1. **：WEB侦查**

本章内容：

1. **使用Mechanize匿名浏览互联网**
2. **Python使用Beautiful Soup映射WEB元素**
3. **使用Python与Google交互**
4. **使用Python和Twitter交互**
5. **自动钓鱼**

在我生命的八十七年中，我亲眼目睹了技术革命的演替。但却没有人完成了人的思考和需要这一问题。   
                                              —Bernard M. Baruch 美国第28到第32任总统的顾问

**简介：今天的社会工程学**

2010年，两个大规模的网络攻击改变了我们对网络战的理解。先前我们在第四章讨论过极光行动。在极光行动攻击中，瞄准了多个跨国的公司，雅虎，赛门铁克，Adobe等还有一些Google账户。华盛顿邮报报道这是一个新的有着先进水平的攻击。Stuxnet，第二次攻击，针对SCADA系统，特别是那些在伊朗的。网络维护者应该关注该蠕虫的发展，这是一个比极光行动更加先进和成熟的蠕虫攻击。尽管这两个网络攻击非常复杂，但他们有一个共同的关键点：他们的传播，至少部分是通过社会工程学传播的。不管多么复杂的和致命的网络攻击增加有效的社会工程学会增加攻击的有效性。在下面的章节中，我们将研究如何使用使用Python来实现自动化的社会工程学攻击。

在进行任何操作之前，攻击者应该有目标的详细信息，信息越多攻击的成功的机会越大。概念延伸到信息战争的世界。在这个邻域和当今时代，大部分所需的信息可以在互联网上找到，由于互联网庞大的规模，遗漏重要信息的可能性很高。为了防止信息丢失，计算机程序可以自动完成整个过程。Python是一个很好的执行自动化任务的工具，大量的第三方库允许我们轻松的和互联网，网站进行交互。

**攻击之前的侦查**

在本章中，我们通过程序对目标进行侦查。在这个发面关键是确保我们收集更多的信息量，而不被警惕性极高，能干的公司总部的网络管理员检测到。最后我们将看看如何汇总数据允许我们发动高度复杂的个性化的社会工程学攻击。确保在应用任何这些技术之前询问了执法官员和法律的意见。我们在这展示攻击和用过的工具是为了更好的理解他们的做法和知道如何在我们的生活中如何防范这种攻击。

**使用Mechanize库浏览互联网**

典型的计算机用户依赖WEB浏览器浏览网站和导航互联网。每一个站点都是不同的，可以包含图片，音乐和视频中的各种各样的组合。然而，浏览器实际上读取一个文本类型的文档，理解它，然后将他显示给用户，类似于一个Python程序的源文件和Python解释器的互动。用户可以使用浏览器访问站点或者使用不同的方法浏览他们的源代码。Linux下的我wget程序是个很受欢迎的方法。在Python中，浏览互联网的唯一途径是取回并下载一个网站的HTML源代码。有许多不同的库已经已经完成了处理WEB内容的任务。我们特别喜欢Mechanize，你在前几章已经用过。Mechanize：http://wwwsearch.sourceforge.net/mechanize/。

Mechanize主要的类Browser，允许任何可以在浏览器是上进行的操作。这个类也有其他的有用的方法是程序变得更简单。下面脚本演示了Mechanize最基本的使用：取回一个站点的源代码。这需要创建一个浏览器对象，然后调用open()函数。

**import mechanize**

**def viewPage(url):**

**browser = mechanize.Browser()**

**page = browser.open(url)**

**source\_code = page.read()**

**print(source\_code)**

**viewPage('http://www.syngress.com/')**

运行这个脚本，我们看到它打印出www.syngress.com首页的HTML代码。

**recon:∼# python viewPage.py**

**<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"**

**"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">**

**<html xmlns="http://www.w3.org/1999/xhtml">**

**<head>**

**<title>**

**Syngress.com - Syngress is a premier publisher of content in**

**the Information Security field. We cover Digital Forensics, Hacking**

**and Penetration Testing, Certification, IT Security and Administration, and**

**more.**

**</title>**

**<meta name="description" content="" /><meta name="keywords"**

**content="" />**

**<..SNIPPED..>**

我们将使用mechanize.Browser类来构建脚本，在本章中浏览互联网。但是你不会受它的约束，Python提供了几个不同的方法浏览。这章使用Mechanize由于他提供了特殊的功能。John J. Lee设计的Mechanize提供可状态编程，简单的HTML表格和方便的解析和处理，例如HTTP-Equiv这样的命令和刷新。此外，它提供给你的内在对象是匿名的。这一切都会在下面的章节中用到。

**匿名---增加代理，用户代理和Cookies**

现在我们有从互联网获取网页内容的能力，退一步想想接下来的处理很有必要。我们的程序和在浏览器中打开一个网站没有什么不同，因此，我们应该采取同样的步骤在正常的浏览网页时建立匿名。网站查找唯一标识符来识别网页游客有几种不同的方法。第一种方法是通过记录请求的IP来确认用户。这可以通过使用虚拟专用网络(VPN)或者tor网络来缓和。一旦一个客户连接到VPN，然后，所有的将通过VPN自动处理。Python可以连接到代理服务器，给程序添加匿名功能。Mechanize的Browser类可以指定一个代理服务器属性。简单的设置浏览器代理是不够巧妙的。有很多的免费的代理网络，所以用户可以进去选择它们，通过它们的功能浏览。在这个例子中，我们选择http://www.hidemyass.com/的HTTP代理。在你读到这里的时候这个代理很有可能已经不工作了。所以去这个网站得到使用不同HTTP代理的细节。此外，McCurdy维护了一个很好的代理列表在网站：http://rmccurdy.com/scripts/proxy/good.txt。我们将测试我们的代理访问NOAA网站，它会友好的告诉你访问该网站时你的IP地址。

**import mechanize**

**def testProxy(url, proxy):**

**browser = mechanize.Browser()**

**browser.set\_proxies(proxy)**

**page = browser.open(url)**

**source\_code = page.read()**

**print source\_code**

**url = 'http://ip.nefsc.noaa.gov/'**

**hideMeProxy = {'http': '216.155.139.115:3128'}**

**testProxy(url, hideMeProxy)**

虽然识别HTML源代码有一点困难，我们看到该网站人为我们的IP地址是216.155.139.115，我们的代理，成功！我们继续构建脚本。

**recon:∼# python proxyTest.py**

**<html><head><title>What's My IP Address?</title></head>**

**<..SNIPPED..>**

**<b>Your IP address is...</b></font><br><font size=+2 face=arial**

**color=red> 216.155.139.115</font><br><br><br><center> <font size=+2face=arial color=white> <b>Your hostname appears to be...</b></**

**font><br><font size=+2 face=arial color=red> 216.155.139.115.**

**choopa.net</font></font><font color=white**

**<..SNIPPED..>**

我们现在有一个简单的匿名浏览器。站点使用浏览器的user-agent字符串来识别唯一用户另一种方法。在正常情况下，user-agent字符串让网站知道关于浏览器的重要信息能制作HTML代码给用户更好的体验。然而，这些信息柏包含内核版本，浏览器版本，和其他关于用户的详细信息。恶意网站利用这些信息针对特定的浏览器进行精密的渗透利用，而其他网站利用这些信息来区分电脑是位与NAT网络还是私有网络。最近，一个丑闻被爆出，一个旅游网站利用user-agent字符串来检测MacBook用户并提供更昂贵的选择。

幸运的是，Mechanize改变user-agent字符串和改变代理一样简单。网站：http://www.useragentstring.com/pages/useragentstring.php为我们展示了一个巨大的有效的user-agent字符串名单供我们选择。我们将编写一个脚本来测试改变我们的user-agent字符串访问http://whatismyuseragent.dotdoh.com/来打印出我们的user-agent字符。

i**mport mechanize**

**def testUserAgent(url, userAgent):**

**browser = mechanize.Browser()**

**browser.addheaders = userAgent**

**page = browser.open(url)**

**source\_code = page.read()**

**print(source\_code)**

**url = 'http://whatismyuseragent.dotdoh.com/'**

**userAgent = [('User-agent','Mozilla/5.0 (X11; U; Linux 2.4.2-2 i586; en-US; m18) Gecko/20010131 Netscape6/6.01')]**

**testUserAgent(url, userAgent)**

运行这个脚本，我们看到我们可以用虚假的user-agent字符串来访问页面。

**recon:∼# python userAgentTest.py**

**<html>**

**<head>**

**<title>Browser UserAgent Test</title>**

**<style type="text/css">**

**<..SNIPPED..>**

**<p><a href="http://www.dotdoh.com" target="\_blank"><img src="logo.**

**gif" alt="Logo" width="646" height="111" border="0"></a></p>**

**<p><h4>Your browser's UserAgent string is: <span**

**class="style1"><em>Mozilla/5.0 (X11; U; Linux 2.4.2-2 i586; en-US;**

**m18) Gecko/20010131 Netscape6/6.01</em></span></h4>**

**</p>**

**<..SNIPPED..>**

最后，网站会返回一些包含独特标识的cookie给WEB浏览器允许网站识别重复的重复的访客。为了防止这一点，我们将执行其他函数从我们的WEB浏览器中清除cookie。另外一个Python标准库cookielib包含几个处理不同类型cookie的容器。这里使用的cookie类型包含储存各种不同的cookie到硬盘的功能。这个功能允许用户查看cookies而不必在初始化后返回给网站。让我们建立一个简单的脚本使用CookieJar来测试。我们将打开http://www.syngress.com页面作为我们的第一个例子。但现在我们打印浏览会话存储的cookie。

**import mechanize**

**import cookielib**

**def printCookies(url):**

**browser = mechanize.Browser()**

**cookie\_jar = cookielib.LWPCookieJar()**

**browser.set\_cookiejar(cookie\_jar)**

**page = browser.open(url)**

**for cookie in cookie\_jar:**

**print(cookie)**

**url = 'http://www.syngress.com/'**

**printCookies(url)**

运行这个脚本，我们可以看到来自网站的session id的cookie。

**recon:∼# python printCookies.py**

**<Cookie \_syngress\_session=BAh7CToNY3VydmVudHkiCHVzZDoJbGFzdCIAOg9zZYNzaW9uX2lkIiU1ZWFmNmIxMTQ5ZTQxMzUxZmE2ZDI1MSBlYTA4ZDUxOSIKZmxhc2hJQzonQWN0aW8uQ29udHJvbGxlcjo6Rmxhc2g6OkZsYXNoSGFzaHsABjoKQHVzZWR7AA%3D%3D--f80f741456f6c0dc82382bd8441b75a7a39f76c8 forwww.syngress.com/>**

**最终封装我们的代码为Python类**

已经有了几个功能，将浏览器作为参数，修改它，偶尔添加一个额外的参数。如果将这些添加到一个类里面将很有用，这些功能可以归结为一个浏览器对象简单的调用，而不是导入我们的函数到某个文件使用笨拙的语法调用。我们我们这么做可以扩展Browser类，我们的新Browser类将会有我们已经创建过的函数，以及初始化的附加功能。这将有利于提高代码的可读性，并封装所有的功能在Browser类中直接处理。

**import mechanize, cookielib, random, time**

**class anonBrowser(mechanize.Browser):**

**def \_\_init\_\_(self, proxies = [], user\_agents = []):**

**mechanize.Browser.\_\_init\_\_(self)**

**self.set\_handle\_robots(False)**

**self.proxies = proxies**

**self.user\_agents = user\_agents + ['Mozilla/4.0 ', 'FireFox/6.01','ExactSearch', 'Nokia7110/1.0']**

**self.cookie\_jar = cookielib.LWPCookieJar()**

**self.set\_cookiejar(self.cookie\_jar)**

**self.anonymize()**

**def clear\_cookies(self):**

**self.cookie\_jar = cookielib.LWPCookieJar()**

**self.set\_cookiejar(self.cookie\_jar)**

**def change\_user\_agent(self):**

**index = random.randrange(0, len(self.user\_agents))**

**self.addheaders = [('User-agent', (self.user\_agents[index]))]**

**def change\_proxy(self):**

**if self.proxies:**

**index = random.randrange(0, len(self.proxies))**

**self.set\_proxies({'http': self.proxies[index]})**

**def anonymize(self, sleep = False):**

**self.clear\_cookies()**

**self.change\_user\_agent()**

**self.change\_proxy()**

**if sleep:**

**time.sleep(60)**

我们的新类有一个默认的user-agents列表，接受列表添加进去，以及用户想使用的代理服务器列表。它还具有我们先前创建的三个功能，可以单独也可以同时使用匿名函数。最后，anonymize提供等待60秒的选项，增加在服务器日志请求访问之间的时间。同时也不改变提供的信息，该额外的步骤减小了被识别为相同的源地址的机会。增加时间和模糊的通过安全是一个道理，但是额外的措施是有帮助的，时间通常不是一个问题。另一个程序可以以相同的方式使用这个新类。文件anonBrowser.py包含新类，如果想在导入调用是看到它，我们必须将它保存在脚本的目录。

让我们编写我们的脚本，导入我们的新类。我有一个教授曾将帮助他四岁的女儿在线投票竞争小猫冠军。由于投票是在会话的基础上的，每个游客的票需要是唯一的。我们来看看是否我们能欺骗这个网站给予我们每次访问唯一的cookie。我们将匿名访问该网站四次。

**from anonBrowser import \***

**ab = anonBrowser(proxies=[],user\_agents=[('User-agent','superSecretBroswer')])**

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

**ab.anonymize()**

**print('[\*] Fetching page')**

**response = ab.open('http://kittenwar.com')**

**for cookie in ab.cookie\_jar:**

**print(cookie)**

运行该脚本，我们看到页面获得五次不同时间不同cookie的访问。成功！随着我们匿名访问类的建立，让我们抹去我们访问网站上的私人信息。

**recon:∼# python kittenTest.py**

**[\*] Fetching page**

**<Cookie PHPSESSID=qg3fbia0t7ue3dnen5i8brem61 for kittenwar.com/>**

**[\*] Fetching page**

**<Cookie PHPSESSID=25s8apnvejkakdjtd67ctonfl0 for kittenwar.com/>**

**[\*] Fetching page**

**<Cookie PHPSESSID=16srf8kscgb2l2e2fknoqf4nh2 for kittenwar.com/>**

**[\*] Fetching page**

**<Cookie PHPSESSID=73uhg6glqge9p2vpk0gt3d4ju3 for kittenwar.com/>**

**用匿名类抹去WEB页面**

现在我们可以用Python取回WEB内容。可以开始侦查目标了。我们可以通过检索大型的网站来开始我们的研究了。攻击者可以深入的调查目标的主页面寻找隐藏的和有价值的数据。然而这种搜索行动会产生大量的页面浏览器。移动网站的内容到本地能减少页面的浏览数。我们可以只访问页面一次，然后研究无数次。有一些框架可以这样做，但是我们将建立我们自己的，利用先前的anonBrowser类。让我们利用anonBrowser类检索目标网站所有的链接吧。

**用Beautiful Soup解析Href链接**

为了从目标网站解析链接，我们有两个选择：(1)利用正则表达式来搜索和替换HTML代码。(2)使用强大的第三方库BeautifulSoup，可以在下面网站下载安装：http://www.crummy.com/software/BeautifulSoup/。BeautifulSoup的创造者构建了这个极好的库来处理和解析HTML代码和XML。首先，我们看看怎样使用两种方法找到链接，然后解释为什么大多数情况下BeautifulSoup是很好的选择。

# coding=UTF-8

**from** anonBrowser **import** \*  
**from** BeautifulSoup **import** BeautifulSoup  
**import** optparse  
**import** re  
  
**def printLinks**(url):  
 ab = anonBrowser()  
 ab.anonymize()  
 page = ab.open(url)  
 html = page.read()  
 **try**:  
 **print** '[+] Printing Links From Regex.'  
 link\_finder = re.compile('href="(.\*?)"')  
 links = link\_finder.findall(html)  
 **for** link **in** links:  
 **print** link  
 **except**:  
 **pass  
 try**:  
 **print** '\n[+] Printing Links From BeautifulSoup.'  
 soup = BeautifulSoup(html)  
 links = soup.findAll(name='a')  
 **for** link **in** links:  
 **if** link.has\_key('href'):  
 **print** link['href']  
 **except**:  
 **pass  
def main**():  
 parser = optparse.OptionParser('usage%prog -u <target url>')  
 parser.add\_option('-u', dest='tgtURL', type='string', help='specify target url')  
 (options, args) = parser.parse\_args()  
 url = options.tgtURL  
 **if** url == None:  
 **print** parser.usage  
 exit(0)  
 **else**:  
 printLinks(url)  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

运行我们的脚本，让我们来解析来自流行网站的链接，我们的脚本产生链接的结果通过正则表达式和BeautifulSoup解析。

**recon:# python linkParser.py -uhttp://www.hampsterdance.com/**

**[+] Printing Links From Regex.**

**styles.css**

**http://Kunaki.com/Sales.asp?PID=PX00ZBMUHD**

**http://Kunaki.com/Sales.asp?PID=PX00ZBMUHD**

**Freshhampstertracks.htm**

**freshhampstertracks.htm**

**freshhampstertracks.htm**

**http://twitter.com/hampsterrific**

**http://twitter.com/hampsterrific**

**https://app.expressemailmarketing.com/Survey.aspx?SFID=32244**

**funnfree.htm**

**https://app.expressemailmarketing.com/Survey.aspx?SFID=32244**

**https://app.expressemailmarketing.com/Survey.aspx?SFID=32244**

**meetngreet.htm**

**http://www.asburyarts.com**

**index.htm**

**meetngreet.htm**

**musicmerch.htm**

**funnfree.htm**

**freshhampstertracks.htm**

**hampsterclassics.htm**

**http://www.statcounter.com/joomla/**

**[+] Printing Links From BeautifulSoup.**

**http://Kunaki.com/Sales.asp?PID=PX00ZBMUHD**

**http://Kunaki.com/Sales.asp?PID=PX00ZBMUHD**

**freshhampstertracks.htm**

**freshhampstertracks.htm**

**freshhampstertracks.htm**

**http://twitter.com/hampsterrific**

**http://twitter.com/hampsterrific**

**https://app.expressemailmarketing.com/Survey.aspx?SFID=32244**

**funnfree.htm**

**https://app.expressemailmarketing.com/Survey.aspx?SFID=32244**

**https://app.expressemailmarketing.com/Survey.aspx?SFID=32244**

**meetngreet.htm**

**http://www.asburyarts.com**

**http://www.statcounter.com/joomla/**

乍一看两个似乎差不多。然而，使用正则表达式和BeautifulSoup产生了不同的结果，与一个特定的数据块相关联的标签变化不大，造成程序更加顽固的是网站管理员的念头。比如，我们的正则表达式包含CSS作为一个link，显然，这不是一个链接，但他被正则表达式匹配了。BeautifulSoup解析时知道忽略它，不包含。

**用Beautiful Soup下载图片**

除了网页上面的链接，它上面的图片可能会有用。在第三章，我们展示了如何从图像中提取元数据。再一次，BeautifulSoup成为了关键，允许在任何HTML中搜索’img‘标签。浏览器对象下载图片保存在本地硬盘，代码的变化只是将链接变为图像。随着这些变化，我们基本的检索器已经变得足够强大到找到网页的链接和下载图像。

# coding=UTF-8  
**from** anonBrowser **import** \*  
**from** BeautifulSoup **import** BeautifulSoup  
**import** os  
**import** optparse  
  
**def mirrorImages**(url, dir):  
 ab = anonBrowser()  
 ab.anonymize()  
 html = ab.open(url)  
 soup = BeautifulSoup(html)  
 image\_tags = soup.findAll('img')  
 **for** image **in** image\_tags:  
 filename = image['src'].lstrip('http://')  
 filename = os.path.join(dir, filename.replace('/', '\_'))  
 **print**('[+] Saving ' + str(filename))  
 data = ab.open(image['src']).read()  
 ab.back()  
 save = open(filename, 'wb')  
 save.write(data)  
 save.close()  
  
**def main**():  
 parser = optparse.OptionParser('usage%prog -u <target url> -d <destination directory>')  
 parser.add\_option('-u', dest='tgtURL', type='string', help='specify target url')  
 parser.add\_option('-d', dest='dir', type='string', help='specify destination directory')  
 (options, args) = parser.parse\_args()  
 url = options.tgtURL  
 dir = options.dir  
 **if** url == None **or** dir == None:  
 **print** parser.usage  
 exit(0)  
 **else**:  
 **try**:  
 mirrorImages(url, dir)  
 **except** Exception, e:  
 **print**('[-] Error Mirroring Images.')  
 **print**('[-] ' + str(e))  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

运行这个脚本，我们看到它成功的下载了网站的所有图像。

**econ:∼# python imageMirror.py -u http://xkcd.com -d /tmp/**

**[+] Saving /tmp/imgs.xkcd.com\_static\_terrible\_small\_logo.png**

**[+] Saving /tmp/imgs.xkcd.com\_comics\_moon\_landing.png**

**[+] Saving /tmp/imgs.xkcd.com\_s\_a899e84.jpg**

**研究，调查，发现**

在大多数现代社会工程学的尝试中，攻击者的目标从公司或者企业开始。对于Stuxnet的肇事者，是一个有权限进入SCADA系统的伊朗人。极光行动背后的人是通过调查公司的人员而获取对重要地点的访问权的。让我们假设，我们有一个有趣的公司并知道背后一个主要人物，一个通常的攻击者可能会有比这个更少的信息。攻击者往往只有攻击者更宏观的知识，他们需要利用互联网和其他资源深入了解个人。从Oracle，Google等所有的，我们利用接下来的一系列的脚本。

**用Python和Google API交互**

想象一下，一个朋友问你一个隐晦的问题，他们错误的以为你知道些什么。你怎么回答？Google一下。所以，我们如何了解目标公司的更多信息了？好的，答案再次是Google。Google提供了应用程序接口API允许程序员进行查询并得到结果，而不必尝试破解正常的Google界面。目前有两套API，老旧的API和API，这些需要开发者密钥。要求独一无二的开发者密钥让匿名变得不可能，一些我们以努力获得成功的脚本将不能用。幸运的是老旧的版本任然允许一天之中进行一系列的查询，大约每天30次搜索结果。用于收集信息的话30次结果足够了解一个组织网站的信息了。我们将建立我们的查询功能，返回攻击者感兴趣的信息。

**import urllib**

**from anonBrowser import \***

**def google(search\_term):**

**ab = anonBrowser()**

**search\_term = urllib.quote\_plus(search\_term)**

**response = ab.open('http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=' + search\_term)**

**print(response.read())**

**google('Boondock Saint')**

从Google返回的内容和下面的类似。

**{"responseData": {"results":[{"GsearchResultClass":"GwebSearch",**

**"unescapedUrl":"http://www.boondocksaints.com/","url":"http://**

**www.boondocksaints.com/","visibleUrl":"www.boondocksaints.**

**com","cacheUrl":"http://www.google.com/search?q\**

**u003dcache:J3XW0wgXgn4J:www.boondocksaints.com","title":"The \**

**u003cb\u003eBoondock Saints\u003c/b\u003e","titleNoFormatting":"The**

**Boondock**

**<..SNIPPED..>**

**\u003cb\u003e...\u003c/b\u003e"}],"cursor":{"resultCount":"62,800",**

**"pages":[{"start":"0","label":1},{"start":"4","label":2},{"start**

**":"8","label":3},{"start":"12","label":4},{"start":"16","label":**

**5},{"start":"20","label":6},{"start":"24","label":7},{"start":"2**

**8","label":8}],"estimatedResultCount":"62800","currentPageIndex"**

**:0,"moreResultsUrl":"http://www.google.com/search?oe\u003dutf8\**

**u0026ie\u003dutf8\u0026source\u003duds\u0026start\u003d0\u0026hl\**

**u003den\u0026q\u003dBoondock+Saint","searchResultTime":"0.16"}},**

**"responseDetails": null, "responseStatus": 200}**

quote\_plus()函数是这个脚本中的新的代码块。URL编码是指非字母数字的字符被转换然后发送到服务器。虽然不是完美的URL编码，但是适合我们的目的。最后打印Google的响应显示：一个长字符串的括号和引号。如果你仔细观察，会发现响应的内容看起来很像字典。这些响应是json格式的，和字典非常相似，不出所料，Python有库可以构建和处理json字符串。让我们添加这个功能重新审视这个响应。

**import urllib, json**

**from anonBrowser import \***

**def google(search\_term):**

**ab = anonBrowser()**

**search\_term = urllib.quote\_plus(search\_term)**

**response = ab.open('http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=' + search\_term)**

**objects = json.load(response**

**print(response.read())**

**google('Boondock Saint')**

当我们打印对象时，看起来非常像第一次函数的响应.json库加载响应到一个字典，让这些字段更容易理解，而不需要手动的解析字符串。

**{u'responseData': {u'cursor': {u'moreResultsUrl': u'http://www.google.**

**com/search?oe=utf8&ie=utf8&source=uds&start=0&hl=en&q=Boondock**

**+Saint', u'estimatedResultCount': u'62800', u'searchResultTime':**

**u'0.16', u'resultCount': u'62,800', u'pages': [{u'start': u'0',**

**u'label': 1}, {u'start': u'4', u'label': 2}, {u'start': u'8',**

**u'label': 3}, {u'start': u'12', u'label': 4}, {u'start': u'16',**

**u'label': 5}, {u'start': u'20', u'label': 6}, {u'start': u'24',**

**u'label': 7}, {u'start': u'28', u..SNIPPED..>**

**Saints</b> - Wikipedia, the free encyclopedia', u'url': u'http://**

**en.wikipedia.org/wiki/The\_Boondock\_Saints', u'cacheUrl': u'http://**

**www.google.com/search?q=cache:BKaGPxznRLYJ:en.wikipedia.org',**

**u'unescapedUrl': u'http://en.wikipedia.org/wiki/The\_Boondock\_**

**Saints', u'content': u'The <b>Boondock Saints</b> is a 1999 American**

**action film written and directed by Troy Duffy. The film stars Sean**

**Patrick Flanery and Norman Reedus as Irish fraternal <b>...</b>'}]},**

**u'responseDetails': None, u'responseStatus': 200}**

现在我们可以考虑在一个给定的Google搜索的结果里什么事是重要的。显然，页面返回的链接很重要。此外，页面的标题和Google用的小的文本断对理解链接指向哪里也很重要。为了组织这些结果，我们创建了一个类来保存结果。这将是访问不同的信息更容易。

# coding=UTF-8  
**import** json  
**import** urllib  
**import** optparse  
**from** anonBrowser **import** \*  
  
**class Google\_Result**:  
 **def** \_\_init\_\_(self,title,text,url):  
 self.title = title  
 self.text = text  
 self.url = url  
 **def** \_\_repr\_\_(self):  
 **return** self.title  
  
**def google**(search\_term):  
 ab = anonBrowser()  
 search\_term = urllib.quote\_plus(search\_term)  
 response = ab.open('http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=' + search\_term)  
 objects = json.load(response)  
 results = []  
 **for** result **in** objects['responseData']['results']:  
 url = result['url']  
 title = result['titleNoFormatting']  
 text = result['content']  
 new\_gr = Google\_Result(title, text, url)  
 results.append(new\_gr)  
 **return** result  
  
**def main**():  
 parser = optparse.OptionParser('usage%prog -k <keywords>')  
 parser.add\_option('-k', dest='keyword', type='string', help='specify google keyword')  
 (options, args) = parser.parse\_args()  
 keyword = options.keyword  
 **if** options.keyword == None:  
 **print**(parser.usage)  
 exit(0)  
 **else**:  
 results = google(keyword)  
 **print**(results)  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

这种更简洁的呈现数据的方式产生以下输出：

**recon:∼# python anonGoogle.py -k 'Boondock Saint'**

**[The Boondock Saints, The Boondock Saints (1999) - IMDb, The Boondock**

**Saints II: All Saints Day (2009) - IMDb, The Boondock Saints -**

**Wikipedia, the free encyclopedia]**

**用Python解析Tweets**

在这一点上，我们的脚本已经自动的收集了一些我们的目标的信息。在我们的下一系列的步骤中，我们将撤离域和组织，开始在网上寻找可用的个人信息。

像Google，Twitter给开发者提供了API，该文档在https://dev.twitter.com/docs，非常深入，提供了更多特点的访问，但在本程序中用不到。

让我们探究以下如何从Twitter检索数据。具体来说，我们要转发美国爱国者黑客th3j35t3r的微博，他把Boondock Saint作为自己的昵称。我们将构建reconPerson()类然后输入th3j35t3r作为Twitter的搜索关键字。

# coding=UTF-8  
  
**import** json  
**import** urllib  
**from** anonBrowser **import** \*  
  
**class reconPerson**:  
 **def** \_\_init\_\_(self,first\_name,last\_name, job='',social\_media={}):  
 self.first\_name = first\_name  
 self.last\_name = last\_name  
 self.job = job  
 self.social\_media = social\_media  
 **def** \_\_repr\_\_(self):  
 **return** self.first\_name + ' ' + self.last\_name + ' has job ' + self.job  
 **def get\_social**(self, media\_name):  
 **if** self.social\_media.has\_key(media\_name):  
 **return** self.social\_media[media\_name]  
 **return** None  
 **def query\_twitter**(self, query):  
 query = urllib.quote\_plus(query)  
 results = []  
 browser = anonBrowser()  
 response = browser.open('http://search.twitter.com/search.json?q='+ query)  
 json\_objects = json.load(response)  
 **for** result **in** json\_objects['results']:  
 new\_result = {}  
 new\_result['from\_user'] = result['from\_user\_name']  
 new\_result['geo'] = result['geo']  
 new\_result['tweet'] = result['text']  
 results.append(new\_result)  
 **return** results  
ap = reconPerson('Boondock', 'Saint')  
**print** ap.query\_twitter('from:th3j35t3r since:2010-01-01 include:retweets')

当进一步的继续检索Twitter，我们已经看到了打来那个的信息，这可能对爱国者黑客有用。他正在和一些黑客团体UGNazi的支持者起冲突。好奇心驱使我们想知道为什么会变成这样。

**recon:∼# python twitterRecon.py**

**[{'tweet': u'RT @XNineDesigns: @th3j35t3r Do NOT give up. You are**

**the bastion so many of us need. Stay Frosty!!!!!!!!', 'geo':**

**None, 'from\_user': u'p\u01ddz\u0131uod\u0250\u01dd\u028d \u029e\**

**u0254opuooq'}, {'tweet': u'RT @droogie1xp: "Do you expect me to**

**talk?" - #UGNazi "No #UGNazi I expect you to die." @th3j35t3r**

**#ticktock', 'geo': None, 'from\_user': u'p\u01ddz\u0131uod\u0250\**

**u01dd\u028d \u029e\u0254opuooq'}, {'tweet': u'RT @Tehvar: @th3j35t3r**

**my thesis paper for my masters will now be focused on supporting the**

**#wwp, while I can not donate money I can give intelligence.'**

**<..SNIPPED..>**

我希望，你看到这个代码的时候在想“我现在知道该怎么做了！”确实是这样，从互联网上检索一些特定模式的信息之后。显然，使用Twitter的结果没有用，使用他们寻找目标的信息。当谈论获取个人信息时社交平台是一个金矿。个人的生日，家乡甚至家庭地址，电话号码等隐秘的信息都会被给予不怀好意的人。人们往往没有意识到这个问题，使用社交网站是不安全的习惯。让我们进一步的探究从Twitter的提交里面提取位置数据。

**获取Twitter的位置数据**

很多Twitter用户遵守一个不成文的规定，当有创作时就与全世界分享。一般来说，计算公式是：【其他Twitter用户的消息是针对】+【文本的消息加上段连接】+【Hash标签】。其他的信息可能也包括，但是不在消息体内，就像图像或者位置。然而，退后一步，以攻击者的眼光审视一下这个公式，对于恶意用户这个公式变成了：【用户感兴趣的人，增加某人真正交流的机会】+【某人感兴趣的链接或者主题，他们会对这个主题的消息很感兴趣】+【某人可能会对这个主题有更多的了解】。图片或者地理标签不在有用或者是朋友的有趣的花边新闻。他们会成为配置中的额外的细节，例如某人经常去哪吃早餐。虽然这可能是一个偏执的观点，我们将自动化的收集从Twitter检索的每一条信息。

# coding=UTF-8  
  
**import** json  
**import** urllib  
**import** optparse  
**from** anonBrowser **import** \*  
  
**def get\_tweets**(handle):  
 query = urllib.quote\_plus('from:' + handle+ ' since:2009-01-01 include:retweets')  
 tweets = []  
 browser = anonBrowser()  
 browser.anonymize()  
 response = browser.open('http://search.twitter.com/search.json?q='+ query)  
 json\_objects = json.load(response)  
 **for** result **in** json\_objects['results']:  
 new\_result = {}  
 new\_result['from\_user'] = result['from\_user\_name']  
 new\_result['geo'] = result['geo']  
 new\_result['tweet'] = result['text']  
 tweets.append(new\_result)  
 **return** tweets  
  
**def load\_cities**(cityFile):  
 cities = []  
 **for** line **in** open(cityFile).readlines():  
 city=line.strip('\n').strip('\r').lower()  
 cities.append(city)  
 **return** cities  
  
**def twitter\_locate**(tweets,cities):  
 locations = []  
 locCnt = 0  
 cityCnt = 0  
 tweetsText = ""  
 **for** tweet **in** tweets:  
 **if** tweet['geo'] != None:  
 locations.append(tweet['geo'])  
 locCnt += 1  
 tweetsText += tweet['tweet'].lower()  
 **for** city **in** cities:  
 **if** city **in** tweetsText:  
 locations.append(city)  
 cityCnt+=1  
 **print**("[+] Found "+str(locCnt)+" locations via Twitter API and "+str(cityCnt)+" locations from text search.")  
 **return** locations  
  
**def main**():  
 parser = optparse.OptionParser('usage%prog -u <twitter handle> [-c <list of cities>]')  
 parser.add\_option('-u', dest='handle', type='string', help='specify twitter handle')  
 parser.add\_option('-c', dest='cityFile', type='string', help='specify file containing cities to search')  
 (options, args) = parser.parse\_args()  
 handle = options.handle  
 cityFile = options.cityFile  
 **if** (handle==None):  
 **print** parser.usage  
 exit(0)  
 cities = []  
 **if** (cityFile!=None):  
 cities = load\_cities(cityFile)  
 tweets = get\_tweets(handle)  
 locations = twitter\_locate(tweets,cities)  
 **print**("[+] Locations: "+str(locations))  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

我了测试我们的脚本，我们建立了城市的列表。

**recon:∼# cat mlb-cities.txt | more**

**baltimore**

**boston**

**chicago**

**cleveland**

**detroit**

**<..SNIPPED..>**

**recon:∼# python twitterGeo.py -u redsox -c mlb-cities.txt**

**[+] Found 0 locations via Twitter API and 1 locations from text**

**search.**

**[+] Locations: ['toronto']**

**recon:∼# python twitterGeo.py -u nationals -c mlb- cities.txt**

**[+] Found 0 locations via Twitter API and 1 locations from text**

**search.**

**[+] Locations: ['denver']**

**用正则表达式解析Twitter的关注**

接下来我们将收集目标的兴趣，这包括其他用户或者是网路内容。任何时候网站都提供了能力知道用户对什么感兴趣，跳过去，这些数据将成为成功的社会工程攻击的基础。如我们前面讨论的，Twitter的兴趣点包含任何链接，Hash标签或者是其他用户提到的内容。用正则表达式找到这些很容易。

# coding=UTF-8  
**import** json  
**import** re  
**import** urllib  
**import** urllib2  
**import** optparse  
**from** anonBrowser **import** \*  
  
**def get\_tweets**(handle):  
 query = urllib.quote\_plus('from:' + handle+ ' since:2009-01-01 include:retweets')  
 tweets = []  
 browser = anonBrowser()  
 browser.anonymize()  
 response = browser.open('http://search.twitter.com/search.json?q=' + query)  
 json\_objects = json.load(response)  
 **for** result **in** json\_objects['results']:  
 new\_result = {}  
 new\_result['from\_user'] = result['from\_user\_name']  
 new\_result['geo'] = result['geo']  
 new\_result['tweet'] = result['text']  
 tweets.append(new\_result)  
 **return** tweets  
  
**def find\_interests**(tweets):  
 interests = {}  
 interests['links'] = []  
 interests['users'] = []  
 interests['hashtags'] = []  
 **for** tweet **in** tweets:  
 text = tweet['tweet']  
 links = re.compile('(http.\*?)\Z|(http.\*?) ').findall(text)  
 **for** link **in** links:  
 **if** link[0]:  
 link = link[0]  
 **elif** link[1]:  
 link = link[1]  
 **else**:  
 **continue  
 try**:  
 response = urllib2.urlopen(link)  
 full\_link = response.url  
 interests['links'].append(full\_link)  
 **except**:  
 **pass** interests['users'] += re.compile('(@\w+)').findall(text)  
 interests['hashtags'] +=re.compile('(#\w+)').findall(text)  
 interests['users'].sort()  
 interests['hashtags'].sort()  
 interests['links'].sort()  
 **return** interests  
  
**def main**():  
 parser = optparse.OptionParser('usage%prog -u <twitter handle>')  
 parser.add\_option('-u', dest='handle', type='sring', help='specify twitter handle')  
 (options, args) = parser.parse\_args()  
 handle = options.handle  
 **if** handle == None:  
 **print**(parser.usage)  
 exit(0)  
 tweets = get\_tweets(handle)  
 interests = find\_interests(tweets)  
 **print**('\n[+] Links.')  
 **for** link **in** set(interests['links']):  
 **print**(' [+] ' + str(link))  
 **print**('\n[+] Users.')  
 **for** user **in** set(interests['users']):  
 **print**(' [+] ' + str(user))  
 **print**('\n[+] HashTags.')  
 **for** hashtag **in** set(interests['hashtags']):  
 **print**('\n[+] ' + str(hashtag))  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

运行我们的兴趣分析脚本，我们看到它解析出针对目标的链接，用户名，Hash标签。请注意，它返回了一个Youtube的视频，一些用户名和当前即将到来的比赛的Hash标签。好奇心再一次让我们知道该怎么做。

**recon:∼# python twitterInterests.py -u sonnench**

**[+] Links.**

**[+]http://www.youtube.com/watch?v=K-BIuZtlC7k&feature=plcp**

**[+] Users.**

**[+] @tomasseeger**

**[+] @sonnench**

**[+] @Benaskren**

**[+] @AirFrayer**

**[+] @NEXERSYS**

**[+] HashTags.**

**[+] #UFC148**

这里使用正则表达式不是寻找信息的合适方法。正则表达式抓住包含链接的文本将会错过某一特定的URL，因为用正则表达式很难匹配所有格式的URL。然而，对我们而言正则表达式99%的情况下会工作。此外，使用urllib2库里的函数打开链接而不是我们的匿名类。

再一次，我们将使用使用一个字典排序信息到一个更加易于管理的数据结构中，所以我们不需要创造一个类。由于Twitter字符的限制，许多URL使用某种服务把URL变短了。这些链接并不非常有用，因为他们能指向任何地方。为了扩展他们，我们将使用urllib2打开。脚本打开页面后，urllib能取回整个URL。其他用户和Hash标签将使用类似的正则表达式来检索。并返回给主要的方法twitter()。位置和关注最后会被调用得到。

可以做其他事情扩展处理Twitter的能力。互联网上有无限的资源，无数中分析数据的方法要求扩大自动化收集信息程序的能力。

将我们整个系列的侦查包装在一起，我们做了一个类来检索位置，兴趣和Twitter。这些在下一节中将会看到用处的。

# coding=UTF-8  
**import** urllib  
**from** anonBrowser **import** \*  
**import** json  
**import** re  
**import** urllib2  
  
**class reconPerson**:  
 **def** \_\_init\_\_(self, handle):  
 self.handle = handle  
 self.tweets = self.get\_tweets()  
 **def get\_tweets**(self):  
 query = urllib.quote\_plus('from:' + self.handle+' since:2009-01-01 include:retweets')  
 tweets = []  
 browser = anonBrowser()  
 browser.anonymize()  
 response = browser.open('http://search.twitter.com/search.json?q=' + query)  
 json\_objects = json.load(response)  
 **for** result **in** json\_objects['results']:  
 new\_result = {}  
 new\_result['from\_user'] = result['from\_user\_name']  
 new\_result['geo'] = result['geo']  
 new\_result['tweet'] = result['text']  
 tweets.append(new\_result)  
 **return** tweets  
 **def find\_interests**(self):  
 interests = {}  
 interests['links'] = []  
 interests['users'] = []  
 interests['hashtags'] = []  
 **for** tweet **in** self.tweets:  
 text = tweet['tweet']  
 links = re.compile('(http.\*?)\Z|(http.\*?) ').findall(text)  
 **for** link **in** links:  
 **if** link[0]:  
 link = link[0]  
 **elif** link[1]:  
 link = link[1]  
 **else**:**continue  
 try**:  
 response = urllib2.urlopen(link)  
 full\_link = response.url  
 interests['links'].append(full\_link)  
 **except**:  
 **pass** interests['users'] +=re.compile('(@\w+)').findall(text)  
 interests['hashtags'] +=re.compile('(#\w+)').findall(text)  
 interests['users'].sort()  
 interests['hashtags'].sort()  
 interests['links'].sort()  
 **return** interests  
 **def twitter\_locate**(self, cityFile):  
 cities = []  
 **if** cityFile != None:  
 **for** line **in** open(cityFile).readlines():  
 city = line.strip('\n').strip('\r').lower()  
 cities.append(city)  
 locations = []  
 locCnt = 0  
 cityCnt = 0  
 tweetsText = ''  
 **for** tweet **in** self.tweets:  
 **if** tweet['geo'] != None:  
 locations.append(tweet['geo'])  
 locCnt += 1  
 tweetsText += tweet['tweet'].lower()  
 **for** city **in** cities:  
 **if** city **in** tweetsText:  
 locations.append(city)  
 cityCnt += 1  
 **return** locations

**匿名邮件**

越来约频繁的，网站要求用户创建并登陆账户，如果他们想访问网站的最佳资源的话。这显然会出现一个问题，对于传统的浏览用户，浏览互联网的浏览器是不同的，登陆显然破坏了匿名浏览，登陆后的任何行为取决于账户。大多数网站只需要一个有效的邮件地址并不检查其他的私人信息。像雅虎，Google提供的邮箱服务是免费的，很容易申请。然而，他们有一些服务和条款你必须接受和理解。

一个很好的选择是使用一个一次性的邮箱账户获得一个永久性的邮箱。十分钟邮箱http://10minutemail.com/10MinuteMail/index.html提供一个一次性的邮箱。攻击者可以利用很难追查的电子邮件创建不依赖他们的账户。大多数网站最起码的使用条款是不允许收集其他用户的信息。虽然实际的攻击者不遵守这些规定，对账户使用这种技术完全可以做到。记住，虽然这一技术可以被用来保护你，你应该采取行动，确保你的账户的行为安全。

**大规模的社会工程**

在这一点上，我们已经收集了目标的大量的有价值的信息。利用这些信息自动的生成邮件是一个复杂的事，尤其是添加了足够的细节让他变得可行。在这一点上一个选项可能会让目前的程序停止：这也允许攻击者利用所有的有用的信息构造一个邮件。然而，手动发邮件给一个大组织的每一个人是不可行的。Python的能力允许我们的这个过程自动化并快速获得结果。为了这个目的，我们将使用收集到的信息建立一个非常简单的邮件并发送给目标。

**使用Smtplib发送邮件给目标**

发送电子邮件的过程中通常需要开发客户的选择，点击新建，然后点击发送。在这背后，客户端连接到服务器，可能记录了日志，交换信息的发送人，收件人和其他必要的资料。Python的Smtplib库将在程序中处理这些过程。我们将通过建立一个Python的电子邮件客户端发送我们的恶意邮件给目标。这个客户端很基本，但让我们在程序中发送邮件很简单。我们这次的目的，我们将使用Google的邮件SMTP服务，你需要创建一个Google邮件账户，在我们的脚本中使用，或者使用自己的SMTP服务器。

**import smtplib**

**from email.mime.text import MIMEText**

**def sendMail(user,pwd,to,subject,text):**

**msg = MIMEText(text)**

**msg['From'] = user**

**msg['To'] = to**

**msg['Subject'] = subject**

**try:**

**smtpServer = smtplib.SMTP('smtp.gmail.com', 587)**

**print("[+] Connecting To Mail Server.")**

**smtpServer.ehlo()**

**print("[+] Starting Encrypted Session.")**

**smtpServer.starttls()**

**smtpServer.ehlo()**

**print("[+] Logging Into Mail Server.")**

**smtpServer.login(user, pwd)**

**print("[+] Sending Mail.")**

**smtpServer.sendmail(user, to, msg.as\_string())**

**smtpServer.close()**

**print("[+] Mail Sent Successfully.")**

**except:**

**print("[-] Sending Mail Failed.")**

**user = 'username'**

**pwd = 'password'**

**sendMail(user, pwd, 'target@tgt.tgt', 'Re: Important', 'Test Message')**

运行脚本，检查我们的邮箱，我们可以看到成功的发送了邮件。

**recon:# python sendMail.py**

**[+] Connecting To Mail Server.**

**[+] Starting Encrypted Session.**

**[+] Logging Into Mail Server.**

**[+] Sending Mail.**

**[+] Mail Sent Successfully.**

提供了有效的邮件服务器和参数，客户端将正确的发送邮件给目标。有许多的邮件服务器，然而，不带开转发，我们只能发送邮件到就特定的地址。在本地的邮件服务中设置转发，或者在互联网上打开转发。将能发送邮件从任何地址到任何地址，发送方的地址甚至可以是无效的。垃圾邮件的发送者使用相同的技术发送邮件来自Potus@whitehouse.gov：他们简单的伪造了发送地址。很少有人会打开来自可疑地址邮件，我们可以伪造邮件的发送地址是关键。使用客户端打开，打开转发功能，是攻击者从一个看起来值得信奈的地址发送邮件，增加用户点开邮件的可能性。

**用Smtplib进行鱼叉式网络钓鱼**

将我们所有的研究放在一起是我们最后的阶段。在这里，我们的脚本创建了一个看起来像目标朋友发来的电子邮件，目标发现一些有趣的事情，邮件看起来是真人写的。大量的研究投入到帮助电脑的的交流看起来更像人，各种各样的方法任然在完善。为了减少这种可能性，我们将创建一个包含攻击荷载的的简单的信息邮件。程序的几个部分之一将涉及选择包含这条信息。我们的程序将按数据随机的选择。采取地步骤是：选择虚假的发件人地址，制作一个主题，创建一个信息，然后发送电子邮件。幸运的是创建发送人和主题是相当的简单。

代码的if语句仔细的处理和如何将短信息连接在一起是很重要的问题。当处理数量巨大的可能性时，在我们的侦查中将使用更多情况的代码，每一个可能性会被分为独立的函数。每一个方法将以特定的的方法承担一块的开始和结束，然后独立与其他代码的操作。这样，收集到某人的信息就越多，唯一改变的是方法。最后一步是通过我们的电子邮件客户端，相信它的人愚蠢的做剩下的活。这个过程的没一部分在这一章中都讨论过，这是任何被用来获取权限的钓鱼网站的产物。在我们的例子中，我们简单的发送一个名不副实的链接，有效荷载可以是附件或者是诈骗网站，或者任何其他的攻击方法。这个过程将对每一个成员重复，只要一个人上当攻击者就能获取权限。

我们特定的脚本将攻击一个用户基于他公开的信息。基于他的地点，用户，Hash标签，链接，脚本将创建一个附带恶意链接的邮件等待用户点击。

# coding=UTF-8  
**import** smtplib  
**import** optparse  
**from** email.mime.text **import** MIMEText  
**from** twitterClass **import** \*  
**from** random **import** choice  
  
**def sendMail**(user,pwd,to,subject,text):  
 msg = MIMEText(text)  
 msg['From'] = user  
 msg['To'] = to  
 msg['Subject'] = subject  
 **try**:  
 smtpServer = smtplib.SMTP('smtp.gmail.com', 587)  
 **print**("[+] Connecting To Mail Server.")  
 smtpServer.ehlo()  
 **print**("[+] Starting Encrypted Session.")  
 smtpServer.starttls()  
 smtpServer.ehlo()  
 **print**("[+] Logging Into Mail Server.")  
 smtpServer.login(user, pwd)  
 **print**("[+] Sending Mail.")  
 smtpServer.sendmail(user, to, msg.as\_string())  
 smtpServer.close()  
 **print**("[+] Mail Sent Successfully.")  
 **except**:  
 **print**("[-] Sending Mail Failed.")  
  
**def main**():  
 parser = optparse.OptionParser('usage%prog -u <twitter target> -t <target email> -l <gmail login> -p <gmail password>')  
 parser.add\_option('-u', dest='handle', type='string', help='specify twitter handle')  
 parser.add\_option('-t', dest='tgt', type='string', help='specify target email')  
 parser.add\_option('-l', dest='user', type='string', help='specify gmail login')  
 parser.add\_option('-p', dest='pwd', type='string', help='specify gmail password')  
 (options, args) = parser.parse\_args()  
 handle = options.handle  
 tgt = options.tgt  
 user = options.user  
 pwd = options.pwd  
 **if** handle == None **or** tgt == None **or** user ==None **or** pwd==None:  
 **print**(parser.usage)  
 exit(0)  
 **print**("[+] Fetching tweets from: "+str(handle))  
 spamTgt = reconPerson(handle)  
 spamTgt.get\_tweets()  
 **print**("[+] Fetching interests from: "+str(handle))  
 interests = spamTgt.find\_interests()  
 **print**("[+] Fetching location information from: "+ str(handle))  
 location = spamTgt.twitter\_locate('mlb-cities.txt')  
 spamMsg = "Dear "+tgt+","  
 **if** (location!=None):  
 randLoc=choice(location)  
 spamMsg += " Its me from "+randLoc+"."  
 **if** (interests['users']!=None):  
 randUser=choice(interests['users'])  
 spamMsg += " "+randUser+" said to say hello."  
 **if** (interests['hashtags']!=None):  
 randHash=choice(interests['hashtags'])  
 spamMsg += " Did you see all the fuss about "+ randHash+"?"  
 **if** (interests['links']!=None):  
 randLink=choice(interests['links'])  
 spamMsg += " I really liked your link to: "+randLink+"."  
 spamMsg += " Check out my link to http://evil.tgt/malware"  
 **print**("[+] Sending Msg: "+spamMsg)  
 sendMail(user, pwd, tgt, 'Re: Important', spamMsg)  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

测试我们的脚本，我们可以获得一些关于Boston Red Sox的信息，从他的Twitter账户上，为了发送一个恶意的垃圾邮件。

**recon# python sendSpam.py -u redsox -t target@tgt -l username -p password**

**[+] Fetching tweets from: redsox**

**[+] Fetching interests from: redsox**

**[+] Fetching location information from: redsox**

**[+] Sending Msg: Dear redsox, Its me from toronto. @davidortiz said**

**to say hello. Did you see all the fuss about #SoxAllStars? I really**

**liked your link to:http://mlb.mlb.com. Check out my link to http://**

**evil.tgt/malware**

**[+] Connecting To Mail Server.**

**[+] Starting Encrypted Session.**

**[+] Logging Into Mail Server.**

**[+] Sending Mail.**

**[+] Mail Sent Successfully.**

**本章总结**

虽然这个方法不是用于另一个人或者组织，但它对认识其可行性和组织的脆弱性很重要。Python和其他脚本语言允许程序员快速的创建一个方法，使用从互联网上找到的广阔的资源，来获取潜在的利益。在我们的代码中，我创建了一个类来模拟浏览器同时增加了匿名访问，检索网站，使用强大的Google，利用Twitter来了解目标的更多信息功能，然后把所有的细节发送一个特殊的电子邮件给目标用户。互联网的连接速度限制了程序，线程的某些函数将大大的减少执行时间。此外，一旦我们学会了如何从数据源中检索信息，对其他网站做同样的信息是很简单的。个人美誉访问和处理互联网上大量的信息的能力，但是强大的Python和它的库允许访问每一个资源的能力远远高于几个熟练的人员。知道这一切，攻击不是你想象中的那么复杂，你的组织是如何的脆弱？什么公开的信息可以被攻击者使用？你会成为一个Python检索信息和恶意邮件的受害者吗？