ARP 缓存中毒攻击实验

1 概要

地址解析协议（ARP）是一种用于在给定 IP 地址的情况下查询链路层地址（例如 MAC 地址）的通讯协议。ARP 协议是一个非常简单的协议，它并未实现任何安全措施。ARP 缓存中毒攻击是对 ARP 协议的一种常见攻击方式。使用此类攻击，攻击者可以让受害者接受伪造的 IP 到 MAC 的映射。这会导致受害者的数据包被重定向到具有伪造 MAC 地址的计算机，从而导致潜在的中间人攻击。

本实验的目标是让学生们获得对 ARP 缓存中毒攻击的第一手体验，并了解这种攻击可能造成的损害。学生将使用 ARP 攻击来发起中间人攻击。在这种攻击中，攻击者可以拦截并修改两个受害者 A 和 B 之间的数据包。本实验的另一个目标是让学生练习数据包嗅探和伪造技能，这些技能在网络安全领域至关重要，也是许多网络攻击与防御工具的基础。学生将使用 Scapy 来完成实验任务。本实验涵盖了以下主题：

- ARP 协议
- ARP 缓存中毒攻击
- 中间人攻击
- Scapy 编程

视频。 关于 ARP 协议和攻击的详细内容可以在以下视频中找到：

- SEED Book, *Internet Security: A Hands-on Approach*, 3rd Edition, by Wenliang Du. 详情请见 `https://www.handsonsecurity.net`.
- SEED Udemy 课程第 3 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 使用容器建立实验环境

在本实验中，我们需要三台机器。我们使用容器来建立实验环境，如图 1 所示。在这个配置中，我们有一个攻击者机器（主机 M），用于对其他两台机器 Host A 和 Host B 发起攻击。这三台机器必须在同一局域网内，因为 ARP 缓存中毒攻击仅限于局域网。我们使用容器来建立实验环境。

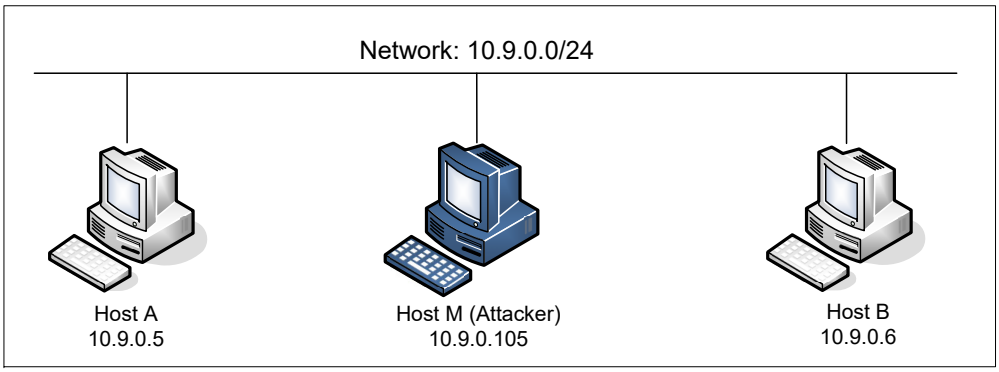

图 1: 实验环境搭建

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
9652715c8e0a  hostC-10.9.0.7
```

[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

```
$ 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，我们可以选择使用 VM 或者攻击者的容器作为攻击机器。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。以下是它们之间的区别：

- 共享文件夹。当我们使用攻击者的容器发起攻击时，需要将攻击代码放在容器内部。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
  ```

- 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
  ```

2.3 嗅探数据包

在本实验中，能够嗅探数据包是非常重要的。因为如果事情没有按预期进行，能够查看数据包的去向可以帮助我们找到问题所在。这里有几种不同的方式来进行数据包嗅探：

- 在容器上运行 `tcpdump`。我们已经在每个容器上安装了 `tcpdump`。要嗅探通过特定接口的数据包，我们只需找出该接口的名称，然后执行以下操作（假设接口名为 `eth0`）：

  ```
  # tcpdump -i eth0 -n
  ```

  需要注意的是，在容器内部，由于 Docker 创建的隔离，当我们在一个容器内运行 `tcpdump` 时，我们只能嗅探进出该容器的数据包，而不能嗅探其他容器之间的数据包。然而，如果容器在网络设置中使用了 `host` 模式，则可以嗅探其他容器的数据包。

- 在 VM 上运行 tcpdump。如果我们在虚拟机上运行 tcpdump，则不会受到容器的限制，并且可以嗅探所有容器之间传递的数据包。与容器不同，VM 中网络接口名称有所不同。在容器中，每个接口名通常以 eth 开头；而在 VM 中，由 Docker 创建的网络接口名称以 br- 开头，后面跟着网络的 ID。您总是可以使用 ip address 命令在 VM 和容器上获取接口名称。
- 我们还可以在 VM 上运行 Wireshark 来嗅探数据包。类似于 tcpdump，我们需要选择要嗅探的接口。

3 任务 1：ARP 缓存中毒攻击

本任务的目的是使用数据包伪造对目标发起 ARP 缓存中毒攻击，使得当两台受害者机器 A 和 B 尝试相互通信时，其数据包将被攻击者拦截并修改，从而使攻击者成为 A 和 B 之间的中间人。这被称为中间人（MITM）攻击。在本任务中，我们的重点在于 ARP 缓存中毒部分。以下代码示例展示了如何使用 Scapy 构造一个 ARP 数据包。

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

E = Ether()
A = ARP()
A.op = 1     # 1 为 ARP 请求; 2 为 ARP 响应

pkt = E/A
sendp(pkt)
```

上述程序构建并发送了一个 ARP 数据包。请设置必要的属性来构建正确的 ARP 包。我们可以通过 ls(ARP) 和 ls(Ether) 查看 ARP 和 Ether 类的属性名称。如果某个属性没有被设置，Scapy 会使用默认值（参见输出的第三列）：

```
$ python3
>>> from scapy.all import *

>>> ls(Ether)
dst        : DestMACField                    = (None)
src        : SourceMACField                  = (None)
type       : XShortEnumField                 = (36864)

>>> ls(ARP)
hwtype     : XShortField                     = (1)
ptype      : XShortEnumField                 = (2048)
hwlen      : ByteField                       = (6)
plen       : ByteField                       = (4)
op         : ShortEnumField                  = (1)
hwsrc      : ARPSourceMACField               = (None)
psrc       : SourceIPField                   = (None)
```

```
hwdst      : MACField                             = ('00:00:00:00:00:00')
pdst       : IPField                              = ('0.0.0.0')
```

在本任务中，我们有三台机器（容器），A、B 和 M。我们将使用 M 作为攻击者机器。我们希望让 A 在其 ARP 缓存中添加一个伪造的条目，使得 B 的 IP 地址被映射到 M 的 MAC 地址。我们可以使用以下命令检查一台计算机的 ARP 缓存。如果你想查看与特定接口关联的 ARP 缓存，可以使用 `-i` 选项。

```
$ arp -n
Address     HWtype  HWaddress           Flags Mask  Iface
10.0.2.1    ether   52:54:00:12:35:00   C           enp0s3
10.0.2.3    ether   08:00:27:48:f4:0b   C           enp0s3
```

有许多方法可以实施 ARP 缓存中毒攻击。学生需要尝试以下三种方法，并汇报每种方法是否有效。

- **任务 1.A（使用 ARP 请求）**在主机 M 上构造一个 ARP 请求包，将 B 的 IP 地址映射到 M 的 MAC 地址，将该包发送给 A 并检查攻击是否成功。
- **任务 1.B（使用 ARP 响应）**在主机 M 上构造一个 ARP 响应包，将 B 的 IP 地址映射到 M 的 MAC 地址，并将该包发送给 A 并检查攻击是否成功。同时在以下两种情景下尝试攻击并汇报攻击结果：
  - 情景 1：B 的 IP 地址已经存在于 A 的缓存中。
  - 情景 2：B 的 IP 地址未出现在 A 的缓存中。你可以使用命令 `"arp -d a.b.c.d"` 来删除 IP 地址 `a.b.c.d` 对应的 ARP 缓存条目。
- **任务 1.C（使用 ARP 免费消息）**在主机 M 上构造一个 ARP 免费数据包，并将 B 的 IP 地址映射到 M 的 MAC 地址。请在与任务 1.B 中描述的相同两种情况下发起攻击。

  ARP 免费数据包是一种特殊的 ARP 请求包，当一台宿主机器需要更新其他机器的 ARP 缓存中的过时信息时会使用它。免费 ARP 数据包具有以下特点：
  - 源和目标的 IP 地址相同，是发出该消息的主机的 IP 地址。
  - ARP 头部和以太网头部的目标 MAC 地址都是广播 MAC 地址 (`ff:ff:ff:ff:ff:ff`)。
  - 不期望有响应。

4 任务 2：使用 ARP 缓存中毒攻击在 Telnet 实施中间人攻击

主机 A 和 B 正在通过 Telnet 进行通信，而主机 M 希望拦截它们之间的通信以便对 A 和 B 传送的数据进行修改。图 2 描述了该设置。我们已经在容器内部创建了一个名为 `seed` 的帐户，密码是 `dees`。你可以通过 Telnet 连接到此账户。

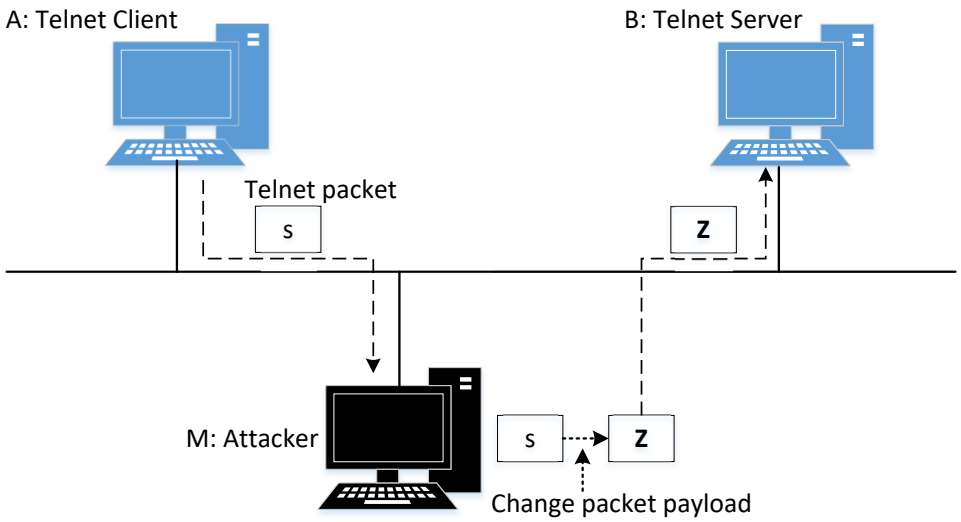

图 2: 针对 Telnet 的中间人攻击

步骤 1（发起 ARP 缓存中毒攻击）。 首先，主机 M 对 A 和 B 均执行 ARP 缓存中毒攻击，使得在 A 的 ARP 缓存中，B 的 IP 地址被映射到 M 的 MAC 地址；在 B 的 ARP 缓存中，A 的 IP 地址也被映射到 M 的 MAC 地址。完成此步骤后，A 和 B 之间的数据包都将发送给 M。我们将使用任务 1 中的 ARP 缓存中毒攻击来实现这一目标。如果你能不断地发送伪造数据包（例如每 5 秒一次）就更好了。否则，伪造的映射可能会被替换。

步骤 2（测试）。 在攻击成功后，请尝试在主机 A 和 B 之间相互 ping，并汇报你的观察结果。请在报告中展示 Wireshark 的结果。在执行此步骤之前，请确保主机 M 的 IP 转发（IP forwarding）已关闭。你可以通过以下命令完成此操作。

```
# sysctl net.ipv4.ip_forward=0
```

步骤 3（开启 IP 转发）。 现在我们将在主机 M 上开启 IP 转发，因此它将转发 A 和 B 之间的数据包。请运行以下命令并重复步骤 2，描述你的观察结果。

```
# sysctl net.ipv4.ip_forward=1
```

步骤 4（实施中间人攻击）。 我们现在可以对 A 和 B 之间的 Telnet 数据进行修改。假设 A 是 Telnet 客户端，B 是 Telnet 服务器。在 A 连接到 B 上的 Telnet 服务器后，在 A 的 Telnet 窗口中键入的每个字符，都会生成一个 TCP 数据包并发送给 B。我们希望拦截这个 TCP 数据包，并将每个输入的字符替换为一个固定字符（例如 Z）。这样无论用户在 A 上键入什么内容，Telnet 都会始终显示 Z。

根据前面的步骤，我们可以将 TCP 数据包重定向到主机 M，但此时我们希望 M 不转发这些数据包，而是用一个伪造的数据包来代替。我们将编写一个程序来完成这一目标。我们需要做到以下几点：

- 首先保持 M 上的 IP 转发，以便 A 和 B 之间可以建立 Telnet 连接。一旦连接建立，我们使用以

下命令关闭 IP 转发。请在 A 的 Telnet 窗口中输入一些内容并报告观察结果：

```
# sysctl net.ipv4.ip_forward=0
```

- 在主机 M 上运行你写的程序，捕获从 A 到 B 发送的数据包，然后生成一个伪造数据包（只修改 TCP 数据部分）。对于从 B 到 A 的数据包（Telnet 响应），我们不做任何更改，因此伪造数据包与原始的完全相同。

为了帮助学生知道从何开始，我们提供了一个示范程序。该程序捕获所有 TCP 数据包，并对从 A 到 B 的数据包进行一些修改（修改部分并未展示，因为这是任务的一部分）。对于从 B 到 A 的数据包，程序不做任何更改。

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

IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"

def spoof_pkt(pkt):
    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
        # 根据捕获的数据包创建一个新的数据包。
        # 1) 我们需要删除 IP 和 TCP 头部的校验和，
        #    因为我们的修改会使它们无效。
        #    Scapy 会在这些字段缺失时重新计算它们。
        # 2) 我们还删除了原始的 TCP 有效载荷。

        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)

        ################################################################
        # 根据旧的有效载荷构建新的有效载荷。
        # 学生需要实现这部分代码。

        if pkt[TCP].payload:
            data = pkt[TCP].payload.load  # 原始的有效载荷数据
            newdata = data   # 在此示例代码中未做更改

            send(newpkt/newdata)
        else:
            send(newpkt)
        ################################################################
```

```
    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
         # 根据捕获的数据包创建一个新的数据包
         # 不进行任何修改。

         newpkt = IP(bytes(pkt[IP]))
         del(newpkt.chksum)
         del(newpkt[TCP].chksum)
         send(newpkt)

f = 'tcp'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

需要注意的是，上述代码捕获了所有 TCP 数据包，包括由程序本身生成的数据包。这是不可取的，因为它会影响程序的表现。学生需要更改过滤器以确保不捕获自身的数据包。

Telnet 的行为。 在 Telnet 中，在通常情况下，我们每在 Telnet 窗口中键入一个字符就会触发一个单独的 TCP 数据包，但如果输入速度很快，几个字符可能会在一个数据包中被一起发送。这就是为什么在从客户端到服务器端的 Telnet 数据包中，有效载荷通常只包含一个字符。被发送到服务器的字符将由服务器回弹，并在客户端窗口中显示。因此，我们看到客户端窗口中的内容并不是直接的输入结果。无论我们在客户端窗口中输入什么内容，它都需要经过一次往返才能显示。如果网络断开连接，则无论我们在客户端窗口中输入什么内容都不会显示，直到网络恢复。如果攻击者在这个过程中将字符更改为 Z，则 Z 将在 Telnet 客户端窗口中显示出来，尽管你实际并未输入 Z。

5 任务 3：使用 ARP 缓存中毒攻击在 Netcat 实施中间人攻击

本任务与任务 2 类似，只是主机 A 和 B 使用 `netcat` 而不是 `telnet` 进行通信。主机 M 希望拦截它们之间的通信并修改 A 和 B 通信的数据。你可以使用以下命令中建立 A 到 B 的 `netcat` TCP 连接：

```
在主机 B（服务器，IP 地址是 10.9.0.6）上运行以下命令：
# nc -lp 9090

在主机 A（客户端）上运行以下命令：
# nc 10.9.0.6 9090
```

一旦建立连接，你可以在 A 中键入信息。每行信息都将被放入一个 TCP 数据包中发送给 B，B 只是显示该信息。你的任务是将信息中你的姓（拼音）替换为一串 A，串的长度应与你姓的拼音长度相同，否则会扰乱 TCP 序列号，从而导致整个 TCP 连接失败。你需要使用真实的姓来完成此任务，以便我们知道该任务是你完成的。

6 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。