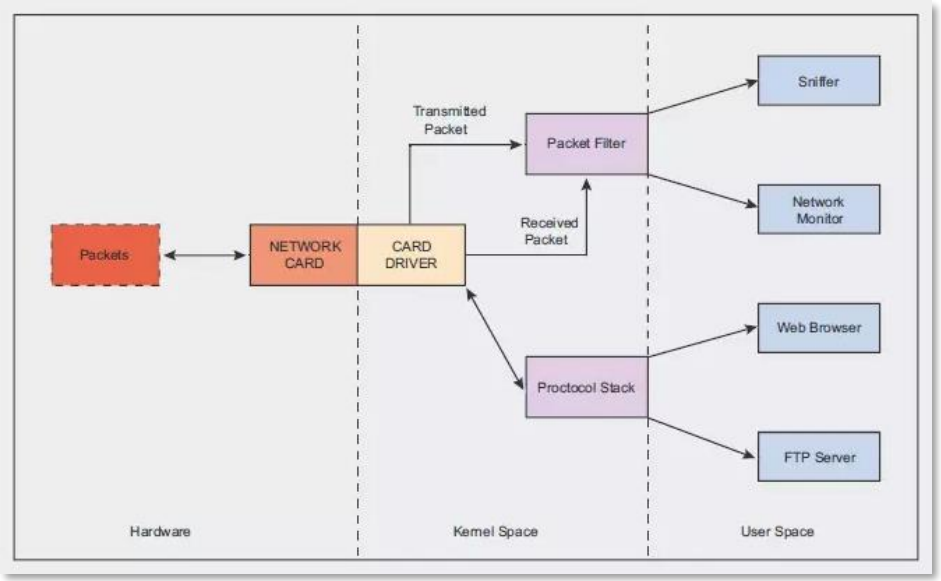
```
pcap_close(dev_handle_pcap);
```

知识讲解

libpcap注意事项

- 使用libpcap库需要添加-lpcap链接选项
- libpcap只能捕获数据包而不能发送数据包。Linux下的另一个开发工具libnet可以填充和发送数据包。libpcap的Windows版本winpcap既可捕获亦可发送数据包

知识讲解



通过tcpdump命令捕获数据包

tcpdump简介

- tcpdump是一款功能十分强大的网络数据包捕获分析工具：
  - 支持针对协议栈层次、协议种类、主机、网络和端口的数据包过滤
  - 支持基于正则表达式形式的过滤策略描述
  - 主要用于对网络的分析、维护、统计和检测，如定位网络瓶颈、跟踪流量变化等
  - 在源代码公开的同时提供应用编程接口(API)，支持二次开发
  - 大多数Linux系统都缺省提供对tcpdump的集成，无需单独安装
  - tcpdump需要网卡工作于混杂模式，必须获得root权限才能运行

知识讲解

tcpdump命令

- tcpdump命令的语法如下：

```
tcpdump [-aAbdDefhHIJKlLnNOpqStuUvxX#][-B size][-c count]
        [-C file_size][-E algo:secret][-F file][-G seconds]
        [-i interface][-j tstamptype][-m module][-M secret][--number]
        [-Q in|out|inout][-r file][-s snaplen][--time-stamp-precision precision]
        [--immediate-mode][-T type][--version][-V file][-w file][-W filecount]
        [-y datalinktype][-z postrotate-command][-Z user][expression]
```

知识讲解

tcpdump选项

- tcpdump命令的选项众多，其中部分常用选项如下表所示：

选项	含义
-a	将网络地址和广播地址转换成名字
-dd	将匹配的数据包以C语言代码的格式输出
-ddd	将匹配的数据包以十进制形式输出
-e	输出数据链路层包头
-f	将外部互联网地址以数字形式输出
-l	对标准输出进行缓冲，可在捕获的同时查看数据包
-n	不把网络地址转换为主机名
-N	不输出主机名中的域名后缀，如tcpdump.org只输出tcpdump
-O	不运行数据包匹配模板的优化器
-p	不将网卡设置为混杂模式
-q	快速输出，只输出较少的协议信息
-S	将TCP序号以绝对值而非相对值的形式输出
-t	不在输出的每一行打印时间戳

知识讲解

tcpdump选项

- tcpdump命令的选项众多，其中部分常用选项如下表所示：

-u	输出未解码的NFS句柄
-v	输出详细信息，如IP包头中的服务类型、生存时间(TTL)等
-vv	输出更详细的信息
-vvv	输出最详细的信息
-c count	指定数据包监听上限，达到此数量即退出tcpdump
-C file_size	指定数据包文件上限，达到此字节数即不再写文件
-E algo:secret	用指定的算法解密IPSec ESP数据包
-F file	从指定的文件中读取用于过滤数据包的正则表达式，忽略命令行中的正则表达式
-i interface	指定监听的网卡
-m module	打开指定的SMI MIB组件(需要系统支持libsmi)
-r file	从指定的文件中读取数据包
-s snaplen	从每个数据包中读取最开始的snaplen字节，而非默认的68字节
-T type	将捕获的数据包解析为指定的类型，如rpc、rtp、rtcp、vat、wb等
-w file	将捕获的数据包写入指定的文件

知识讲解

tcpdump的正则表达式

- tcpdump通过正则表达式过滤数据包。如果没有指定正则表达式，则捕获全部数据包，否则只捕获那些满足正则表达式规则的数据包
- tcpdump的正则表达式包含三种主要关键字：
  - 类型关键字：host、net、port，缺省为host。如：
    - tcpdump host 192.168.1.27
    - 捕获IP地址为192.168.1.27的主机收发的所有数据包
  - 方向关键字：src、dst、src or dst、src and dst。缺省为src or dst。如：
    - tcpdump dst net 192.168.1.1
    - 捕获目标网络地址为192.168.1.1的所有数据包
  - 协议关键字：ether、fddi、tr、ip、ip6、arp、rarp、decnet、tcp、udp。如：
    - tcpdump udp
    - 捕获所有UDP协议的数据包

知识讲解

tcpdump的正则表达式

- tcpdump的正则表达式还包含其它一些关键字：
  - gateway、broadcast
  - less、greater
  - and(&&)、or(||)、not(!)
- 通过这些关键字的灵活组合，可以表达各种复杂的过滤条件

知识讲解

tcpdump简单示例

- 捕获ARP和UDP包
  - tcpdump arp or udp
- 捕获除主机alice之外的所有IP包
  - tcpdump ip host not alice
- 捕获主机192.168.1.27收发的所有TELNET包
  - tcpdump tcp and host 192.168.1.27 and port 23
- 捕获以192.168.1.27为源地址，以192.168.1.28为目的地址的80端口TCP包
  - tcpdump tcp and src host 192.168.1.27 and dst host 192.168.1.28 and port 80
- 捕获在主机192.168.1.27和192.168.1.28之间传递的所有TCP包
  - tcpdump \(ip and tcp\) and \(\(src host 192.168.1.27 and dst host 192.168.1.28\) or \(src host 192.168.1.28 and dst host 192.168.1.27\)\)

知识讲解

tcpdump注意事项

- tcpdump的正则表达式中如果包含括号，前面必须加上反斜杠“\”

知识讲解

总结和答疑