黑客攻防

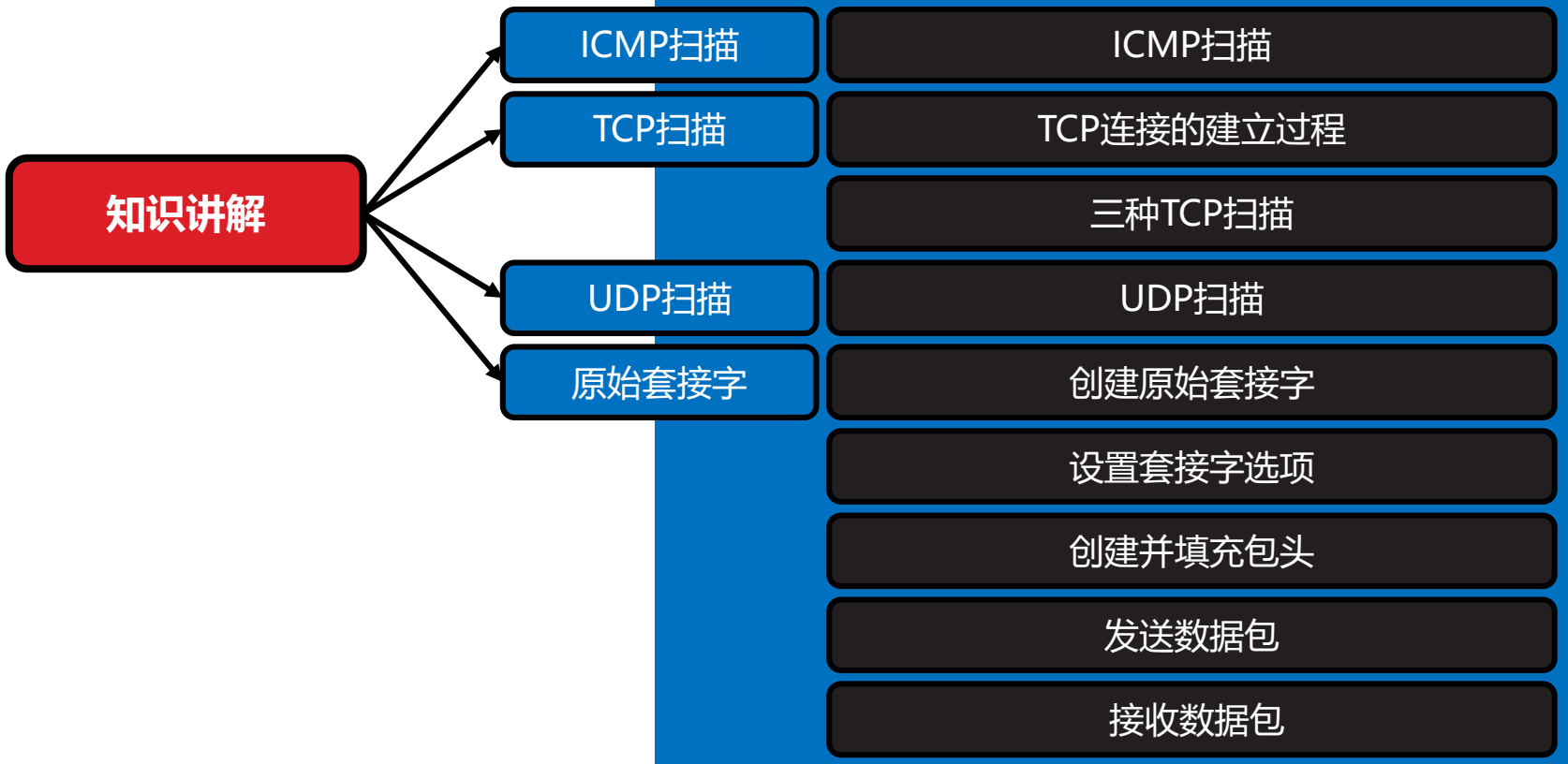
端口扫描

Unit08

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

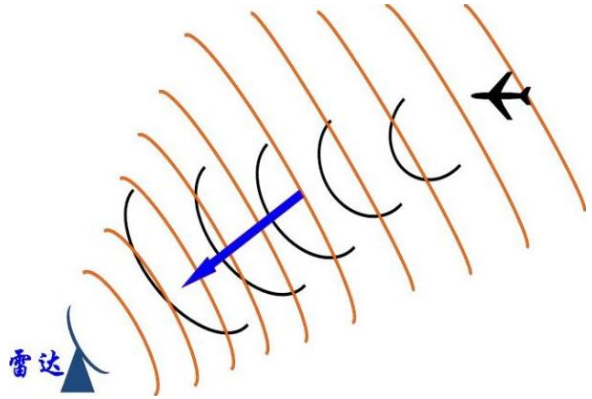

知识讲解
ICMP扫描
TCP扫描
UDP扫描
原始套接字
ICMP扫描
TCP连接的建立过程
三种TCP扫描
UDP扫描
创建原始套接字
设置套接字选项
创建并填充包头
发送数据包
接收数据包

ICMP扫描

ICMP扫描

- ping程序只是简单探测某个IP地址所对应的主机是否存在，其原理十分简单：
  - 扫描发起主机向目标主机发送一个要求回射(type=8，即ICMP_ECHO)的ICMP数据包
  - 目标主机向扫描发起主机返回一个应答回射(type=0，即ICMP_ECHOREPLY)的ICMP数据包
  - 扫描发起主机根据是否收到回射应答判断目标主机是否存在

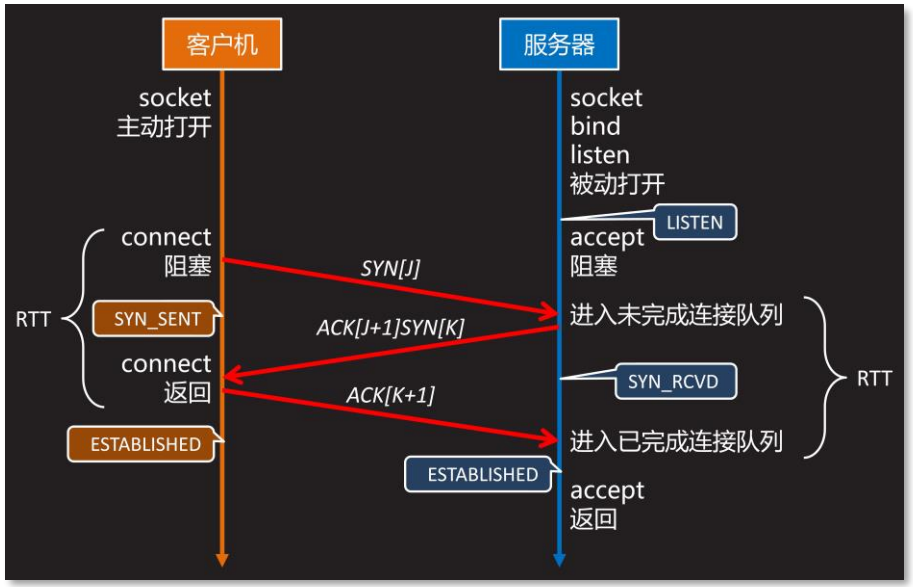
知识讲解



TCP扫描

TCP连接的建立过程

- 建立TCP连接的三路握手过程如下：
  - 客户端向服务器发送一个SYN=1，ACK=0的TCP数据包，希望与之建立连接
  - 如果服务器接受客户端的连接请求，就返回一个SYN=1，ACK=1的TCP数据包，同意建立连接，并要求对方确认
  - 客户端再次向服务器发送一个SYN=0，ACK=1的TCP数据包，确认建立连接



知识讲解

三种TCP扫描

- CONNECT扫描
  - 扫描发起主机通过套接字应用编程接口(Socket API)的connect函数尝试连接目标主机的被扫描端口，如果connect函数返回成功，即表明扫描发起主机和目标主机之间至少经历了一轮包含三路握手在内的TCP连接建立过程，据此可以断定目标主机的被扫描端口是开放的，否则该端口就是关闭的
  - 由于TCP协议是可靠传输协议，connect函数不会仅在一次尝试建立连接失败后即返回失败，而是要经历多次失败的连接尝试后才会彻底放弃，这中间需要漫长的等待时间
  - 失败的connect函数会在系统中产生大量连续失败日志，容易被系统管理员发现

知识讲解

三种TCP扫描

- SYN扫描
  - 扫描发起主机向目标主机的被扫描端口发送一个SYN数据包：
    - 收到一个ACK数据包，表示该端口开放
    - 收到一个RST数据包，表示该端口关闭
    - 未收到任何数据包，表示目标主机存在，但所发送SYN数据包可能被防火墙等安全设备屏蔽
  - SYN扫描不需要完成TCP连接的三路握手过程，因此它又被称为半开放扫描
  - SYN扫描的最大优点就是速度快。在互联网环境下，如果不考虑防火墙等安全设备的影响，每秒钟可以扫描数千个端口。但由于其扫描行为明显，很容易被入侵检测系统发现，也极易被防火墙等安全设备屏蔽，同时构造原始TCP数据包通常需要较高的系统权限，如果是在Linux系统上，就必须拥有root权限

知识讲解

三种TCP扫描

- FIN扫描
  - 扫描发起主机向目标主机的被扫描端口发送一个FIN数据包：
    - 确定存在的目标主机没有任何响应，表示该端口开放且处于监听状态
    - 确定存在的目标主机返回一个RST数据包，表示该端口关闭
  - FIN扫描具有很好的隐蔽性，不会留下日志，但它的局限性很大，仅对UNIX/Linux系统的网络协议栈有效。运行Windows系统的目标主机，无论被扫描端口开放与否，都会返回RST数据包，因此无法对该端口当前所处状态做出准确的判断

知识讲解

UDP扫描

UDP扫描

- 向一个处于关闭状态的UDP端口发送数据，目标主机会返回端口不可达(Port Unreachable)错误
- 扫描发起主机向目标主机的被扫描端口发送0字节的UDP数据包：
  - 收到ICMP端口不可达(type=3, code=3)，表示该端口关闭
  - 长时间没有任何响应，表示该端口开放
- 大部分系统都限制了ICMP差错报文的产生速度，因此大范围的UDP端口扫描可能会相当耗时
- UDP和ICMP都是不可靠协议，数据包丢失也可能导致收不到响应，因此扫描程序有必要对同一端口进行多次尝试，每次都收不到响应才有理由相信该端口是开放的

知识讲解

通过原始套接字发送数据包

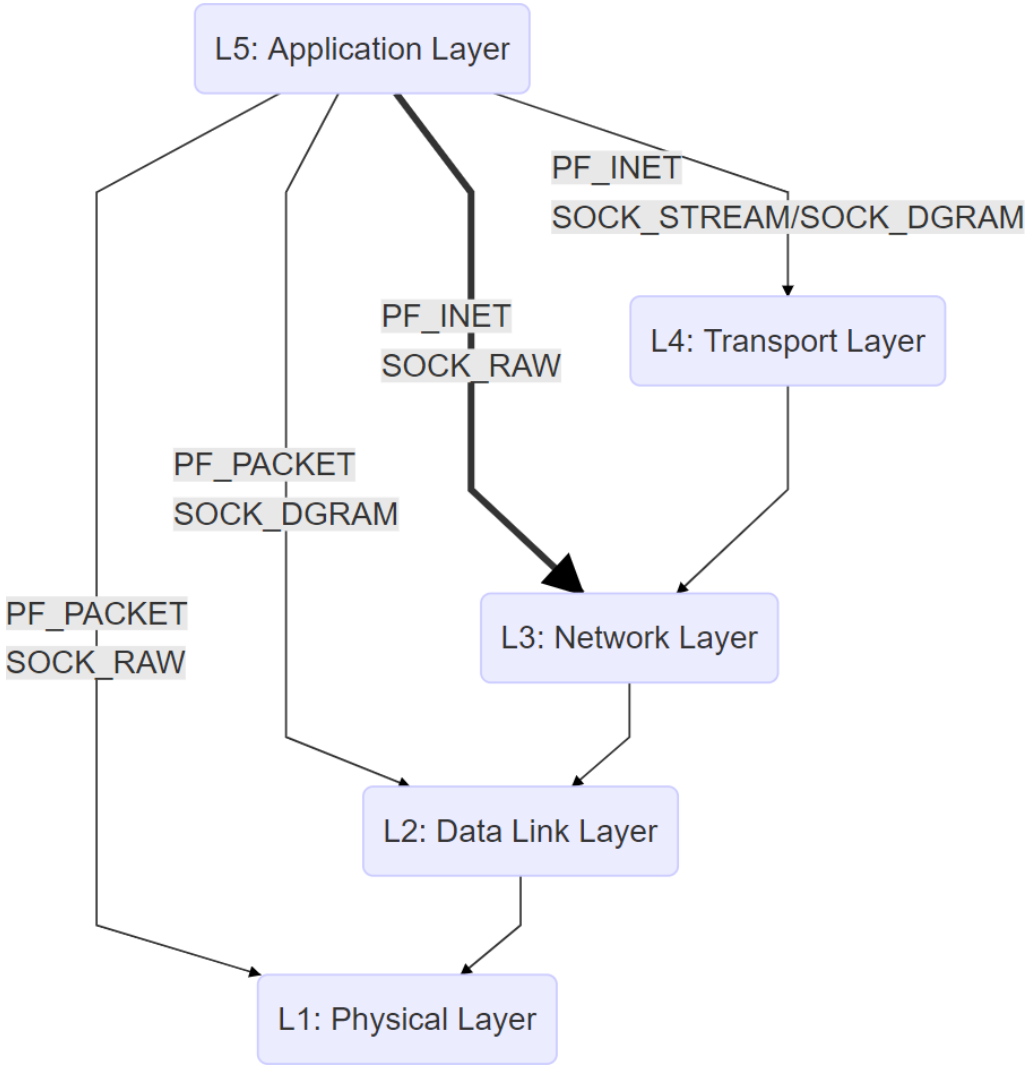
通过原始套接字发送数据包

- 在以上所列举的各种端口扫描方法中，只有TCP CONNECT扫描可以直接通过connect函数完成数据包的构造、发送和接收，其它端口扫描方法都必须通过原始套接字手动构造数据包，并借助专门的系统调用函数实现对所构造数据包的发送和接收

知识讲解

创建原始套接字

- 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

创建原始套接字

- 创建基于IP协议的原始套接字

```
int sockfd = socket(PF_INET, SOCK_RAW, IPPROTO_IP);
```

- 创建基于ICMP协议的原始套接字

```
int sockfd = socket(PF_INET, SOCK_RAW, IPPROTO_ICMP);
```

- 创建基于TCP协议的原始套接字

```
int sockfd = socket(PF_INET, SOCK_RAW, IPPROTO_TCP);
```

- 创建基于UDP协议的原始套接字

```
int sockfd = socket(PF_INET, SOCK_RAW, IPPROTO_UDP);
```

知识讲解

设置套接字选项

- 设置套接字选项的函数：

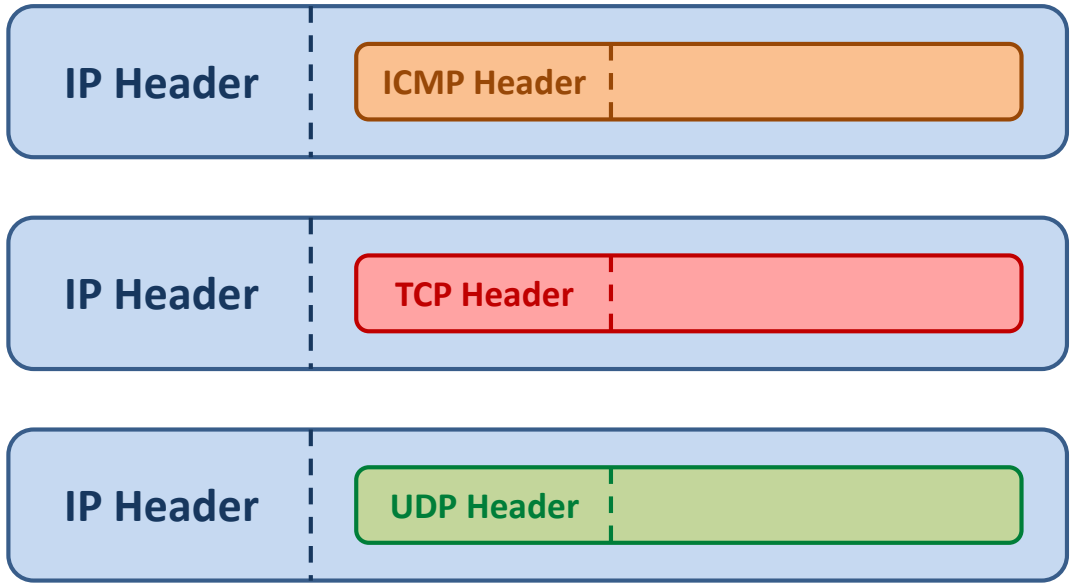
```
int setsockopt(
    int             sockfd,   // 套接字文件描述符
    int             level,    // 选项层次
                              // SOL_SOCKET   : 通用选项
                              // IPPROTO_IP   : IP选项
                              // IPPROTO_TCP  : TCP选项
    int             optname,  // 选项名称
    const void *    optval,   // 选项值地址
    socklen_t   *   optlen);  // 选项值长度
```

知识讲解

创建并填充包头

- Linux系统已经预定义了常用网络协议的包头结构体：
  - IP包头：iphdr
  - ICMP包头：icmphdr
  - TCP包头：tcphdr
  - UDP包头：udphdr

知识讲解



创建并填充包头

知识讲解

- IP包头

```
#include <netinet/ip.h>

struct iphdr {
    unsigned int version:4; // 版本(4位)
    unsigned int ihl:4;     // 包头长度(4位)
    uint8_t      tos;       // 服务类型
    uint16_t     tot_len;   // 包长度
    uint16_t     id;        // 标识符
    uint16_t     frag_off;  // 标志(3位)和偏移(13位)
    uint8_t      ttl;       // 生存时间
    uint8_t      protocol;  // 上层协议
    uint16_t     check;     // 校验和
    uint32_t     saddr;     // 源IP地址
    uint32_t     daddr;     // 目的IP地址
};
```

- ICMP包头

```
#include <netinet/ip_icmp.h>

struct icmphdr {
    uint8_t  type;                      // 类型
    uint8_t  code;                      // 代码
    uint16_t checksum;                  // 校验和
    union {
        struct {
            uint16_t id;                // 标识符
            uint16_t sequence;          // 序号
        }   echo;                       // 回射数据包
        uint32_t gateway;               // 网关地址
        struct {
            uint16_t __glibc_reserved; // 保留未用
            uint16_t mtu;               // 最大传输单元
        }   frag;
    }   un;
};
```

创建并填充包头

知识讲解

- TCP包头

```
#include <netinet/tcp.h>

struct tcphdr {
    uint16_t source;  // 源端口
    uint16_t dest;    // 目的端口
    uint32_t seq;     // 序号
    uint32_t ack_seq; // 确认序号
    uint16_t doff:4;  // 包头长度(4位)
    uint16_t res1:4;  // 保留未用(4位)
    uint16_t res2:2;  // 保留未用(2位)
    uint16_t urg:1;   // URG位 \
    uint16_t ack:1;   // ACK位 |
    uint16_t psh:1;   // PSH位 | 标志(6位)
    uint16_t rst:1;   // RST位 |
    uint16_t syn:1;   // SYN位 |
    uint16_t fin:1;   // FIN位 /
    uint16_t window;  // 窗口大小
    uint16_t check;   // 校验和
    uint16_t urg_ptr; // 紧急指针
};
```

- UDP包头

```
#include <netinet/udp.h>

struct udphdr {
    uint16_t source; // 源端口
    uint16_t dest;   // 目的端口
    uint16_t len;    // 包长度
    uint16_t check;  // 校验和
};
```

发送数据包

- 在填充完数据包之后，即可通过sendto函数发送数据包：

知识讲解

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

ssize_t sendto(
    int                     sockfd,     // 套接字文件描述符
    const void              * buf,      // 发送数据包缓冲区
    size_t                  len,        // 发送数据包字节数
    int                     flags,      // 发送标志
    const struct sockaddr * dest_addr, // 目的地址结构
    socklen_t               addrlen);  // 目的地址结构字节数
```

发送数据包

- 在填充完数据包之后，即可通过sendto函数发送数据包：
  - 对于面向连接的SOCK_STREAM类型套接字，最后两个参数将被忽略
  - sendto函数成功返回实际发送的字节数，失败返回-1，并设置errno变量
  - 一般情况下，buf所指向的发送数据包缓冲区中并不需要包含IP包头，网络层会负责IP报文的封装。如果希望自己组织IP包头的内容，可以通过setsockopt函数设置相应的套接字选项：

```
int on = 1;
setsockopt(sockfd, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on));
```

    - 通过带有IP_HDRINCL选项的套接字发送的数据包，由IP包头、TCP或UDP包头以及数据载荷三部分组成

知识讲解

接收数据包

- 在完成数据包的发送后，可以通过recvfrom函数接收远程主机的响应数据包：

知识讲解

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

ssize_t recvfrom(
    int                 sockfd,   // 套接字文件描述符
    void              * buf,      // 接收数据包缓冲区
    size_t              len,      // 期望接收的字节数
    int                 flags,    // 接收标志
    struct sockaddr   * src_addr, // 源地址结构
    socklen_t         * addrlen); // 源地址结构字节数
```

接收数据包

- 在完成数据包的发送后，可以通过recvfrom函数接收远程主机的响应数据包：
  - recvfrom函数成功返回实际接收的字节数，失败返回-1，并设置errno变量。通过基于TCP协议的流式套接字(SOCK_STREAM)接收数据包，该函数可能返回0，这通常意味着远程主机关闭了连接
  - 一般在接收数据包的过程中，recvfrom函数会一直处于阻塞状态，直到收到一个数据包之后，该函数才会返回。如果希望recvfrom函数以非阻塞方式接收数据包，可以通过fcntl函数将套接字设置为非阻塞模式：

```
fcntl(sockfd, F_SETFL, O_NONBLOCK);
```

知识讲解

接收数据包

- 通过阻塞或非阻塞套接字调用sendto函数的返回值如下表所示：

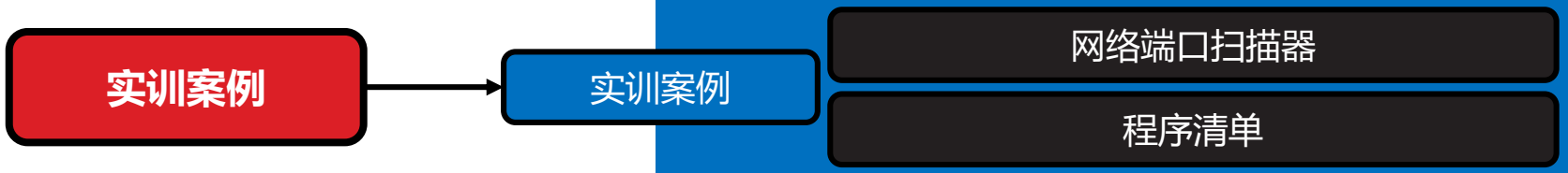
sendto	slen = sendto(sockfd, buf, len, 0, (struct sockaddr*)&addr, addrlen)			
	slen == len	0 < slen && slen < len	slen == 0	slen == -1
阻塞套接字	发送缓冲区充足	——		出错
非阻塞套接字		发送缓冲区不足	发送缓冲区充满	

- 通过阻塞或非阻塞套接字调用recvfrom函数的返回值如下表所示：

recvfrom	rlen = recvfrom(sockfd, buf, len, 0, (struct sockaddr*)&addr, &addrlen)			
	rlen == len	0 < rlen && rlen < len	rlen == 0	rlen == -1
阻塞套接字	可接收数据足够	可接收数据不足	TCP连接断开	出错
非阻塞套接字			没有可接收数据	

知识讲解

实训案例



实训案例

网络端口扫描器

- 在Linux环境中编写一个网络端口扫描器
  - 基于套接字实现ping程序，探测用户输入的主机IP地址是否可达
  - 基于套接字实现TCP CONNECT扫描、TCP SYN扫描、TCP FIN扫描和UDP扫描，针对用户输入的主机IP地址和端口号范围进行扫描，打印每个端口号的扫描结果

代码实践

程序清单

- 声明Scanner类
  - scanner.h
- 实现Scanner类
  - scanner.cpp
- 声明TCPScanner类
  - tcpscanner.h
- 实现TCPScanner类
  - tcpscanner.cpp
- 声明SYNScanner类
  - synscanner.h
- 实现SYNScanner类
  - synscanner.cpp
- 声明FINScanner类
  - finscanner.h
- 实现FINScanner类
  - finscanner.cpp

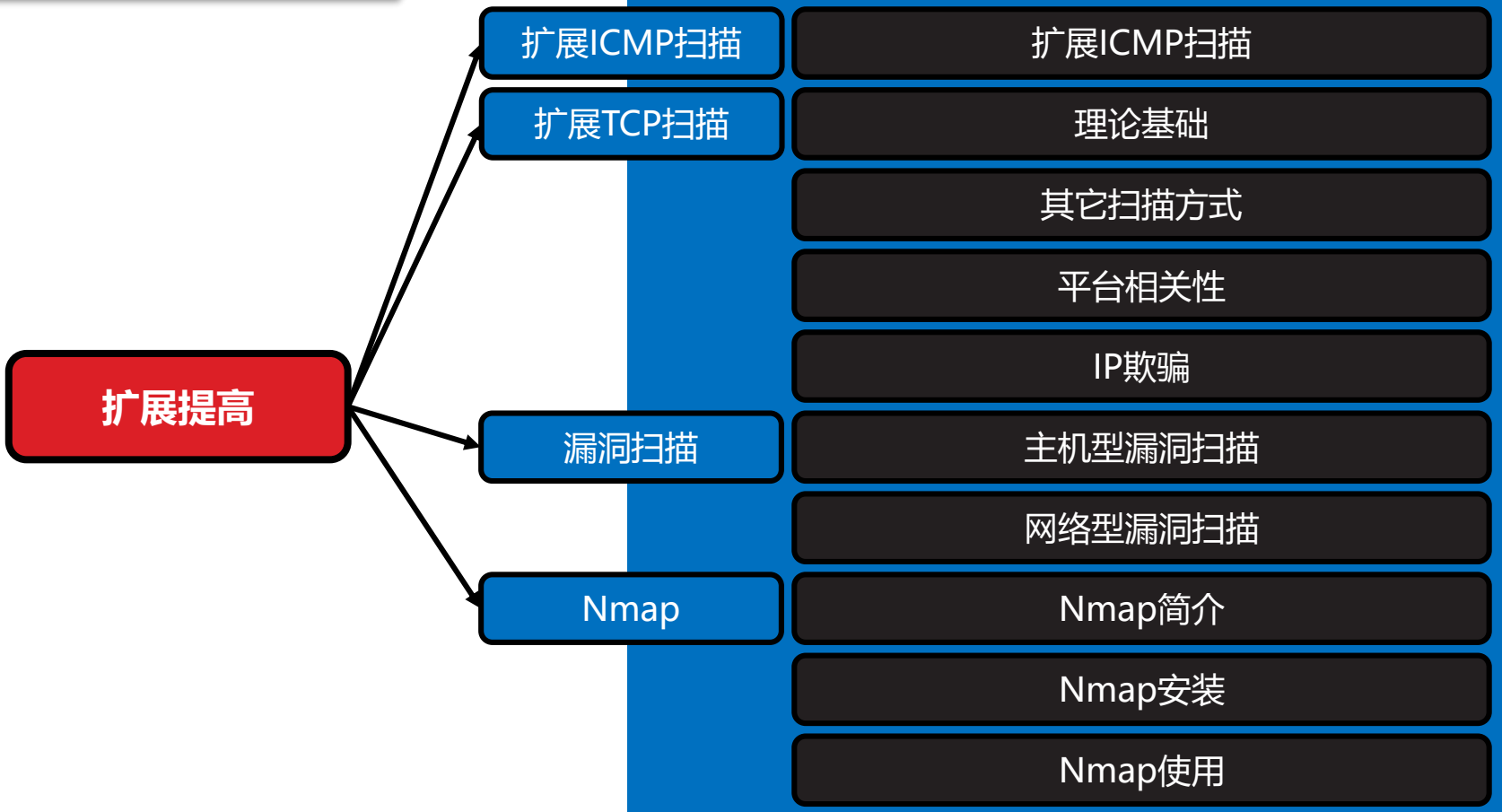
代码实践

程序清单

- 声明UDPScanner类
  - udpscanner.h
- 实现UDPScanner类
  - udpscanner.cpp
- 测试Scanner类及其子类
  - scanner_test.cpp
- 测试Scanner类及其子类构建脚本
  - scanner_test.mak

代码实践

扩展提高


扩展提高
扩展ICMP扩展
扩展TCP扩展
漏洞扫描
Nmap
扩展ICMP扩展
理论基础
其它扫描方式
平台相关性
IP欺骗
主机型漏洞扫描
网络型漏洞扫描
Nmap简介
Nmap安装
Nmap使用

扩展ICMP扫描

扩展ICMP扫描

- ICMP扫描易被防火墙过滤，扩展ICMP扫描可以绕开防火墙探测其后的主机
  - 根据TCP/IP协议，如果在通信过程中出现错误，接收端将向发送端响应一个ICMP错误报文以告知对方具体错误信息，这些错误报文一般不会被防火墙拦截：
    - 发送一个只有IP头的IP数据包，对方返回“Destination Unreachable”
    - 发送一个包含错误(如IP头长度错误)的IP数据包，对方返回“Parameter Problem”
    - 没有发送足够的数据包分片，对方返回“分片组装超时”
    - 发送一个IP数据包，但上层协议是错的，对方返回“Destination Unreachable”
- 扩展ICMP扫描还有两种其它功能：
  - 探测防火墙是否存在：将数据包IP头上层协议字段设置为一个无人使用的大值，发送给确定存在的目标主机，若收不到响应，则说明该主机处于防火墙保护之中
  - 探测目标主机的协议：IP包头上层协议字段由八个二进制位组成，因此最多只能表示256种协议。穷举这256种可能，便可探测出目标主机当前使用的协议

知识讲解

扩展TCP扫描

理论基础

- 前面提到的TCP FIN扫描属于秘密扫描的一种，即在扫描过程中无需建立TCP连接，因此不会被目标主机系统察觉，也不会留下任何日志记录
- 当一个包头中带有不同标志位的TCP报文，到达一个处于监听或关闭状态的端口时，网络协议栈将以不同的标志位做出响应或不响应。如下表所示：

STATE	URG	ACK	PSH	RST	SYN	FIN	NULL
LISTEN	——	RST	——	——	ACK+SYN	——	——
CLOSED	RST	——	RST	——	ACK+RST	ACK+RST	RST

- 端口扫描程序即根据目标主机是否响应，以及响应包头中的标志位，判断被扫描端口的当前状态

知识讲解

其它扫描方式

- ACK扫描
  - 向目标主机的被扫描端口，发送一个仅包含ACK标志位的TCP包头：
    - 响应包头带RST标志位：该端口处于监听状态
    - 没有收到任何响应：该端口处于关闭状态
- NULL扫描
  - 向目标主机的被扫描端口，发送一个不包含任何标志位的TCP包头：
    - 响应包头带RST标志位：该端口处于关闭状态
    - 没有收到任何响应：该端口处于监听状态
- URG+PSH+FIN扫描
  - 向目标主机的被扫描端口，发送一个包含URG、PSH和FIN标志位的TCP包头：
    - 响应包头带RST标志位：该端口处于关闭状态
    - 没有收到任何响应：该端口处于监听状态

知识讲解

平台相关性

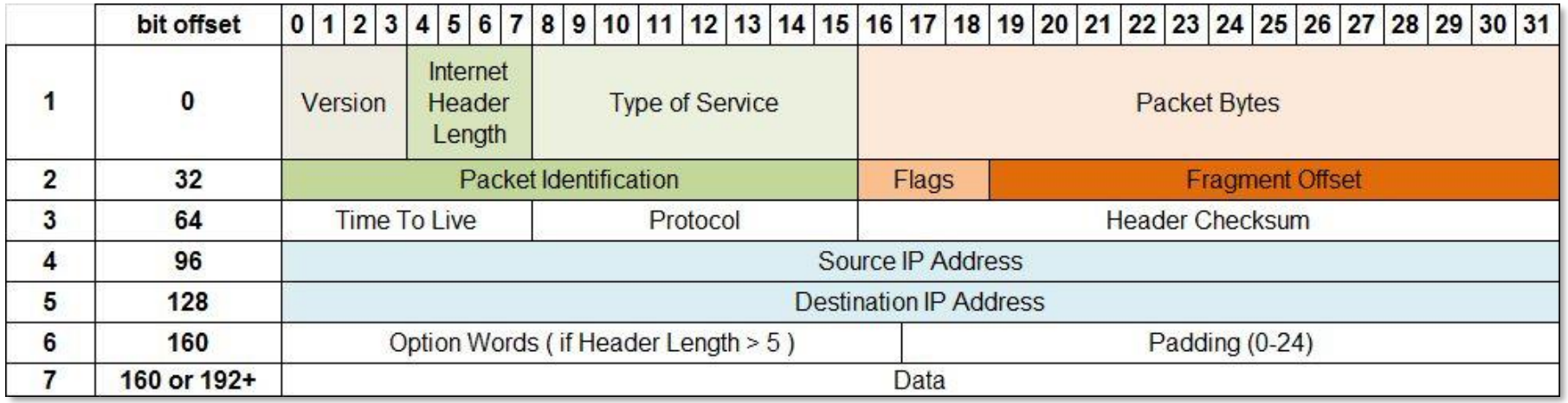
- 秘密扫描虽然种类繁多，但大都依赖于系统平台网络协议栈的实现细节，因此同一种扫描方法未必对所有的操作系统都有效。只有选择那些与特定目标平台相适应的扫描方法，才能达到预期的效果

知识讲解

IP欺骗

- 为了规避目标主机对特定IP地址集的防护性过滤，扫描程序可以将报文IP包头中的源IP地址字段，设置为某个被目标主机信任的IP地址，冒充该地址所表示的主机，达到隐藏自身，欺骗目标的目的

知识讲解



	bit offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	0	Version				Internet Header Length				Type of Service								Packet Bytes															
2	32	Packet Identification																Flags			Fragment Offset											
3	64	Time To Live								Protocol								Header Checksum															
4	96	Source IP Address																															
5	128	Destination IP Address																															
6	160	Option Words (if Header Length > 5)																Padding (0-24)															
7	160 or 192+	Data																															

漏洞扫描

漏洞扫描

- 漏洞扫描最初是一种黑客攻击手段。黑客通过自己编写或利用他人开发的软件工具，检测被扫描主机系统中是否存在某些已知的漏洞，然后专门针对这些漏洞发起致命的攻击
- 随着扫描技术的发展和漏洞资料库的日趋完善，逐渐出现了专业化的安全评估工具——漏洞扫描器。借助漏洞扫描器，网络安全人员或设备制造商可以提前发现系统中存在的潜在风险，及时修补，防患于未然

知识讲解

主机型漏洞扫描

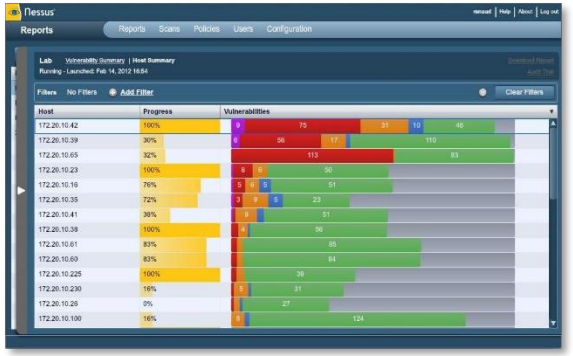
- 主机型漏洞扫描亦称被动扫描，即通过一系列非破坏性的方法对系统的用户权限、文件属性、服务配置、应用程序乃至安全补丁等进行扫描检测，最后向用户提供检测报告
- 主机型漏洞扫描通常需要较高的用户权限，因此它能够准确定位系统中存在的各种安全隐患，所能发现的问题比较全面，检测到的漏洞也比较彻底

知识讲解

网络型漏洞扫描

- 网络型漏洞扫描通过定制化的脚本，模拟黑客对被扫描系统发起攻击，并对结果进行分析，检查其是否与漏洞数据库中的已知规则相匹配，若匹配则说明存在安全漏洞
- 目前大部分漏洞扫描产品都是基于网络型的，同时它们也借鉴了主机型的很多优点，既可以对网络上的服务器系统进行远程漏洞检测，也可以对本地主机系统进行更深层次的安全检测

知识讲解



网络型漏洞扫描

- 网络漏洞扫描器的工作过程一般是这样的：
  - 首先探测目标网络中处于活动状态的服务器主机
  - 对活动主机进行端口扫描，找到其开放的端口，同时利用协议指纹技术，识别出它们的操作系统
  - 对所发现的开放端口进行网络服务类型识别，确定其提供的网络服务
  - 根据操作系统和网络服务查询漏洞数据库，获取相应的漏洞特征记录
  - 根据漏洞特征记录所描述的规则组织特定的探测报文，发给探测目标
  - 接收对方的响应报文，检查其中是否含有漏洞特征记录所描述的响应特征，判断该漏洞是否存在

知识讲解

基于Nmap的网络探测

Nmap简介

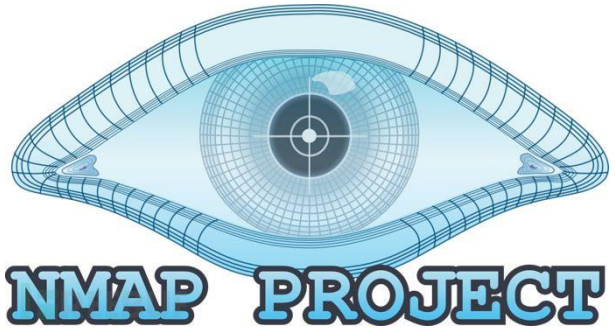
- 网络映射器(Network mapper, Nmap)是一款开源网络探测和安全审计实用工具。系统管理人员可以利用该工具：
  - 快速扫描大型网络，发现其中运行的主机
  - 探测主机上运行的操作系统、提供的网络服务、报文过滤器和防火墙类型
- Nmap支持多种类型的扫描：
  - TCP CONNECT扫描
  - TCP SYN扫描
  - TCP FIN扫描
  - TCP ACK扫描
  - TCP NULL扫描
  - TCP Maimon扫描
  - TCP Window扫描
  - TCP XmasTree扫描
  - UDP扫描
  - IP扫描
  - FTP弹跳扫描

知识讲解

Nmap安装

- 在Ubuntu上安装Nmap非常简单：
  - sudo apt install nmap

知识讲解




```
31337
# nmap -A -T4 scanme.nmap.org d0ze

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2006-03-20 15:53 PST
Interesting ports on scanme.nmap.org (205.217.153.62):
(The 1667 ports scanned but not shown below are in state: filtered)
PORT    STATE  SERVICE VERSION
22/tcp  open   ssh     OpenSSH 3.9p1 (protocol 1.99)
25/tcp  opn    smtp    Postfix smtpd
53/tcp  open   domain  ISC Bind 9.2.1
70/tcp  closed gopher
80/tcp  open   http    Apache httpd 2.0.52 ((Fedora))
113/tcp closed auth
Device type: general purpose
Running: Linux 2.6.X
OS details: Linux 2.6.0 - 2.6.11
Uptime 26.177 days (since Wed Feb 22 11:39:16 2006)

Interesting ports on d0ze.internal (192.168.12.3):
(The 1664 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE  VERSION
21/tcp   open  ftp          Serv-U ftpd 4.0
25/tcp   open  smtp         IMail NT-ESMTP 7.15 2015-2
80/tcp   open  http         Microsoft IIS webserver 5.0
110/tcp  open  pop3         IMail pop3d 7.15 931-1
135/tcp  open  mstask       Microsoft mstask (task server - c:\winnt\system32\
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds Microsoft Windows XP microsoft-ds
1025/tcp open  msrpc        Microsoft Windows RPC
5800/tcp open  vnc-http     Ultr@VNC (Resolution 1024x800; VNC TCP port: 5900)
MAC Address: 00:A0:CC:51:72:7E (Lite-on Communications)
Device type: general purpose
Running: Microsoft Windows NT/2K/XP
OS details: Microsoft Windows 2000 Professional
Service Info: OS: Windows

Nmap finished: 2 IP addresses (2 hosts up) scanned in 42.291 seconds
flog/home/fyodor/nmap-misc/Screenshots/042006#
```

Nmap使用

- Nmap的命令语法如下：

```
nmap [Scan Type(s)][Options]{target specification}
```

知识讲解

– 扫描类型(Scan Type(s))

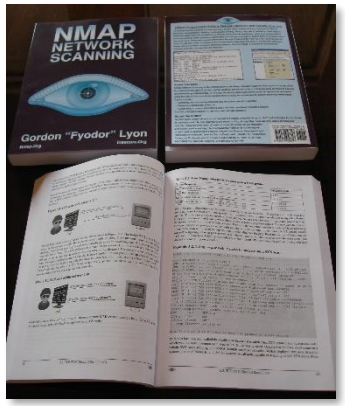
Scan Type(s)	扫描类型
-sT	TCP CONNECT扫描
-sS	TCP SYN扫描
-sF	TCP FIN扫描
-sA	TCP ACK扫描
-sN	TCP NULL扫描
-sM	TCP Maimon扫描
-sW	TCP Window扫描
-sX	TCP XmasTree扫描
-sU	UDP扫描
-sO	IP扫描
-b	FTP弹跳扫描

Nmap使用

- Nmap的命令语法如下：

```
nmap [Scan Type(s)][Options]{target specification}
```

  - 功能选项(Options)
    - 不同的扫描类型会有不同的功能选项，它们可以组合使用，但并非所有组合都是合理的。对不支持的功能组合，Nmap会向用户发出警告
    - Namp的功能选项非常多，详情请参见：https://nmap.org/book/man.html



知识讲解

Nmap使用

- Nmap的命令语法如下：

```
nmap [Scan Type(s)][Options]{target specification}
```

  - 目标说明(target specification)
    - 目标说明用于指定扫描的对象，一个IP地址、一个域名或一个CIDR风格的地址范围
    - 例如：对本地主机执行TCP SYN扫描

```
$ sudo nmap -sS 127.0.0.1

Starting Nmap 7.60 ( https://nmap.org ) at 2019-04-04 15:01 CST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000012s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE
631/tcp  open  ipp
8000/tcp open  http-alt

Nmap done: 1 IP address (1 host up) scanned in 1.85 seconds
```

知识讲解

Nmap使用

- Nmap的命令语法如下：

```
nmap [Scan Type(s)][Options]{target specification}
```

  - 目标说明(target specification)
    - 例如：对IP地址介于192.168.1.0到192.168.1.255之间的主机执行TCP SYN扫描

```
$ sudo nmap -sS 192.168.1.0/24
```

其中，24表示被扫描主机IP地址的前24位与192.168.1.0相同

知识讲解

Nmap使用

- Nmap的命令语法如下：

```
nmap [Scan Type(s)][Options]{target specification}
```

知识讲解

  - 目标说明(target specification)
    - 例如：对IP地址介于192.168.1.1到192.168.1.254之间的主机执行TCP SYN扫描

```
$ sudo nmap -sS 192.168.1.1-254
```

其中，1-254表示范围。范围形式的表示不仅限于IP地址的最后八位，例如：

```
$ sudo nmap -sS 192.168.1-10.1-254
```

总结和答疑