Mitnick 攻击实验室

版权归杜文亮所有
本作品采用 Creative Commons 署名-非商业性使用-相同方式共享 4.0 国际许可协议授权。如果您重新混合、改变这个材料，或基于该材料进行创作，本版权声明必须原封不动地保留，或以合理的方式进行复制或修改。

1 概述

凯文 · 米特尼克（Kevin Mitnick）可能是美国最著名的黑客之一。他曾在 FBI 的通缉犯名单上。在逃亡期间，他对黑客攻击蜂窝电话网络产生了兴趣，并需要一些能够帮助他做到这一点的专用软件。这将他引向了筱村毅（Tsutomu Shimomura），一位在圣地亚哥超级计算中心工作的研究人员，也是蜂窝电话网络安全领域的领先研究者之一，掌握了米特尼克想要的代码。

在 1994 年，米特尼克成功发起了对筱村电脑的攻击，利用了 TCP 协议的漏洞以及筱村的两台电脑之间的信任关系。此次攻击引发了两人之间的戏剧性较量，最终导致了米特尼克的逮捕。这场较量后来被改编成了书籍和好莱坞电影。该攻击现被称为米特尼克攻击，这是一种特殊类型的 TCP 会话劫持。

本实验的目标是重现经典的米特尼克攻击，以便学生能够获得此次攻击的第一手经验。我们将模拟当初筱村电脑上的设置，然后发起米特尼克攻击，以在筱村的两台电脑之间创建伪造的 TCP 会话。如果攻击成功，我们应该能够在筱村的电脑上运行任何命令。此实验涵盖以下主题：

- TCP 会话劫持攻击
- TCP 三路握手协议
- 米特尼克攻击
- 远程 shell `rsh`
- 数据包嗅探和伪造

阅读材料和视频 详细的 TCP 会话劫持内容可以在以下资料中找到：

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

实验环境。 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 米特尼克攻击如何工作

米特尼克攻击是 TCP 会话劫持攻击的一种特殊情况。与其劫持受害者 A 和 B 之间的现有 TCP 连接，米特尼克攻击首先代表他们创建 A 和 B 之间的 TCP 连接，然后自然地劫持该连接。

在实际的米特尼克攻击中，主机 A 称为 X-Terminal，这是一个目标。米特尼克想要登录到 X-Terminal 并在其上运行命令。主机 B 是一个受信任的服务器，它被允许无需密码登录 X-Terminal。为了登录 X-Terminal，米特尼克必须伪装成受信任的服务器，这样他就不需要提供任何密码。图 1描绘了该攻击的高层次动作。此攻击有四个主要步骤。

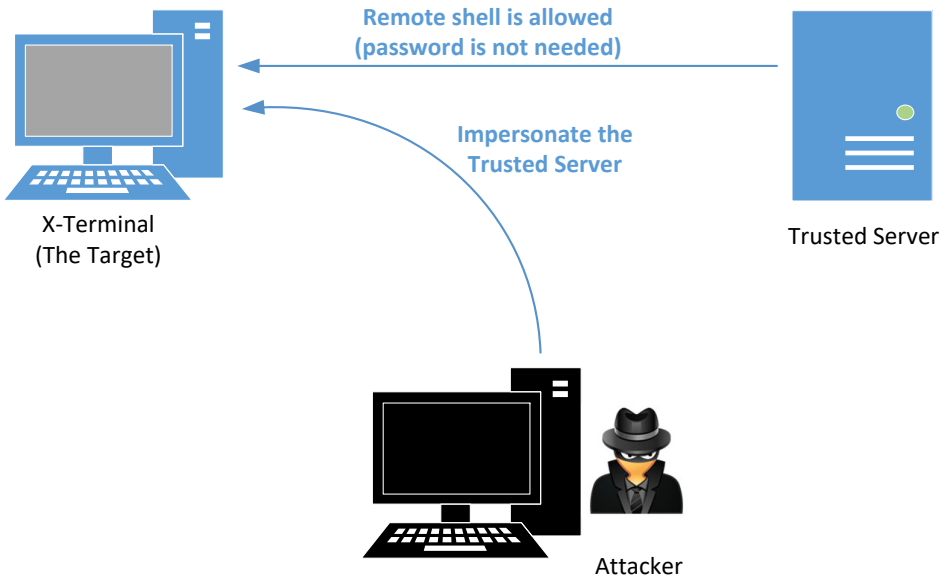

图 1: 米特尼克攻击的示意图

步骤 1: 序列号预测。 在攻击之前，米特尼克需要了解 X-Terminal 上的初始序列号（ISN）的模式（在那个时代，ISN 并不随机）。米特尼克向 X-Terminal 发送 SYN 请求，并收到 SYN+ACK 响应，然后他发送 RESET 数据包到 X-Terminal，以清除 X-Terminal 队列中的半开连接（以防止队列被填满）。在重复此操作 20 次后，他发现两个连续的 TCP ISN 之间存在模式。这使米特尼克能够预测 ISN，这是攻击所必需的。

步骤 2: 对受信任服务器的 SYN 泛滥攻击。 为了从受信任服务器向 X-Terminal 发送连接请求，米特尼克需要从受信任服务器向 X-Terminal 发送 SYN 数据包。X-Terminal 会回复一个 SYN+ACK 数据包，该数据包会发送到受信任服务器。由于受信任服务器并未实际发起请求，因此它会向 X-Terminal 发送 RESET 数据包，请求 X-Terminal 停止三路握手。此行为对米特尼克攻击造成了麻烦。

为了解决这个问题，米特尼克必须使受信任的服务器保持静默。因此，在伪造之前，米特尼克对服务器发起了 SYN 泛滥攻击。那时，操作系统对 SYN 泛滥攻击非常脆弱。攻击实际上可以让受信任计算机关闭，完全使其静默。

步骤 3: 伪造 TCP 连接。 米特尼克想要使用 `rsh`（远程 shell）在 X-Terminal 上运行后门命令；一旦后门被设置，他就可以登录到 X-Terminal。为了在 X-Terminal 上运行远程 shell，米特尼克需要通过身份验证，即他需要在 X-Terminal 上拥有有效账户并知道其密码。显然，他没有这些。

筱村常常需要从受信任的服务器登录到 X-Terminal。为了避免每次都输入密码，他在 X-Terminal 上的`.rhosts` 文件中添加了一些信息，以便当他从受信任的服务器登录到 X-Terminal 时，不会要求密码。这在当时是一种相当常见的做法。有了这个设置，筱村可以使用 `rsh` 从受信任的服务器运行命令，或者使用 `rlogin` 登录到 X-Terminal，而无需输入任何密码。米特尼克想要利用这种信任关系。

他需要在受信任的服务器和 X-Terminal 之间创建 TCP 连接，然后在该连接中运行 `rsh` 。他首先向 X-Terminal 发送一个 SYN 请求，使用受信任服务器的 IP 作为源 IP 地址。X-Terminal 随后向服务器发送 SYN+ACK 响应。由于服务器已关闭，因此它不会发送 RESET 来关闭连接。

为了完成三路握手协议，米特尼克需要伪造一个 ACK 数据包，该数据包必须确认 X-Terminal 的 SYN+ACK 数据包中的序列号。不幸的是，SYN+ACK 响应只发送给受信任服务器，而不发送给米特尼克，因此他无法看到序列号。然而，由于之前的调查，米特尼克能够预测这个数字，因此他能够成功伪造发往 X-Terminal 的 ACK 响应，以完成 TCP 的三路握手。

步骤 4: 运行远程 shell。 通过建立的受信任服务器与 X-Terminal 之间的 TCP 连接，米特尼克可以向 X-Terminal 发送远程 shell 请求，请求其执行命令。使用此命令，米特尼克希望在 X-Terminal 上创建一个后门，以便他能随时在 X-Terminal 上获得 shell，而无需重复攻击。

他所需做的就是在 X-Terminal 的`.rhosts` 文件中添加`"+ +"`。他可以通过在 X-Terminal 上使用 `rsh` 执行以下命令来实现这一点：`"echo + + > .rhosts"`. 由于 `rsh` 和 `rlogin` 程序使用`.rhosts` 文件进行身份验证，添加此内容后，X-Terminal 将信任来自任何人的每个 `rsh` 和 `rlogin` 请求。

3 使用容器设置实验环境

3.1 容器设置

在本实验中，我们需要三台机器，一台用于 X-Terminal，一台用于受信任服务器，另一台用于攻击者。在实际的米特尼克攻击中，攻击者机器是远程机器。为了简化，我们在同一网络上放置了这三台机器。学生可以使用三台虚拟机进行本实验，但使用容器会更加方便。实验环境如图 2所示。

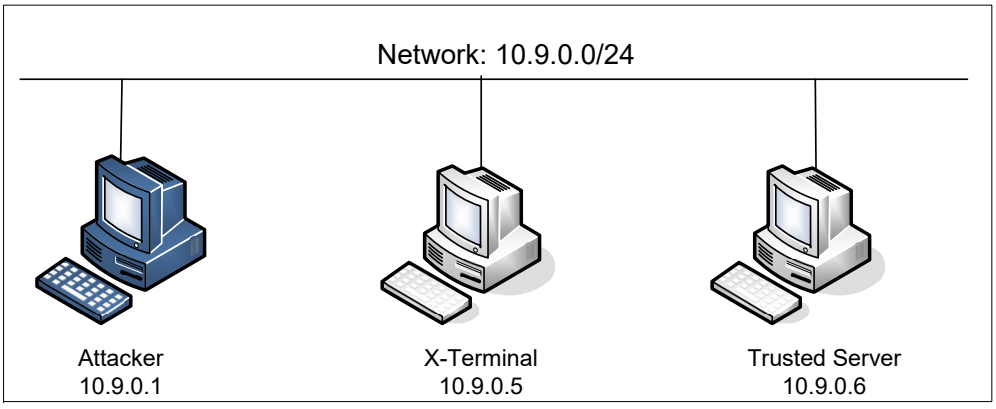

图 2: 实验环境设置

3.2 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
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。

[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

3.3 关于攻击者容器

在本实验中，我们可以使用虚拟机或攻击者容器作为攻击者机器。如果你查看 Docker Compose 文件，你会发现攻击者容器的配置与其他容器不同。以下是这些区别：

- 共享文件夹。当我们使用攻击者容器发起攻击时，我们需要将攻击代码放入攻击者容器内。

  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。

- 特权模式。为了能够在运行时修改内核参数（使用 `sysctl`），例如为了启用 IP 转发，容器需要被赋予特权。这是通过在容器中的 Docker Compose 文件包含以下条目来实现的。

  ```
  privileged: true
  ```

3.4 安装 rsh 程序（不需要执行任何操作）

远程 shell `rsh` 是一个命令行程序，可以远程执行 shell 命令。虽然我们将在此任务中使用 `rsh` ，但我们应该知道 `rsh` 和 `rlogin` 程序并不安全，而且它们不再使用。它们已被更安全的程序所取代，如 `ssh`。这就是为什么在现代 Linux 操作系统中，`rsh` 命令实际上是一个指向 `ssh` 程序的符号链接。

```
$ ls -al /etc/alternatives | grep rsh
lrwxrwxrwx   1 root root    12 Jul 25  2017 rsh -> /usr/bin/ssh
```

为了重现米特尼克攻击，我们需要安装不安全版本的 `rsh` 程序。显然，旧版本的 `rsh` 已经不再工作，但一个开源项目重新实现了远程 shell 客户端和服务器。它被称为 `rsh-redone`。我们可以使用以下命令安装 `rsh` 服务器和客户端。**注意：** `rsh` 程序已经安装在 X-Terminal 和 Trusted Server 容器中（请参见容器镜像文件夹内的 `Dockerfile`）。

```
$ sudo apt-get install rsh-redone-client
$ sudo apt-get install rsh-redone-server
```

3.5 配置

rsh 服务器程序使用两个文件进行身份验证，.rhosts 和 /etc/hosts.equiv。每次服务器接收到远程命令请求时，它都会检查/etc/hosts.equiv。如果请求来自文件中存储的主机名，服务器将接受该请求而不要求输入密码。如果/etc/hosts.equiv 不存在或没有该主机名，rsh 将检查用户主目录中的 .rhosts 文件。

筱村常常需要从受信任的服务器在 X-Terminal 上运行远程命令。为了避免输入密码，他在 X-Terminal 上创建了.rhosts 文件，并将受信任服务器的 IP 地址放入该文件中。请注意，.rhosts 文件必须位于用户主目录的顶级，并且只能由所有者/用户写入。

请使用以下命令在 X-Terminal 上设置.rhosts 文件。需要注意的是，当我们进入容器时，我们将处于 root 账户中。在本实验中，我们需要切换到名称为“seed”的普通用户帐户，这个账户已经在容器中创建：

```
# su seed          ← 切换到seed帐户
$ cd               ← 进入seed的主目录
$ touch .rhosts    ← 创建空文件
$ echo [服务器的IP地址] > .rhosts
$ chmod 644 .rhosts
```

要验证配置，请尝试在受信任服务器上运行以下命令。

```
# su seed          ← 切换到seed帐户
$ rsh [X-Terminal的IP] date
```

如果命令打印当前日期和时间，则配置正常工作。如果您看到“身份验证失败”，则设置中的某些内容可能不正确。常见错误之一是在.rhosts 文件上的权限：您应该确保该文件仅对所有者可写。

允许所有 要允许用户从所有 IP 地址在 X-Terminal 上执行命令，我们只需在.rhosts 文件中放入两个加号（"+ +"）。这非常危险，任何人都不应该这样做。但如果您是一个攻击者，这是一个设置后门的方便方式。正如我们之前提到的，这正是米特尼克攻击中使用的方式。

4 任务 1：模拟 SYN 泛滥

米特尼克攻击发生时，操作系统易受 SYN 泛滥攻击的影响，这可能使目标机器静默甚至关闭。然而，对于现代操作系统，SYN 泛滥已经不能再造成如此损害。我们将模拟这种效果。

我们可以手动停止受信任服务器容器，但这并不够。当 X-Terminal 接收到来自受信任服务器的 SYN 数据包时，它将以 SYN+ACK 数据包作出响应。在发送此数据包之前，它需要知道受信任服务器的 MAC 地址。会先检查 ARP 缓存。如果没有受信任服务器的条目，X-Terminal 将发送 ARP 请求数据

包以请求 MAC 地址。由于受信任服务器已被静默，所以没有人会回应该 ARP 请求，因此 X-Terminal 无法发送响应。因此，TCP 连接不会建立。

在实际攻击中，受信任服务器的 MAC 地址实际上在 X-Terminal 的 ARP 缓存中。即使没有，在使受信任服务器静默之前，我们可以简单地伪造一个 ICMP 回显请求从受信任服务器发送到 X-Terminal，这将触发 X-Terminal 回复受信任服务器，从而得到受信任服务器的 MAC 地址，并将其保存到缓存中。

为了简化任务，在停止受信任服务器之前，我们将从 X-Terminal 对其进行 ping 一次，然后使用 `arp` 命令检查并确保 MAC 地址在缓存中。需要注意的是，如果操作系统无法使用缓存的 MAC 地址到达目标，缓存条目可能会被操作系统删除。为了简化攻击，您可以在 X-Terminal 上运行以下命令以永久添加一个条目到 ARP 缓存（需要在 root 帐户中运行）：

```
# arp -s [服务器的IP] [服务器的MAC]
```

5 任务 2：伪造 TCP 连接和 rsh 会话

现在我们“让”受信任服务器“关闭”，我们可以冒充受信任服务器，并尝试与 X-Terminal 发起 `rsh` 会话。由于 `rsh` 在 TCP 之上运行，我们首先需要在受信任服务器和 X-Terminal 之间建立 TCP 连接，然后在此 TCP 连接中运行 `rsh` 。

米特尼克攻击中的一个困难是预测 TCP 序列号。当时 TCP 序列号是不随机的，因此这成为可能。然而，现代操作系统现在会随机化其 TCP 序列号（作为对 TCP 会话劫持攻击的对策），因此预测这些数字变得不可行。为了模拟原始米特尼克攻击的情况，我们允许学生嗅探数据包，以便他们能够获取序列号，而不是进行猜测。

限制 为了尽可能接近原始米特尼克攻击，即使学生能够从 X-Terminal 嗅探 TCP 数据包，他们也不能使用捕获数据包中的所有字段，因为在实际攻击中，米特尼克无法嗅探数据包。当学生编写他们的攻击程序时，他们只能使用捕获数据包中的以下字段。如果使用其他字段，将会受到惩罚。

- **TCP 序列号字段**（这不包括确认字段）。
- **TCP 标志字段**。这使我们能够知道捕获的 TCP 数据包类型。在实际的米特尼克攻击中，米特尼克确切知道 X-Terminal 发送了哪种类型的数据包，因为它们是 TCP 三次握手协议的一部分。我们允许学生在任务简化中使用此字段。
- **所有长度字段**，包括 IP 头长度，IP 总长度和 TCP 头长度。这些信息对于攻击并不是必需的。在实际的米特尼克攻击中，米特尼克确切知道它们的值。我们允许学生使用这些字段以简化任务。

`rsh` 的行为 为了在受信任服务器和 X-Terminal 之间创建伪造的 `rsh` 会话，我们需要了解 `rsh` 的行为。让我们从受信任服务器到 X-Terminal 开始一个 `rsh` 会话，然后使用 Wireshark 捕获它们之间的数据包（注意：我们将在攻击者虚拟机上运行 Wireshark，请确保选择正确的网络接口，与 `10.9.0.0/24` 网络对应）。我们使用以下命令通过 `rsh` 在主机 A 上运行主机 B 的 `date` 命令。

```
// 在受信任的服务器上
$ rsh 10.9.0.5 date
```

以下是此 rsh 会话的数据包跟踪。这里 10.9.0.6 是受信任服务器的 IP 地址，10.9.0.5 是 X-Terminal 的 IP 地址。如果数据包不携带 TCP 数据，则长度信息（即 Len=0）被省略。

Listing 1: rsh 会话的数据包跟踪

```
# 第一个连接
   源 IP       目标 IP    TCP 头部
1  10.9.0.6  10.9.0.5  1023 -> 514 [SYN] Seq=778933536
2  10.9.0.5  10.9.0.6  514 -> 1023 [SYN,ACK] Seq=10879102 Ack=778933537
3  10.9.0.6  10.9.0.5  1023 -> 514 [ACK] Seq=778933537 Ack=10879103
4  10.9.0.6  10.9.0.5  1023 -> 514 [ACK] Seq=778933537 Ack=10879103 Len=20
                       RSH会话建立
                       数据: 1022\x00seed\x00seed\x00date\x00
5  10.9.0.5  10.9.0.6  514 -> 1023 [ACK] Seq=10879103 Ack=778933557

# 第二个连接
6  10.9.0.5  10.9.0.6  1023 -> 1022 [SYN] Seq=3920611526
7  10.9.0.6  10.9.0.5  1022 -> 1023 [SYN,ACK] Seq=3958269143 Ack=3920611527
8  10.9.0.5  10.9.0.6  1023 -> 1022 [ACK] Seq=3920611527 Ack=3958269144

# 返回第一个连接
9  10.9.0.5  10.9.0.6  514 -> 1023 [ACK] Seq=10879103 Ack=778933557 Len=1
                       数据: \x00
10 10.9.0.6  10.9.0.5  1023 -> 514 [ACK] Seq=778933557 Ack=10879104
11 10.9.0.5  10.9.0.6  514 -> 1023 [ACK] Seq=10879104 Ack=778933557 Len=29
                       数据: Sun Feb 16 13:41:17 EST 2020
```

我们可以观察到，rsh 会话包含两个 TCP 连接。第一个连接由主机 A（客户端）发起。主机 B 上的 rshd 进程在 514 端口侦听连接请求。数据包 1 到 3 用于三路握手协议。连接建立后，客户端将 rsh 数据（包括用户 ID 和命令）发送到主机 B（数据包 4）。rshd 进程将验证用户，如果用户通过身份验证，rshd 将与客户端发起一个单独的 TCP 连接。

第二个连接用于发送错误消息。在上面的跟踪中，由于没有错误，因此未使用此连接，但必须成功建立该连接，否则 rshd 将不会继续。数据包 6 到 7 是第二个连接的三路握手协议。

在第二个连接建立后，主机 B 将发送一个零字节到客户端（使用第一个连接），主机 A 将确认该数据包。在这之后，主机 B 上的 rshd 将运行客户端发送的命令，命令的输出将通过第一个连接发送回客户端。学生可以使用 Wireshark 捕获 rsh 会话并研究其行为，然后再发起米特尼克攻击。我们将攻击任务分为两个子任务，每个子任务专注于一个连接。

5.1 任务 2.1: 伪造第一个 TCP 连接

第一个 TCP 连接由攻击者通过伪造的 SYN 数据包发起。如图 3所示，当 X-Terminal 接收到 SYN 数据包后，它会向受信任服务器发送 SYN+ACK 数据包。由于服务器已被关闭，因此它不会重置连接。攻击者位于同一网络，可以嗅探数据包以获取序列号。

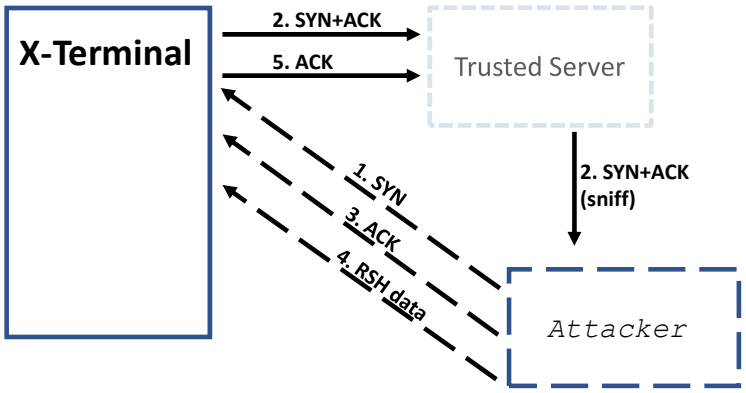

图 3: 第一个连接

步骤 1: 伪造一个 SYN 数据包。 学生应该编写程序来伪造一个从受信任服务器到 X-Terminal 的 SYN 数据包（见数据包 1，列出 1）。TCP 标头中有六个标准代码位，可以在标志字段中设置。以下代码示例显示了如何设置标志字段以及如何检查某些位是否在标志字段中设置。

```
# 'U': URG位
# 'A': ACK位
# 'P': PSH位
# 'R': RST位
# 'S': SYN位
# 'F': FIN位

tcp = TCP()

# 设置SYN和ACK位
tcp.flags = "SA"

# 检查SYN和ACK是否是唯一设置的位
if tcp.flags == "SA":

# 检查SYN和ACK位是否被设置
if 'S' in tcp.flags and 'A' in tcp.flags:
```

需要注意的是，SYN 数据包的源端口必须来自端口 `1023`。如果使用不同的端口，`rsh` 在连接建立后会重置该连接。如果此步骤成功，通过 Wireshark，我们应该能够看到来自 X-Terminal 的 SYN+ACK 数据包（见数据包 2，列出 1）。

步骤 2: 回复 SYN+ACK 数据包。 在 X-Terminal 发送 SYN+ACK 后，受信任服务器需要发送 ACK 数据包以完成三路握手协议。数据包中的确认号应为 `S+1`，其中 `S` 是 SYN+ACK 数据包中的序列号。见数据包 3，列出 1。

在实际的米特尼克攻击中，攻击者无法看到 SYN+ACK 数据包，因为它是发送给受信任服务器的，而不是给攻击者。因此，米特尼克不得不猜测序列号的值。在本实验中，我们允许学生通过嗅探数据包

获取序列号。

学生需要编写一个使用 Scapy 的嗅探与伪造程序，并在攻击者机器上运行。以下是一个可能有用的嗅探与伪造程序的骨架。请确保遵循本节开头描述的限制，否则将会受到惩罚。

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

x_ip      = "10.9.0.5"  # X-Terminal
x_port    = 514         # X-Terminal使用的端口号

srv_ip    = "10.9.0.6"  # 受信任服务器
srv_port  = 1023        # 受信任服务器使用的端口号

# 在伪造的SYN中加1序列号
seq_num     = 0x1000 + 1


def spoof(pkt):
  global seq_num   # 我们将在函数中更新此全局变量

  old_ip  = pkt[IP]
  old_tcp = pkt[TCP]

  # 打印调试信息
  tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4  # TCP数据长度
  print("{}:{} -> {}:{}  Flags={} Len={}".format(old_ip.src, old_tcp.sport,
                          old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len))

  # 构建响应的IP头
  ip = IP(src=srv_ip, dst=x_ip)

  # 检查是否为SYN+ACK数据包;
  # 如果是，则伪造一个ACK数据包

  # ... 在此添加代码 ...

myFilter = 'tcp'   # 需要使过滤器更加具体
sniff(iface='br-****', filter=myFilter, prn=spoof)
                  ↖ 您需要在这里设置正确的值。
```

步骤 3: 伪造 rsh 数据包 一旦连接建立，攻击者需要向 X-Terminal 发送 rsh 数据。rsh 数据的结构如下所示。

```
[端口号]\x00[客户端用户ID]\x00[服务器用户ID]\x00[您的命令]\x00
```

数据包含四个部分：端口号、客户端用户 ID、服务器用户 ID 和命令。端口号将用于第二个连接（见任务 2.2）。在我们的容器中，客户端和服务器的用户 ID 都是 `seed`。这四个字段由一个字节 0 分隔。请注意，在 `rsh` 数据的末尾也有一个字节 0。以下是一个示例。在此示例中，我们告诉 X-Terminal，我们将监听端口 9090 以进行第二个连接，并且我们想要执行的命令是`"touch /tmp/xyz"`。

```
data = '9090\x00seed\x00seed\x00touch /tmp/xyz\x00'
send(IP()/TCP()/data, verbose=0)
```

学生应修改步骤 2 中编写的嗅探与伪造程序，以便向 X-Terminal 发送 `rsh` 数据包（见数据包 4，列出 1）。如果此步骤成功，通过 Wireshark 我们可以看到 X-Terminal 会向受信任服务器的端口 `9090` 发起 TCP 连接，这是在我们的 `rsh` 数据中指定的端口号。

在您的报告中，请描述在 X-Terminal 上是否执行了 `touch` 命令。还请附上您的 Wireshark 快照。

5.2 任务 2.2: 伪造第二个 TCP 连接

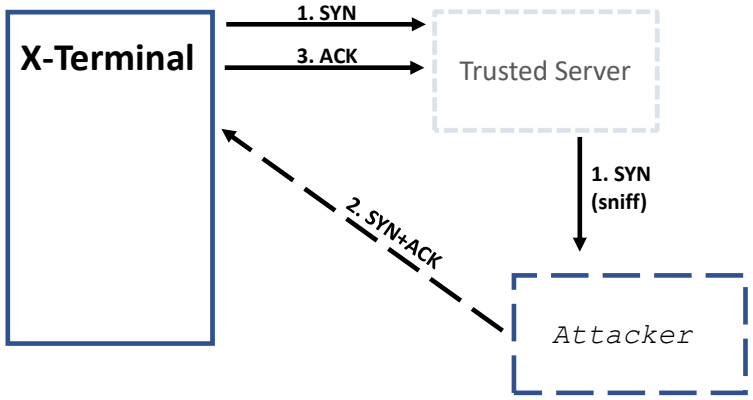

图 4: 第二个连接

在第一个连接建立后，X-Terminal 将发起第二个连接。此连接用于 `rshd` 发送错误消息。在我们的攻击中，我们不会使用此连接，但如果未建立该连接，`rshd` 将不会执行我们的命令。因此，我们需要使用伪造技术帮助 X-Terminal 和受信任服务器完成建立该连接。见图 4。

学生需要编写另一个嗅探与伪造程序，嗅探发送到受信任服务器端口 9090 的 TCP 流量（假设在任务 2.1 中使用了 `9090`）。当它看到 SYN 数据包时，它应该用 SYN+ACK 数据包作出响应。见数据包 7，列出 1中的描述。

如果两个连接都成功建立，`rshd` 将执行 `rsh` 数据包中包含的命令。请检查`/tmp` 文件夹，查看`/tmp/xyz` 是否被创建，时间戳是否与当前时间匹配。请在您的报告中附上证据。

6 任务 3: 设置后门

在任务 2 中，我们只是在攻击中运行了 `touch` 命令，以证明我们可以成功地在 X-Terminal 上运行命令。如果我们想要在后续运行更多命令，我们每次都要进行相同的攻击，这非常不方便。

米特尼克确实计划再次回到 X-Terminal。在初次攻击后，他在 X-Terminal 中植入了一个后门。这个后门使他每次想要的时候都能正常登录到 X-Terminal，而无需输入任何密码。为了实现这个目标，正如我们在 3.5 节中讨论的那样，我们所需要做的就是在 `.rhosts` 文件中添加字符串 `"+ +"`（以单行形式）。我们可以在我们的 `rsh` 数据中包含以下命令。

```
echo + + > .rhosts
```

学生应将任务 2 中的 `touch` 命令替换为上面的 `echo` 命令，然后重复攻击。如果攻击成功，攻击者应能够使用以下命令远程登录到 X-Terminal，而无需输入密码：

```
$ rsh [X-Terminal的IP]
```

`rsh` 程序可能未在攻击者容器上安装，但可以使用以下命令轻松安装：

```
# apt-get update && apt-get -y install rsh-redone-client
```

7 作业提交

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。