1. ：法庭调查取证

本章内容：

1. **通过Windows注册表定位**
2. **回收站调查**
3. **审查PDF和DOC文件的元数据**
4. **从Exif元数据中提取GPS坐标**
5. **探究Skype结构**
6. **从火狐的数据库中枚举浏览器结构**
7. **审查移动设备结构**

**最终，你必须忘记技术。你越是进步，教导的也就越少，伟大的路是没有真正的道路的。**

**---Ueshiba Morihei, Kaiso, Founder, Aikido**

**简介：如何解决BTK谋杀案**

BTK谋杀案及其调查取证过程略。

计算机取证调查只需要好的调查员和他的好工具。调查员往往有很多挑剔的问题，但没有工具能解决他们的问题。进入Python。在前几章我们看到，解决复杂的问题只用极少的代码证明了Python编程语言的实力。正如我们将在下面章节中看到的，我们能用极少数的Python代码解决复杂的问题。让我们开始用一些独特的Windows注册表来物理跟踪用户吧。

**你去哪里了？---在注册表中分析无线接入点**

Windows注册表包含了一个存储操作系统配置设置的层次化数据库。随着无线网的出现，Windows注册表存储了与无线连接相关的信息。了解注册表键值的位置和意义可以为我们提供详细的笔记本到过的地理位置。从Windows Vista之后，注册表存储每一个网络信息在**HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\NetworkList\Signatures\Unmanaged**子键值下。从Windows命令提示符，我们可以列出每一个网络显示描述GUID，网络描述，网络名称和网关MAC地址。

**C:\Windows\system32>reg query "HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\**

**Windows NT\CurrentVersion\NetworkList\Signatures\Unmanaged" /s**

**HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows**

**NT\CurrentVersion\NetworkList\Sign**

**atures\Unmanaged\010103000F0000F0080000000F0000F04BCC2360E4B8F7DC8BDAF**

**AB8AE4DAD8**

**62E3960B979A7AD52FA5F70188E103148**

**ProfileGuid REG\_SZ {3B24CE70-AA79-4C9A-B9CC-83F90C2C9C0D}**

**Description REG\_SZ Hooters\_San\_Pedro**

**Source REG\_DWORD 0x8**

**DnsSuffix REG\_SZ <none>**

**FirstNetwork REG\_SZ Public\_Library**

**DefaultGatewayMac REG\_BINARY 00115024687F0000**

**使用WinReg读取Windows注册表**

注册表存储的网关MAC地址作为REG\_BINARY类型。在前面的例子中，16进制\x00\x11\x50\x24\x68\x7F\x00\x00表示的实际地址为00:11:50:24:68:7F。我们将写一个快速的函数将REG\_BINARY的值转换为实际的MAC地址。在后面我们将会看到无线网络的MAC地址是有用的。

**def val2addr(val):**

**addr = ""**

**for ch in val:**

**addr += ("%02x "% ord(ch))**

**addr = addr.strip(" ").replace(" ",":")[0:17]**

**return addr**

现在，让我们来编写一个函数从Windows注册表键值中获取每一个列出来的网络的网络名称和MAC地址。为此，我们将利用\_winreg模块，Windows版的Python默认安装的模块。连接到注册表后，我们可以使用OpenKey()函数打开键，并循环获取键下面的网络描述。对于每一个描述，包含下面子键：ProfileGuid, Description, Source, DnsSuffix, FirstNetwork, DefaultGatewayMac。网络名称和网关MAC地址在注册表键列表中的第四个和第五个。现在我们可以枚举每一个键，并在屏幕上面打印出来。

把所有的组合在一起，现在我们有一个脚本将打印存储在注册表中的先前连接的无线网络的信息。

# coding=UTF-8  
  
**import** \_winreg  
  
**def val2addr**(val):  
 addr = ""  
 **for** ch **in** val:  
 addr += ("%02x "% ord(ch))  
 addr = addr.strip(" ").replace(" ",":")[0:17]  
 **return** addr  
  
**def** printNets():  
 net = "SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList\Signatures\Unmanaged"  
 key = \_winreg.OpenKey(\_winreg.HKEY\_LOCAL\_MACHINE, net)  
 **print** '\n[\*] Networks You have Joined.'  
 **for** i **in** range(100):  
 **try**:  
 guid = \_winreg.EnumKey(key, i)  
 netKey = \_winreg.OpenKey(key, str(guid))  
 (n, addr, t) = \_winreg.EnumValue(netKey, 5)  
 (n, name, t) = \_winreg.EnumValue(netKey, 4)  
 macAddr = val2addr(addr)  
 netName = str(name)  
 **print** '[+] ' + netName + ' ' + macAddr  
 \_winreg.CloseKey(netKey)  
 **except**:  
 **break  
  
def main**():  
 printNets()  
**if** \_\_name\_\_ == "\_\_main\_\_":  
 main()

在我们的目标笔记本上运行我们的脚本，我们可以看到先前连接过的无线网络及其MAC地址。当测试脚本时，确保使用管理员权限运行的脚本，否则将无法读取注册表键值。

**C:\Users\investigator\Desktop\python discoverNetworks.py**

**[\*] Networks You have Joined.**

**[+] Hooters\_San\_Pedro, 00:11:50:24:68:7F**

**[+] LAX Airport, 00:30:65:03:e8:c6**

**[+] Senate\_public\_wifi, 00:0b:85:23:23:3e**

**使用Mechanize 将MAC地址提交到Wigle**

然而，脚本不会在次结束。随着获得无线接入点的MAC地址，我们现在开可以打印出无线接入点的物理位置。有相当多的数据库，包括开源的和专有的，包含了大量与无线接入点物理位置相关的信息。专利产品，如手机就是使用这样的地理位置的数据库而没有使用GPS。

SkyHook数据库，可以在http://www.skyhookwireless.com/找到。提供了一个基于WIFI接入点的软件开发工具包。Ian McCracken开发的一个开源项目提供了提供了对这个数据库的访问能力在http://code.google.com/p/maclocate/网站。然而，最近，SkyHook改变了SDK而是使用API密钥来使用数据库。Google也维护这同样大的数据库用于关联无线接入点的MAC地址到物理位置。然而，不久后，不久后，Gorjan Petrovski开发了一个NMAP NSE脚本来和Google的数据库进行交互。Google反对开源代码和他的数据库进行交互。不久之后，由于隐私问题，微软也关闭了类似的WIFI物理位置数据库。

剩下的数据库和开源项目WiGLE.net继续允许用户通过无线接入点的搜索物理位置。注册一个账号之后，用户就能通过一个小的Python脚本和wigle.net进行交互。让我们快速检查如何建立一个脚本与WiGLE.net交互。

使用WiGLE. Net，用户很快就会意识到为了得到WiGLE他必须与第三方的页面进行交互。首先，用户必须打开WiGLE.net的初始化页面在https://wigle.net/网页；然后用户必须登陆到WiGLE在https://wigle.net/页面。最后，用户可以查询特定的无线SSID的MAC地址在https://wigle.net/页面。捕获MAC地址查询请求，我们可以看到在请求无线接入点的GPS地址的HTTP POST请求中tnetid(网络标识符)包含了MAC地址。

**POST /gps/gps/main/confirmquery/ HTTP/1.1**

**Accept-Encoding: identity**

**Content-Length: 33**

**Host: wigle.net**

**User-Agent: AppleWebKit/531.21.10**

**Connection: close**

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

**netid=0A%3A2C%3AEF%3A3D%3A25%3A1B**

**<..SNIPPED..>**

此外，我们看到从页面响应的数据中包含了GPS坐标。字符串maplat=47.25264359&maplon=-87.25624084包含了接入点的经度和纬度。

**<tr class="search"><td>**

**<a href="/gps/gps/Map/onlinemap2/?maplat=47.25264359&maplon=-**

**87.25624084&mapzoom=17&ssid=McDonald's FREE Wifi&netid=0A:2C:EF:3D:**

**25:1B">Get Map</a></td>**

**<td>0A:2C:EF:3D:25:1B</td><td>McDonald's FREE Wifi</td><**

有了这些信息，我们现在足够建立建立一个简单的函数用来返回WiGLE数据库中记录的无线接入点的的经度和纬度。注意，要使用mechanize模块。可以从http://wwwsearch.sourceforge.net/mechanize/网站获得该模块。mechanize允许通过Python进行WEB状态编程，类似于urllib2模块的功能。这就意味着，一旦我们正常的登陆到WiGLE服务，它就会存储和重用我们的验证cookie。

**import mechanize, urllib, re, urlparse**

**def wiglePrint(username, password, netid):**

**browser = mechanize.Browser()**

**browser.open('http://wigle.net')**

**reqData = urllib.urlencode({'credential\_0': username, 'credential\_1': password})**

**browser.open('https://wigle.net/gps/gps/main/login', reqData)**

**params = {}**

**params['netid'] = netid**

**reqParams = urllib.urlencode(params)**

**respURL = 'http://wigle.net/gps/gps/main/confirmquery/'**

**resp = browser.open(respURL, reqParams).read()**

**mapLat = 'N/A'**

**mapLon = 'N/A'**

**rLat = re.findall(r'maplat=.\*\&', resp)**

**if rLat:**

**mapLat = rLat[0].split('&')[0].split('=')[1]**

**rLon = re.findall(r'maplon=.\*\&', resp)**

**if rLon:**

**mapLon = rLon[0].split**

**print('[-] Lat: ' + mapLat + ', Lon: ' + mapLon)**

添加WiGLE MAC地址查询功能到我们原来的脚本。我们现在有能力检查注册表中以前连接过的无线接入点并查询他们的物理位置。

# coding=UTF-8  
**import** optparse  
**import** mechanize  
**import** urllib  
**import** re  
**import** \_winreg  
  
**def val2addr**(val):  
 addr = ""  
 **for** ch **in** val:  
 addr += ("%02x " % ord(ch))  
 addr = addr.strip(" ").replace(" ", ":")[0:17]  
 **return** addr

**def wiglePrint**(username, password, netid):  
 browser = mechanize.Browser()  
 browser.open('http://wigle.net')  
 reqData = urllib.urlencode({'credential\_0': username, 'credential\_1': password})  
 browser.open('https://wigle.net/gps/gps/main/login', reqData)  
 params = {}  
 params['netid'] = netid  
 reqParams = urllib.urlencode(params)  
 respURL = 'http://wigle.net/gps/gps/main/confirmquery/'  
 resp = browser.open(respURL, reqParams).read()  
 mapLat = 'N/A'  
 mapLon = 'N/A'  
 rLat = re.findall(r'maplat=.\*\&', resp)  
 **if** rLat:  
 mapLat = rLat[0].split('&')[0].split('=')[1]  
 rLon = re.findall(r'maplon=.\*\&', resp)  
 **if** rLon:  
 mapLon = rLon[0].split  
 **print**('[-] Lat: ' + mapLat + ', Lon: ' + mapLon)  
  
**def printNets**(username, password):  
 net = "SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList\Signatures\Unmanaged"  
 key = \_winreg.OpenKey(\_winreg.HKEY\_LOCAL\_MACHINE, net)  
 **print** '\n[\*] Networks You have Joined.'  
 **for** i **in** range(100):  
 **try**:  
 guid = \_winreg.EnumKey(key, i)  
 netKey = \_winreg.OpenKey(key, str(guid))  
 (n, addr, t) = \_winreg.EnumValue(netKey, 5)  
 (n, name, t) = \_winreg.EnumValue(netKey, 4)  
 macAddr = val2addr(addr)  
 netName = str(name)  
 **print**('[+] ' + netName + ' ' + macAddr)  
 wiglePrint(username, password, macAddr)  
 \_winreg.CloseKey(netKey)  
 **except**:  
 **break  
  
def main**():  
 parser = optparse.OptionParser("usage%prog -u <wigle username> -p <wigle password>")  
 parser.add\_option('-u', dest='username', type='string', help='specify wigle password')  
 parser.add\_option('-p', dest='password', type='string', help='specify wigle username')  
 (options, args) = parser.parse\_args()  
 username = options.username  
 password = options.password  
 **if** username == None **or** password == None:  
 **print**(parser.usage)  
 exit(0)  
 **else**:  
 printNets(username, password)  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

运行我们的新脚本，我们可以看到先前连接过的无线网络和他们的物理位置。知道了计算机在哪，让我们在下一节检查一下回收站。

**C:\Users\investigator\Desktop\python discoverNetworks.py**

**[\*] Networks You have Joined.**

**[+] Hooters\_San\_Pedro, 00:11:50:24:68:7F**

**[-] Lat: 29.55995369, Lon: -98.48358154**

**[+] LAX Airport, 00:30:65:03:e8:c6**

**[-] Lat: 28.04605293, Lon: -82.60256195**

**[+] Senate\_public\_wifi, 00:0b:85:23:23:3e**

**[-] Lat: 44.95574570, Lon: -93.10277557**

**用Python来恢复回收站中删除的项目**

在微软的操作系统中，回收站作为一个特殊的文件夹包含了已经删除的文件。当用户通过Windows Explorer删除文件时，操作系统会将这个文件移动到这个特殊的文件夹中并标记这文件已删除，但是并不是实际上的删除它们。在Windows 98和更早的系统中用的是FAT文件系统。C:\Recycled\目录保存着回收站目录。支持NTFS的操作系统有Windows NT，2000，和XP存储回收站在C:\Recycler\目录下。Windows Vista和Windows 7系统存储在C:\$Recycle.Bin目录下。

**使用OS模块查找已删除的项目**

为了让我们的脚本不依赖与特定的Windows操作系统，让我们编写一个函数来测试每一个可能的候选目录并返回系统上存在的第一个目录。

**import os**

**def returnDir():**

**dirs = ['C:\\Recycler\\', 'C:\\Recycled\\', 'C:\\$Recycle.Bin\\']**

**for recycleDir in dirs:**

**if os.path.isdir(recycleDir):**

**return recycleDir**

**return None**

在发现回收站目录之后，我们需要检查里面的内容。注意两个子目录，它们都包含字符串S-1-5-21-1275210071-1715567821-725345543-并终止与1005或者500。这个字符串用户的SID，与机器上的用户的账户一一相对应。

**C:\RECYCLER>dir /a**

**Volume in drive C has no label.**

**Volume Serial Number is 882A-6E93**

**Directory of C:\RECYCLER**

**04/12/2011 09:24 AM <DIR> .**

**04/12/2011 09:24 AM <DIR> ..**

**04/12/2011 09:56 AM**

**<DIR> S-1-5-21-1275210071-1715567821-**

**725345543-**

**1005**

**04/12/2011 09:20 AM <DIR> S-1-5-21-1275210071-1715567821-**

**725345543-**

**500**

**0 File(s) 0 bytes**

**4 Dir(s) 30,700,670,976 bytes free**

**用Python将用户的SID关联起来**

我们将使用Windows注册表将SID转化为一个准确的用户名。通过检查Windows注册表键值HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList\<SID>\ProfileImagePath，我们可以看到它返回一个是%SystemDrive%\Documents and Settings\<USERID>。在下图中，我们看到这允许我们将SID为S-1-5-21-1275210071-1715567821-725345543-1005转化为用户名“alex”。

**C:\RECYCLER>reg query "HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\ProfileList\S-1-5-21-1275210071-1715567821-725345543-1005" /v**

**ProfileImagePath**

**! REG.EXE VERSION 3.0**

**HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows**

**NT\CurrentVersion\ProfileList \S-1-5-21-1275210071-1715567821-**

**725345543-1005 ProfileImagePath**

**REG\_EXPAND\_SZ %SystemDrive%\Documents and Settings\alex**

我们想知道回收站里谁删除了什么文件。让我们编写一个小的函数来将每一个SID转化为用户名。当我们恢复回收站中被删除的项目时这将使我们打印更多有用的输出。这个函数将打开注册便检查ProfileImagePath键值，找到其值并从中找到用户名。

**import \_winreg**

**def sid2user(sid):**

**try:**

**key = \_winreg.OpenKey(\_winreg.HKEY\_LOCAL\_MACHINE, "SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList\\" + sid)**

**(value, type) = \_winreg.QueryValueEx(key, 'ProfileImagePath')**

**user = value.split('\\')[-1]**

**return user**

**except:**

**return sid**

最后，我们将所有的代码放在一起生成一个脚本，它将打印已删除但还在回收站中的项目。

# coding=UTF-8  
  
**import** os  
**import** \_winreg  
  
**def returnDir**():  
 dirs = ['C:\\Recycler\\', 'C:\\Recycled\\', 'C:\\$Recycle.Bin\\']  
 **for** recycleDir **in** dirs:  
 **if** os.path.isdir(recycleDir):  
 **return** recycleDir  
 **return** None  
  
**def sid2user**(sid):  
 **try**:  
 key = \_winreg.OpenKey(\_winreg.HKEY\_LOCAL\_MACHINE, "SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList\\" + sid)  
 (value, type) = \_winreg.QueryValueEx(key, 'ProfileImagePath')  
 user = value.split('\\')[-1]  
 **return** user  
 **except**:  
 **return** sid  
  
**def findRecycled**(recycleDir):  
 dirList = os.listdir(recycleDir)  
 **for** sid **in** dirList:  
 files = os.listdir(recycleDir + sid)  
 user = sid2user(sid)  
 **print**('\n[\*] Listing Files For User: ' + str(user))  
 **for** file **in** files:  
 **print**('[+] Found File: ' + str(file))  
  
**def main**():  
 recycledDir = returnDir()  
 findRecycled(recycledDir)  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

在目标机器上运行我们的脚本，我们看到脚本发现了两个用户：Administrator和alex。它列出了回收站中每个用户的文件。在下一节中，我们将审查一个方法，用于检查那些包含在调查中可能有用的文件的内部内容。

**Microsoft Windows XP [Version 5.1.2600]**

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

**C:\>python dumpRecycleBin.py**

**[\*] Listing Files For User: alex**

**[+] Found File: Notes\_on\_removing\_MetaData.pdf**

**[+] Found File: ANONOPS\_The\_Press\_Release.pdf**

**[\*] Listing Files For User: Administrator**

**[+] Found File: 192.168.13.1-router-config.txt**

**[+] Found File: Room\_Combinations.xls**

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

**元数据**

在本节中，我们将编写一个脚本用来从文件中提取元数据。文件不是清晰可见的对象，元数据可以存在于文档，电子表格，图像，音频和视频等文件类型中。创作应用程序可能会存储一些细节如文件的作者，创建和修改时间，潜在的修订和注释。例如，拍照手机可以标记本地的GPS在照片中或者微软的Word应用程序可以存储文档的作者。检查每一个文件是个艰难的任务，我们可以使用Python自动处理。

**Anonymous的元数据失败**

2010年12月10日，黑客组织Anonymous发布了一份声明稿，描述了最近一次命名为Operation Payback攻击的背后的动机。因为对公司不支持维基解密而感到愤怒，从而对有关公司进行分布式拒绝服务攻击(DDOS)报复。黑客发布的声明稿没有签名，没有来源。是一个PDF文件，但是发行时包含元数据。被创建文档的程序添加进的元数据包含作者的名字Mr. Alex Tapanaris。几天内，警方逮捕了他。

**使用PyPDF解析PDF元数据**

让我们快速创建一个脚本对被逮捕的黑客组织Anonymous的成员用过的文档进行法庭调查取证。Wired.com还保留着ANONOPS\_The\_Press\_Release.pdf那份文档。我们可以从使用wget下载这份文档开始。

**forensic:∼# wget**

**http://www.wired.com/images\_blogs/threatlevel/2010/12/ANONOPS\_The\_**

**Press\_Release.pdf**

**--2012-01-19 11:43:36--**

**http://www.wired.com/images\_blogs/threatlevel/2010/12/ANONOPS\_The\_**

**Press\_Release.pdf**

**Resolving www.wired.com... 64.145.92.35, 64.145.92.34**

**Connecting to www.wired.com|64.145.92.35|:80... connected.**

**HTTP request sent, awaiting response... 200 OK**

**Length: 70214 (69K) [application/pdf]**

**Saving to: 'ANONOPS\_The\_Press\_Release.pdf.1'**

**100%[==================================================**

**================================>] 70,214 364K/s in 0.2s**

**2012-01-19 11:43:39 (364 KB/s) - 'ANONOPS\_The\_Press\_Release.pdf' saved**

**[70214/70214]**

PyPDF是一个优秀的第三方管理PDF文件很实用的库，可以从网站http://pybrary.net/pyPdf/获得。它提供了文档的信息提取，分割，合并，加密和解密的能力。为了提取元数据，我们使用函数getDocumentInfo()。这个方法返回一个元组数组，每一个元组包含一个元数据元素和它的值。遍历这个数组并打印PDF文件的全部元数据。

**import pyPdf**

**from pyPdf import PdfFileReader**

**def printMeta(fileName):**

**pdfFile = PdfFileReader(file(fileName, 'rb'))**

**docInfo = pdfFile.getDocumentInfo()**

**print('[\*] PDF MetaData For: ' + str(fileName))**

**for metaItem in docInfo:**

**print('[+] ' + metaItem + ':' + docInfo[metaItem])**

添加一个选项分析器来识别特定的文件，我们有一个工具可以识别嵌入到PDF中的元数据。同样，我们可以修改我们的脚本来测试特定的元数据，例如特定的用户。当然，这有可能帮助执法管来搜索文件来列出作者名字。

# coding=UTF-8  
**import** pyPdf  
**from** pyPdf **import** PdfFileReader  
**import** optparse  
  
**def printMeta**(fileName):  
 pdfFile = PdfFileReader(file(fileName, 'rb'))  
 docInfo = pdfFile.getDocumentInfo()  
 **print**('[\*] PDF MetaData For: ' + str(fileName))  
 **for** metaItem **in** docInfo:  
 **print**('[+] ' + metaItem + ':' + docInfo[metaItem])  
**def main**():  
 parser = optparse.OptionParser('usage %prog -F <PDF file name>')  
 parser.add\_option('-F', dest='fileName', type='string', help='specify PDF file name')  
 (options, args) = parser.parse\_args()  
 fileName = options.fileName  
 **if** fileName == None:  
 **print**(parser.usage)  
 exit(0)  
 **else**:  
 printMeta(fileName)  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

指定Anonymous发布的声明高运行我们的脚本，我们可以看到同样的元数据导致警方逮捕了Mr. Tapanaris。

**forensic:∼# python pdfRead.py -F ANONOPS\_The\_Press\_Release.pdf**

**[\*] PDF MetaData For: ANONOPS\_The\_Press\_Release.pdf**

**[+] /Author:Alex Tapanaris**

**[+] /Producer:OpenOffice.org 3.2**

**[+] /Creator:Writer**

**[+] /CreationDate:D:20101210031827+02'00'**

**理解Exif元数据**

(Exif是一种图象文件格式，它的数据存储与JPEG格式是完全相同的。实际上Exif格式就是在JPEG格式头部插入了数码照片的信息，包括拍摄时的光圈、快门、白平衡、ISO、焦距、日期时间等各种和拍摄条件以及相机品牌、型号、色彩编码、拍摄时录制的声音以及全球定位系统（GPS）、缩略图等。简单地说，Exif=JPEG+拍摄参数。因此，你可以利用任何可以查看JPEG文件的看图软件浏览Exif格式的照片，但并不是所有的图形程序都能处理Exif信息。)

交换图像文件格式(Exif)标准的定义了如何存储图像和视频文件的规范。如数码相机，扫描仪和智能手机使用这个标准来保存图像和视频文件。Exif标准文件包含了几个对法庭调查取证有用的信息。Phil Harvey编写了一个实用的工具名叫exiftool(从http://www.sno.phy.queensu.ca/~phil/exiftool/可获得)能解析这些参数。检查所有的Exif参数可能会返回几页的信息，所以我们只检查部分需要的参数信息。注意Exif参数包含相机型号名称iPhone 4S以及图像实际的GPS经纬度坐标。这些信息在组织图像是很有帮助的。比如说，Mac OS X应用程序iPhoto使用位置信息来整齐的排列世界地图上的照片。然而，这些信息也被大量的恶意的使用。想象一个士兵将Exif照片放到博客或网站上，敌人可以下载所有的照片几秒钟之类便可以知道士兵的调动信息。在下面的章节中，我们将建立一个脚本来连接WEB网站，下载图像，并检查他们的Exif元数据。

**investigator$ exiftool photo.JPG**

**ExifTool Version Number : 8.76**

**File Name : photo.JPG**

**Directory : /home/investigator/photo.JPG**

**File Size : 1626 kB**

**File Modification Date/Time : 2012:02:01 08:25:37-07:00**

**File Permissions : rw-r--r--**

**File Type : JPEG**

**MIME Type : image/jpeg**

**Exif Byte Order : Big-endian (Motorola, MM)**

**Make**

**: Apple**

**Camera Model Name : iPhone 4S**

**Orientation : Rotate 90 CW**

**<..SNIPPED..>**

**GPS Altitude : 10 m Above Sea Level**

**GPS Latitude : 89 deg 59' 59.97" N**

**GPS Longitude : 36 deg 26' 58.57" W**

**<..SNIPPED..>**

**使用BeautifulSoup下载图像**

可以从www.crummy.com/software/BeautifulSoup/获得BeautifulSoup。BeautifulSoup允许我们快速的解析HTML和XML文档。更新BeautifulSoup到最新版本，并使用easy\_install获取安装BeautifulSoup库。

**investigator:∼# easy\_install beautifulsoup4**

**Searching for beautifulsoup4**

**Reading http://pypi.python.org/simple/beautifulsoup4/**

**<..SNIPPED..>**

**Installed /usr/local/lib/python2.6/dist-packages/beautifulsoup4-4.1.0-**

**py2.6.egg**

**Processing dependencies for beautifulsoup4**

**Finished processing dependencies for beautifulsoup4**

在本节中，我们将使用BeautifulSoup来抓取HTML文档的内容来获取文档中所有的图像。注意，我们使用urllib2打开文档并读取它。接下来我们可以创造BeautifulSoup对象或者一个包含不同HTML文档对象的解析树。用这样的对象，我么可以提取所有的图像标签，通过使用findall(‘img’)函数，这个函数返回一个包含所有图像标签的数组。

**import urllib2**

**from bs4 import BeautifulSoup**

**def findImages(url):**

**print('[+] Finding images on ' + url)**

**urlContent = urllib2.urlopen(url).read()**

**soup = BeautifulSoup(urlContent)**

**imgTags = soup.findAll('img')**

**return imgTags**

接下来，我们需要从网站中下载每一个图像，然后在单独的函数中进行检查。为了下载图像，我们将用到urllib2，urlparse和os模块。首先，我们从图像标签中提取源地址，接着我们读取图像的二进制内容到一个变量，最后我们以写-二进制模式打开文件将图像内容写入文件。

**import urllib2**

**from urlparse import urlsplit**

**from os.path import basename**

**def downloadImage(imgTag):**

**try:**

**print('[+] Dowloading image...')**

**imgSrc = imgTag['src']**

**imgContent = urllib2.urlopen(imgSrc).read()**

**imgFileName = basename(urlsplit(imgSrc)[2])**

**imgFile = open(imgFileName, 'wb')**

**imgFile.write(imgContent)**

**imgFile.close()**

**return imgFileName**

**except:**

**return ''**

**使用Python的图像库从图像阅读Exif元数据**

为了测试图像的内容特到Exif元数据，我们将使用Python图像库PIL来处理文件，可以从http://www.pythonware.com/products/pil/获得，以增加Python的图像处理能力，并允许我们快速的提取与地理位置相关的元数据信息。为了测试文件元数据，我们将打开的对象作为PIL图像对象并使用函数getexif()。接下来我们解析Exif数据到一个数组，通过元数据类型索引。数组完成后，我们可以搜索数组看看它是否包含有GPSInfo的Exif参数。如果它包含GPSInfo参数，我们就知道对象包含GPS元数据并打印信息到屏幕上。

**from PIL import Image**

**from PIL.ExifTags import TAGS**

**def testForExif(imgFileName):**

**try:**

**exifData = {}**

**imgFile = Image.open(imgFileName)**

**info = imgFile.\_getexif()**

**if info:**

**for (tag, value) in info.items():**

**decoded = TAGS.get(tag, tag)**

**exifData[decoded] = value**

**exifGPS = exifData['GPSInfo']**

**if exifGPS:**

**print('[\*] ' + imgFileName + ' contains GPS MetaData')**

**except:**

**Pass**

将所有的包装在一起，我们的脚本现在可以连接到一个URL地址，解析并下载所有的图像文件，然后测试每个文件的Exif元数据。注意main()函数中，我们首先获取站点上的所有图像的列表，然后对数组中的每一个图像，我们将下载图像并测试它的GPS元数据。

# coding=UTF-8  
**import** urllib2  
**import** optparse  
**from** bs4 **import** BeautifulSoup  
**from** urlparse **import** urlsplit  
**from** os.path **import** basename  
**from** PIL **import** Image  
**from** PIL.ExifTags **import** TAGS  
  
**def findImages**(url):  
 **print**('[+] Finding images on ' + url)  
 urlContent = urllib2.urlopen(url).read()  
 soup = BeautifulSoup(urlContent)  
 imgTags = soup.findAll('img')  
 **return** imgTags  
  
**def downloadImage**(imgTag):  
 **try**:  
 **print**('[+] Dowloading image...')  
 imgSrc = imgTag['src']  
 imgContent = urllib2.urlopen(imgSrc).read()  
 imgFileName = basename(urlsplit(imgSrc)[2])  
 imgFile = open(imgFileName, 'wb')  
 imgFile.write(imgContent)  
 imgFile.close()  
 **return** imgFileName  
 **except**:  
 **return** ''  
  
**def testForExif**(imgFileName):  
 **try**:  
 exifData = {}  
 imgFile = Image.open(imgFileName)  
 info = imgFile.\_getexif()  
 **if** info:  
 **for** (tag, value) **in** info.items():  
 decoded = TAGS.get(tag, tag)  
 exifData[decoded] = value  
 exifGPS = exifData['GPSInfo']  
 **if** exifGPS:  
 **print**('[\*] ' + imgFileName + ' contains GPS MetaData')  
 **except**:  
 **pass  
  
def main**():  
 parser = optparse.OptionParser('usage%prog -u <target url>')  
 parser.add\_option('-u', dest='url', type='string', help='specify url address')  
 (options, args) = parser.parse\_args()  
 url = options.url  
 **if** url == None:  
 **print**(parser.usage)  
 exit(0)  
 **else**:  
 imgTags = findImages(url)  
 **for** imgTag **in** imgTags:  
 imgFileName = downloadImage(imgTag)  
 testForExif(imgFileName)  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

对目标地址测试刚刚生成的脚本，我们可以看到其中一个图像包含GPS元数据信息。这个能用于对个人目标的进攻侦查，我们也可以使用此脚本来确认我们自己的漏洞，在黑客攻击前。

**forensics: # python exifFetch.py -u**

**http://www.flickr.com/photos/dvids/4999001925/sizes/o**

**[+] Finding images on**

**http://www.flickr.com/photos/dvids/4999001925/sizes/o**

**[+] Dowloading image...**

**[+] Dowloading image...**

**[+] Dowloading image...**

**[\*] 4999001925\_ab6da92710\_o.jpg contains GPS MetaData**

**[+] Dowloading image...**

**[+] Dowloading image...**

**[+] Dowloading image...**

**用Python调查应用程序结构**

这一节我们将讨论应用程序结构，即两个流行的应用程序存储在SQLite数据库中的数据。SQLite数据库在几个不同的应用程序中是很流行的选择，对于local/client存储类型来说。尤其是WEB浏览器，因为与编程语言不相关绑定。与其相对应的client/server关系数据库，SQLite数据库存储整个数据库在主机上作为单个文件。最初由Dr. Richard Hipp在美国海军工作时创立，SQLite数据库在许多流行的应用程序中的使用不断的增长。被Apple，Mozilla，Google，McAfee，MicrosoftMircso，Intuit，通用电气，DropBox，AdobeAdro甚至是Airbus等公司内建到应用程序中使用SQLite数据库格式。了解如何解析SQLite数据库并在法庭调查取证中使用Python自动处理是非常有用的。下一节的开始，我们将利用流行的语音聊天客户端Skype来审查SQLite数据库。

**了解Skype SQLite3数据库**

作为4.0版本，流行的聊天工具Skype改变了它的内部数据库格式，使用SQLite格式。在Windows系统中，Skype存储了的一个名叫main.db的数据库在路径C:\Documents and Settings\<User>\ApplicationData\Skype\<Skype-account>目录下，在MAC OS X系统中，相同的数据库放在/Users/<User>/Library/Application\ Support/Skype/<Skype-account>目录下。但是Skype存储了什么在该数据库中？为了更好的了解Skype SQLite数据库信息的模式，让我们使用sqlite3命令行工具快速的连接到数据库。连接后，我们执行命令：

SELECT tbl\_name FROM sqlite\_master WHERE type==”table”;

SQLite数据库维护了一个表名为sqlite \_master，这个表包含了列名为tbl\_name，用来描述数据库中的每一个表。执行这句SELECT语句允许我们查看Skype的main.db数据库中的表。我们可以看到，该数据库保存的表包含电话，账户，消息甚至是SMS消息的信息。

**investigator$ sqlite3 main.db**

**SQLite version 3.7.9 2011-11-01 00:52:41**

**Enter ".help" for instructions**

**Enter SQL statements terminated with a ";"**

**sqlite> SELECT tbl\_name FROM sqlite\_master WHERE type=="table";**

**DbMeta**

**Contacts**

**LegacyMessages**

**Calls**

**Accounts**

**Transfers**

**Voicemails**

**Chats**

**Messages**

**ContactGroups**

**Videos**

**SMSes**

**CallMembers**

**ChatMembers**

**Alerts**

**Conversations**

**Participants**

账户表使用Skype应用程序账户的信息。它包含的列包括用户的名字，Skype的简介名称，用户的位置，账户的创建日期等信息。为了查询这些信息，我们可以创建一个SQL语句选择这些列。注意，数据库存储在UNIX时间日期要求转化为更友好的格式。UNIX时间日期提供了一个简单的测量时间方式。它将日期简单的记录为自1970年1月1日来的秒数的整数值。SQL函数datatime()可以将这种值转化为易懂的格式。

**sqlite> SELECT fullname, skypename, city, country, datetime(profile\_**

**timestamp,'unixepoch') FROM accounts;**

**TJ OConnor|<accountname>|New York|us|22010-01-17 16:28:18**

**使用Python的Sqlite3自动完成Skype数据库查询**

连接数据库并执行一个SELECT语句很容易，我们希望能够自动的处理数据库中几个不同的表和列中的额外的信息。让我们利用sqlite3库来编写一个小的Python程序来完成这些。注意我们的函数printProfile()，它创建一个到main.db数据库的连接，创建一个连接之后，它需要一个光标提示然后执行我们先前的SELECT语句，SELECT语句的结果返回一个包含数组的数组。对于每个返回的结果，它包含用户，Skype用户名，位置和介绍数据的索引列。我们解释这些结果，然后漂亮的打印他们到屏幕上。

**# coding=UTF-8**

**import sqlite3**

**def printProfile(skypeDB):**

**conn = sqlite3.connect(skypeDB)**

**c = conn.cursor()**

**c.execute("SELECT fullname, skypename, city, country, datetime(profile\_timestamp,'unixepoch') FROM Accounts;")**

**for row in c:**

**print('[\*] -- Found Account --')**

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

**print('[+] Skype Username: '+str(row[1]))**

**print('[+] Location: '+str(row[2])+','+str(row[3]))**

**print('[+] Profile Date: '+str(row[4]))**

**def main():**

**skypeDB = "main.db"**

**printProfile(skypeDB)**

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

**main()**

运行我们的脚本，我们看到，Skype的main.db数据库包含了一个用户账户，处于隐私的问题，我们用<accountname>代替真正的用户名。

**investigator$ python printProfile.py**

**[\*] -- Found Account --**

**[+] User**

**: TJ OConnor**

**[+] Skype Username : <accountname>**

**[+] Location : New York, NY,us**

**[+] Profile Date : 2010-01-17 16:28:18**

让我们通过检查存储的联系人地址进一步调查Skype的数据库。注意，联系表存储信息如显示名，Skype用户名，位置，移动电话，甚至是生日等每一个联系都存储在数据库中。所有这些个人信息当我们调查或者是攻击一个目标时都是有用的，所以我们将信息收集起来。让我们输出SELECT 语句返回的信息，注意几个字段，比如生日可能是null，在这种情况下，我们利用条件语句只打印不等于空的结果。

**def printContacts(skypeDB):**

**conn = sqlite3.connect(skypeDB)**

**c = conn.cursor()**

**c.execute("SELECT displayname, skypename, city, country, phone\_mobile, birthday FROM Contacts;")**

**for row in c:**

**print('\n[\*] -- Found Contact --')**

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

**print('[+] Skype Username : ' + str(row[1]))**

**if str(row[2]) != '' and str(row[2]) != 'None':**

**print('[+] Location : ' + str(row[2]) + ',' + str(row[3]))**

**if str(row[4]) != 'None':**

**print('[+] Mobile Number : ' + str(row[4]))**

**if str(row[5]) != 'None':**

**print('[+] Birthday : ' + str(row[5]))**

直到现在我们只是从特定的表中提取特定的列检查。然而，当我们想将两个表中的信息一起输出怎么办？在这种情况下，我们不得不将唯一标识结果的值加入数据库表中。为了说明这一点，我们来探究如何输出存储在Skype数据库中的通话记录。为了输出详细的Skype通话记录，我们需要同时使用通话表和联系表。通话表维护着通话的时间戳和和每个通话的唯一索引字段名为conv\_dbid。联系表维护了通话者的身份和每一个电话的ID列明为id。因此，为了连接两个表我们需要查询的SELECT语句有田条件语句WHERE calls.conv\_dbid = conversations.id来确认。这条语句的结果返回包含所有存储在Skype数据库中的Skype的通话记录时间和身份。

**def printCallLog(skypeDB):**

**conn = sqlite3.connect(skypeDB)**

**c = conn.cursor()**

**c.execute("SELECT datetime(begin\_timestamp,'unixepoch'), identity FROM calls, conversations WHERE calls.conv\_dbid = conversations.id;")**

**print('\n[\*] -- Found Calls --')**

**for row in c:**

**print('[+] Time: '+str(row[0]) + ' | Partner: ' + str(row[1]))**

让我们添加最后一个函数来完成我们的脚本。证据丰富，Skype数据库实际默认包含了所有用户发送和接受的信息。存储这些信息的为Message表。从这个表，我们将执行SELECT the timestamp, dialog\_partner, author, and body\_xml (raw text of the message)语句。注意，如果作者来子不同的dialog\_partner，数据库的拥有者发送初始化信息到dialog\_partner。否则，如果作者和dialog\_partner相同，dialog\_partner初始化这些信息，我们将从dialog\_partner打印。

**def printMessages(skypeDB):**

**conn = sqlite3.connect(skypeDB)**

**c = conn.cursor()**

**c.execute("SELECT datetime(timestamp,'unixepoch'), dialog\_partner, author, body\_xml FROM Messages;")**

**print('\n[\*] -- Found Messages --')**

**for row in c:**

**try:**

**if 'partlist' not in str(row[3]):**

**if str(row[1]) != str(row[2]):**

**msgDirection = 'To ' + str(row[1]) + ': '**

**else:**

**msgDirection = 'From ' + str(row[2]) + ': '**

**print('Time: ' + str(row[0]) + ' ' + msgDirection + str(row[3]))**

**except:**

**pass**

将所有的包装在一起，我们有一个非常强的脚本来检查Skype资料数据库。我们的脚本可以打印配置文件信息，联系人地址，通话记录甚至是存储在数据库中的消息。我们可以在main()函数中添加一些选项解析，利用OS模块的功能确保在调查数据库时执行每个函数之前配置文件存在。

# coding=UTF-8  
  
**import** sqlite3  
**import** optparse  
**import** os  
  
**def printProfile**(skypeDB):  
 conn = sqlite3.connect(skypeDB)  
 c = conn.cursor()  
 c.execute("SELECT fullname, skypename, city, country, *datetime*(profile\_timestamp,'unixepoch') FROM Accounts;")  
 **for** row **in** c:  
 **print**('[\*] -- Found Account --')  
 **print**('[+] User: '+str(row[0]))  
 **print**('[+] Skype Username: '+str(row[1]))  
 **print**('[+] Location: '+str(row[2])+','+str(row[3]))  
 **print**('[+] Profile Date: '+str(row[4]))  
  
**def printContacts**(skypeDB):  
 conn = sqlite3.connect(skypeDB)  
 c = conn.cursor()  
 c.execute("SELECT displayname, skypename, city, country, phone\_mobile, birthday FROM Contacts;")  
 **for** row **in** c:  
 **print**('\n[\*] -- Found Contact --')  
 **print**('[+] User : ' + str(row[0]))  
 **print**('[+] Skype Username : ' + str(row[1]))  
 **if** str(row[2]) != '' **and** str(row[2]) != 'None':  
 **print**('[+] Location : ' + str(row[2]) + ',' + str(row[3]))  
 **if** str(row[4]) != 'None':  
 **print**('[+] Mobile Number : ' + str(row[4]))  
 **if** str(row[5]) != 'None':  
 **print**('[+] Birthday : ' + str(row[5]))  
  
**def printCallLog**(skypeDB):  
 conn = sqlite3.connect(skypeDB)  
 c = conn.cursor()  
 c.execute("SELECT *datetime*(begin\_timestamp,'unixepoch'), identity FROM calls, conversations WHERE calls.conv\_dbid = conversations.id;")  
 **print**('\n[\*] -- Found Calls --')  
 **for** row **in** c:  
 **print**('[+] Time: '+str(row[0]) + ' | Partner: ' + str(row[1]))  
  
**def printMessages**(skypeDB):  
 conn = sqlite3.connect(skypeDB)  
 c = conn.cursor()  
 c.execute("SELECT *datetime*(timestamp,'unixepoch'), dialog\_partner, author, body\_xml FROM Messages;")  
 **print**('\n[\*] -- Found Messages --')  
 **for** row **in** c:  
 **try**:  
 **if** 'partlist' **not in** str(row[3]):  
 **if** str(row[1]) != str(row[2]):  
 msgDirection = 'To ' + str(row[1]) + ': '  
 **else**:  
 msgDirection = 'From ' + str(row[2]) + ': '  
 **print**('Time: ' + str(row[0]) + ' ' + msgDirection + str(row[3]))  
 **except**:  
 **pass  
  
def main**():  
 parser = optparse.OptionParser("usage%prog -p <skype profile path> ")  
 parser.add\_option('-p', dest='pathName', type='string', help='specify skype profile path')  
 (options, args) = parser.parse\_args()  
 pathName = options.pathName  
 **if** pathName == None:  
 **print** parser.usage  
 exit(0)  
 **elif** os.path.isdir(pathName) == False:  
 **print** '[!] Path Does Not Exist: ' + pathName  
 exit(0)  
 **else**:  
 skypeDB = os.path.join(pathName, 'main.db')  
 **if** os.path.isfile(skypeDB):  
 printProfile(skypeDB)  
 printContacts(skypeDB)  
 printCallLog(skypeDB)  
 printMessages(skypeDB)  
 **else**:  
 **print** '[!] Skype Database does not exist: ' + skypeDB  
  
**if** \_\_name\_\_ == "\_\_main\_\_":  
 main()

运行该脚本，我们添加一个-p选项来确定Skype配置数据库路径。脚本打印出存储在目标机器上的账户配置，联系人，电话和消息。成功！在下一节中，我们将用我们的sqlite3的知识来调查流行的火狐浏览器存储的结构。

**investigator$ python skype-parse.py -p /root/.Skype/not.myaccount**

**[\*] -- Found Account --**

**[+] User**

**: TJ OConnor**

**[+] Skype Username : <accountname>**

**[+] Location : New York, US**

**[+] Profile Date : 2010-01-17 16:28:18**

**[\*] -- Found Contact --**

**[+] User**

**: Some User**

**[+] Skype Username : some.user**

**[+] Location**

**[+] Mobile Number**

**[+] Birthday**

**: Basking Ridge, NJ,us**

**: +19085555555**

**: 19750101**

**[\*] -- Found Calls --**

**[+] Time: 2011-12-04 15:45:20 | Partner: +18005233273**

**[+] Time: 2011-12-04 15:48:23 | Partner: +18005210810**

**[+] Time: 2011-12-04 15:48:39 | Partner: +18004284322**

**[\*] -- Found Messages --**

**Time: 2011-12-02 00:13:45 From some.user: Have you made plane**

**reservations yets?**

**Time: 2011-12-02 00:14:00 To some.user: Working on it...**

**Time: 2011-12-19 16:39:44 To some.user: Continental does not have any**

**flights available tonight.**

**Time: 2012-01-10 18:01:39 From some.user: Try United or US Airways,**

**they should fly into Jersey.**

**其他有用的Skype查询**

如果有兴趣的话可以花时间更深入的调查Skype数据库，编写新的脚本。考虑以下可能会用到的其他查询：

想只打印出联系人列表中的联系人生日？

SELECT fullname, birthday FROM contacts WHERE birthday > 0;

想打印只有特定的<SKYPE-PARTNER>联系人记录？

SELECT datetime(timestamp,’unixepoch’), dialog\_partner, author, body\_xml

FROM Messages WHERE dialog\_partner = ‘<SKYPE-PARTNER>’

想删除特定的<SKYPE-PARTNER>联系记录？

DELETE FROM messages WHERE skypename = ‘<SKYPE-PARTNER>’

**用Python解析火狐Sqlite3数据库**

在上一节中，我们研究了Skype存储的单一的应用数据库。该数据库提供了大量的调查数据。在本节中，我们将探究火狐存储的是一系列的什么样的数据库。火狐存储这些数据库的默认目录为C:\Documents and Settings\<USER>\Application Data\Mozilla\Firefox\Profiles\<profile folder>\，在Windows系统下，在MAC OS X系统中存储在/Users/<USER>/Library/Application\ Support/Firefox/Profiles/<profile folder>目录下。让我们列出存储在目录中的数据库吧。

**investigator$ ls \*.sqlite**

**places.sqlite downloads.sqlite search.sqlite**

**addons.sqlite extensions.sqlite signons.sqlite**

**chromeappsstore.sqlite formhistory.sqlite webappsstore.sqlite**

**content-prefs.sqlite permissions.sqlite**

**cookies.sqlite places.sqlite**

检查目录列表，很明显火狐存储了相当丰富的数据。但是我们该从哪儿开始调查？让我们从downloads.sqlite数据库开始调查。downloads.sqlite文件存储了火狐用户下载文件的信息。它包含了一个表明为moz\_downloads，用来存储文件名，下载源，下载日期，文件大小，存储在本地的位置等信息。我们使用一个Python脚本来执行SELECT语句来查询适当的列：名称，来源和日期时间。注意火狐用的也是UNIX时间日期。但为了存储UNIX时间日期到数据库，它将日期乘以1000000秒，因此我们正确的时间格式应该是除以1000000秒。

**import sqlite3**

**def printDownloads(downloadDB):**

**conn = sqlite3.connect(downloadDB)**

**c = conn.cursor()**

**c.execute('SELECT name, source, datetime(endTime/1000000, \'unixepoch\') FROM moz\_downloads;')**

**print '\n[\*] --- Files Downloaded --- '**

**for row in c:**

**print('[+] File: ' + str(row[0]) + ' from source: ' + str(row[1]) + ' at: ' + str(row[2]))**

对downloads.sqlite文件运行脚本，我们看到，此配置文件包含了我们以前下载文件的信息。事实上，我们是在前面学习元数据时下载的文件。

**investigator$ python firefoxDownloads.py**

**[\*] --- Files Downloaded ---**

**[+] File: ANONOPS\_The\_Press\_Release.pdf from source:**

**http://www.wired.com/images\_blogs/threatlevel/2010/12/ANONOPS\_The\_**

**Press\_Release.pdf at: 2011-12-14 05:54:31**

好极了!我们现在知道什么时候用户使用火狐下载过什么文件。然而，如果调查者想使用用户的认证重新登陆到网站该怎么办？例如，警方调查员确认用户从基于邮件的网站上下载了对儿童有害的图片该怎么办？警方调查员想重新登陆到网站，最有可能的是缺少密码或者是用户认证的电子邮件。进入cookies，由于HTTP西意缺乏状态设计，网站利用cookies来维护状态。

考虑一下，例如，当用户登陆到站点，如果浏览器不能维护cookies，用户需要登陆能阅读每一个人的私人邮件。火狐存储了这些cookies在cookies. sqlite数据库中。如调查员可以提取cookies并重用，就提供了需要认证才能登陆到资源的条件。

让我们快速编写一个Python脚本提取用户的cookies。我们连接到数据库并执行我们的SELECT语句。在数据库中，moz\_cookies维护这cookies，从cookies.sqlite数据库中的moz\_cookies表中，我们将查询主机，名称，cookies的值，并在屏幕中打印出来。

**def printCookies(cookiesDB):**

**try:**

**conn = sqlite3.connect(cookiesDB)**

**c = conn.cursor()**

**c.execute('SELECT host, name, value FROM moz\_cookies')**

**print('\n[\*] -- Found Cookies --')**

**for row in c:**

**host = str(row[0])**

**name = str(row[1])**

**value = str(row[2])**

**print('[+] Host: ' + host + ', Cookie: ' + name + ', Value: ' + value)**

**except Exception as e:**

**if 'encrypted' in str(e):**

**print('\n[\*] Error reading your cookies database.')**

**print('[\*] Upgrade your Python-Sqlite3 Library')**

**更新sqlite3**

你可能会注意到如果你尝试用默认的sqlite3打开cookies.sqlite数据库会报告文件被加密或者是这不是一个数据库。默认安装的Sqlite3的版本是Sqlite3.6.22不支持WAL日志模式。最新版本的火狐使用 PRAGMA journal\_mode=WAL模式在cookies.sqlite和places.sqlite数据库中。试图用旧版本的Sqlite3或者是sqlite3模块会报错。

**SQLite version 3.6.22**

**Enter ".help" for instructions**

**Enter SQL statements terminated with a ";"**

**sqlite> select \* from moz\_cookies;**

**Error: file is encrypted or is not a database**

**After upgrading your Sqlite3 binary and Pyton-Sqlite3 libraries to**

**a version > 3.7, you should be able to open the newer Firefox**

**databases.**

**investigator:∼# sqlite3.7 ∼/.mozilla/firefox/nq474mcm.default/**

**cookies.sqlite**

**SQLite version 3.7.13 2012-06-11 02:05:22**

**Enter ".help" for instructions**

**Enter SQL statements terminated with a ";"**

**sqlite> select \* from moz\_cookies;**

**1|backtrack-linux.org|\_\_<..SNIPPED..>**

**4|sourceforge.net|sf\_mirror\_attempt|<..SNIPPED..>**

**To avoid our script crashing on this unhandled error, with the**

**cookies.sqlite and places.sqlite databases, we put exceptions to**

**catch the encrypted database error message. To avoid receiving**

**this error, upgrade your Python-Sqlite3 library or use the older**

**Firefox cookies.sqlite and places.sqlite databases included on the**

**companion Web site.**

为了避免我们的脚本在这个错误上崩溃，我们将cookies.sqlite和places.sqlite数据库放在异常处理中。为了避免这个错误，升级你的Python-sqlite3库或使用旧版本的火狐。

调查者可能也希望列举浏览历史，火狐存储这些数据在places.sqlite数据库中。在这里，moz\_places表给我们提供了宝贵的列，包含了什么时候用户访问了什么网站的信息。而我们的脚本printHistory()函数只考虑到moz\_places表，而维基百科推荐使用moz\_places表和moz\_historyvisits表得到浏览器历史。

**def printHistory(placesDB):**

**try:**

**conn = sqlite3.connect(placesDB)**

**c = conn.cursor()**

**c.execute("select url, datetime(visit\_date/1000000, 'unixepoch') from moz\_places, moz\_historyvisits where visit\_count > 0 and moz\_places.id==moz\_historyvisits.place\_id;")**

**print('\n[\*] -- Found History --')**

**for row in c:**

**url = str(row[0])**

**date = str(row[1])**

**print '[+] ' + date + ' - Visited: ' + url**

**except Exception as e:**

**if 'encrypted' in str(e):**

**print('\n[\*] Error reading your places database.')**

**print('[\*] Upgrade your Python-Sqlite3 Library')**

**exit(0)**

让我们使用最后的知识和先前的正则表达式的知识扩展我们的函数。浏览历史及其有价值，对深入了解一些特定的URL很有用。Google搜索查询包含搜索URL内部的权限，比如说，在无线的章节里，我们将就此展开深入。然而，现在让我们只提取搜索条件URL右边的条目。如果在我们的历史里发现包含Google，我们发现他的特点q=后面跟随者&。这个特定的字符序列标识Google搜索。如果我们真的找到这个条目，我们将通过用空格替换一些URL中用的字符来清理输出。最后，我们将打印校正后的输出到屏幕上，现在我们有一个函数可以搜索places.sqlite文件并打印Google搜索查询历史。

**import sqlite3, re**

**def printGoogle(placesDB):**

**conn = sqlite3.connect(placesDB)**

**c = conn.cursor()**

**c.execute("select url, datetime(visit\_date/1000000, 'unixepoch') from moz\_places, moz\_historyvisits where visit\_count > 0 and moz\_places.id==moz\_historyvisits.place\_id;")**

**print('\n[\*] -- Found Google --')**

**for row in c:**

**url = str(row[0])**

**date = str(row[1])**

**if 'google' in url.lower():**

**r = re.findall(r'q=.\*\&', url)**

**if r:**

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

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

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

将所有的包装在一起，我们现在有下载文件信息，读取cookies和浏览历史，甚至是用户的Google的搜索历史功能。该选项的解析应该看看前面非常相似的Skype数据库的探究。

你可能注意到使用os.join.path函数来创建完整的路径会问为什么不是只添加路径和文件的字符串值在一起。是什么让我们不这样使用让我们来看一个例子：

downloadDB = pathName + “\\downloads.sqlite”替换

downloadDB = os.path.join(pathName,“downloads.sqlite”)

考虑一下，Windows用户使用 C:\Users\<user\_name>\来表示路径，而Linux和Mac OS使用/home/<user\_name>/来表示用户路径，不同的操作系统中，斜杠表示的意义不一样，这点当我们创建文件的完整路径时不得不考虑。OS库允许我们创建一个独立于操作系统都能工作的脚本。

# coding=UTF-8  
**import** sqlite3  
**import** re  
**import** optparse  
**import** os  
  
**def printDownloads**(downloadDB):  
 conn = sqlite3.connect(downloadDB)  
 c = conn.cursor()  
 c.execute('SELECT name, source, *datetime*(endTime/1000000, \'unixepoch\') FROM moz\_downloads;')  
 **print** '\n[\*] --- Files Downloaded --- '  
 **for** row **in** c:  
 **print**('[+] File: ' + str(row[0]) + ' from source: ' + str(row[1]) + ' at: ' + str(row[2]))  
**def printCookies**(cookiesDB):  
 **try**:  
 conn = sqlite3.connect(cookiesDB)  
 c = conn.cursor()  
 c.execute('SELECT host, name, value FROM moz\_cookies')  
 **print**('\n[\*] -- Found Cookies --')  
 **for** row **in** c:  
 host = str(row[0])  
 name = str(row[1])  
 value = str(row[2])  
 **print**('[+] Host: ' + host + ', Cookie: ' + name + ', Value: ' + value)  
 **except** Exception **as** e:  
 **if** 'encrypted' **in** str(e):  
 **print**('\n[\*] Error reading your cookies database.')  
 **print**('[\*] Upgrade your Python-Sqlite3 Library')  
**def printHistory**(placesDB):  
 **try**:  
 conn = sqlite3.connect(placesDB)  
 c = conn.cursor()  
 c.execute("select url, *datetime*(visit\_date/1000000, 'unixepoch') from moz\_places, moz\_historyvisits where visit\_count > 0 and moz\_places.id==moz\_historyvisits.place\_id;")  
 **print**('\n[\*] -- Found History --')  
 **for** row **in** c:  
 url = str(row[0])  
 date = str(row[1])  
 **print** '[+] ' + date + ' - Visited: ' + url  
 **except** Exception **as** e:  
 **if** 'encrypted' **in** str(e):  
 **print**('\n[\*] Error reading your places database.')  
 **print**('[\*] Upgrade your Python-Sqlite3 Library')  
 exit(0)  
  
**def printGoogle**(placesDB):  
 conn = sqlite3.connect(placesDB)  
 c = conn.cursor()  
 c.execute("select url, *datetime*(visit\_date/1000000, 'unixepoch') from moz\_places, moz\_historyvisits where visit\_count > 0 and moz\_places.id==moz\_historyvisits.place\_id;")  
 **print**('\n[\*] -- Found Google --')  
 **for** row **in** c:  
 url = str(row[0])  
 date = str(row[1])  
 **if** 'google' **in** url.lower():  
 r = re.findall(r'q=.\*\&', url)  
 **if** r:  
 search=r[0].split('&')[0]  
 search=search.replace('q=', '').replace('+', ' ')  
 **print**('[+] '+date+' - Searched For: ' + search)  
**def main**():  
 parser = optparse.OptionParser("usage%prog -p <firefox profile path> ")  
 parser.add\_option('-p', dest='pathName', type='string', help='specify skype profile path')  
 (options, args) = parser.parse\_args()  
 pathName = options.pathName  
 **if** pathName == None:  
 **print**(parser.usage)  
 exit(0)  
 **elif** os.path.isdir(pathName) == False:  
 **print**('[!] Path Does Not Exist: ' + pathName)  
 exit(0)  
 **else**:  
 downloadDB = os.path.join(pathName, 'downloads.sqlite')  
 **if** os.path.isfile(downloadDB):  
 printDownloads(downloadDB)  
 **else**:  
 **print**('[!] Downloads Db does not exist: '+downloadDB)  
 cookiesDB = os.path.join(pathName, 'cookies.sqlite')  
 **if** os.path.isfile(cookiesDB):  
 printCookies(cookiesDB)  
 **else**:  
 **print**('[!] Cookies Db does not exist:' + cookiesDB)  
 placesDB = os.path.join(pathName, 'places.sqlite')  
 **if** os.path.isfile(placesDB):  
 printHistory(placesDB)  
 printGoogle(placesDB)  
 **else**:  
 **print**('[!] PlacesDb does not exist: ' + placesDB)  
**if** \_\_name\_\_ == "\_\_main\_\_":  
 main()

运行我们的脚本调查火狐用户的配置文件，我们可以看到这些结果。在下一节中，我们将使用部分我们前面学到的技巧，但是通过在数据库的干草堆中搜索一根针来扩展我们的SQLite知识。

**investigator$ python parse-firefox.py -p ∼/Library/Application\**

**Support/Firefox/Profiles/5ab3jj51.default/**

**[\*] --- Files Downloaded ---**

**[+] File: ANONOPS\_The\_Press\_Release.pdf from source:**

**http://www.wired.com/images\_blogs/threatlevel/2010/12/ANONOPS\_The\_**

**Press\_Release.pdf at: 2011-12-14 05:54:31**

**[\*] -- Found Cookies --**

**[+] Host: .mozilla.org, Cookie: wtspl, Value: 894880**

**[+] Host: www.webassessor.com, Cookie: \_\_utma, Value:**

**1.224660440401.13211820353.1352185053.131218016553.1**

**[\*] -- Found History --**

**[+] 2011-11-20 16:28:15 - Visited: http://www.mozilla.com/en-US/**

**firefox/8.0/firstrun/**

**[+] 2011-11-20 16:28:16 - Visited: http://www.mozilla.org/en-US/**

**firefox/8.0/firstrun/**

**[\*] -- Found Google --**

**[+] 2011-12-14 05:33:57 - Searched For: The meaning of life?**

**[+] 2011-12-14 05:52:40 - Searched For: Pterodactyl**

**[+] 2011-12-14 05:59:50 - Searched For: How did Lost end?**

**用Python调查移动设备的iTunes备份**

在2011年4月，安全研究人员和前苹果员工公开了iPhone和Ipad IOS操作系统的一个隐私问题。一个重要的调查之后发现IOS系统事实上跟踪和记录设备的GPS坐标并存储在手机的consolidated.db数据库中。在这个数据库中一个名为 Cell-Location的表包含了收集的手机的GPS坐标。该设备通过综合了最近的手机信号发射塔来确定定位信息为用户提供更好的服务。然而，安全人员提出，该收据可能会被恶意的使用，用来跟踪iPhone/Ipad用户的完整活动路线。此外，使用备份和存储移动设备的信息到电脑上也记录了这些信息。虽然定位记录信息已经从苹果系统中移出了，但发现数据的过程任然还在。在本节中，我们将重复这一过程，从iPhone设备中提取备份信息。具体来说，我们将使用Python脚本从IOS备份中提取所有的文本消息。

当用户对iPhone或者iPad设备进行备份时，它将文件存储到机器的特殊目录。对于Windows系统，iTunes程序存储移动设备备份目录在 C:\Documents and Settings\<USERNAME>\Application Data\AppleComputer\MobileSync\Backup下，在Mac OS X系统上储存目录在 /Users/<USERNAME>/Library/Application Support/MobileSync/Backup/。iTunes程序备份移动设备存储所有的移动设备到这些目录下。让我们来探究我的iPhone最近的备份文件。

**investigator$ ls**

**68b16471ed678a3a470949963678d47b7a415be3**

**68c96ac7d7f02c20e30ba2acc8d91c42f7d2f77f**

**68b16471ed678a3a470949963678d47b7a415be3**

**68d321993fe03f7fe6754f5f4ba15a9893fe38db**

**69005cb27b4af77b149382d1669ee34b30780c99**

**693a31889800047f02c64b0a744e68d2a2cff267**

**6957b494a71f191934601d08ea579b889f417af9**

**698b7961028238a63d02592940088f232d23267e**

**6a2330120539895328d6e84d5575cf44a082c62d**

**<..SNIPPED..>**

为了获得关于每个文件更多的信息，我们将使用UNIX命令file来提取每个文件的文件类型。这个命令使用文件头的字节信息类确认文件类型。这为我们提供了更多的信息，我们看到移动备份目录包含了一些sqlite3数据库，JPEG图像，原始数据和ASCII文本文件。

**investigator$ file \***

**68b16471ed678a3a470949963678d47b7a415be3: data**

**68c96ac7d7f02c20e30ba2acc8d91c42f7d2f77f: SQLite 3.x database**

**68b16471ed678a3a470949963678d47b7a415be3: JPEG image data**

**68d321993fe03f7fe6754f5f4ba15a9893fe38db: JPEG image data**

**69005cb27b4af77b149382d1669ee34b30780c99: JPEG image data**

**693a31889800047f02c64b0a744e68d2a2cff267: SQLite 3.x database**

**6957b494a71f191934601d08ea579b889f417af9: SQLite 3.x database**

**698b7961028238a63d02592940088f232d23267e: JPEG image data**

**6a2330120539895328d6e84d5575cf44a082c62d: ASCII English text**

**<..SNIPPED..>**

file命令让我们知道一些文件包含SQLite数据库并对灭个数据库的内容有少量的描述。我们将使用Python脚本快速的快速的枚举在移动备份目录下找到的每一个数据库的所有的表。注意我们将再次在我们的Python脚本中使用sqlite3。我们的脚本列出工作目录的内容然后尝试连接每一个数据库。对于那些成功的连接，脚本将执行命令：SELECT tbl\_name FROM sqlite\_master WHERE type==‘table’。，每一个SQLite数据库维护了一个sqlite\_master的表包含了数据库的总体结构，说明了数据库的总体架构。上面的命令允许我们列举数据库模式。

**import os**

**import sqlite3**

**def printTables(iphoneDB):**

**try:**

**conn = sqlite3.connect(iphoneDB)**

**c = conn.cursor()**

**c.execute('SELECT tbl\_name FROM sqlite\_master WHERE type==\"table\";')**

**print("\n[\*] Database: "+iphoneDB)**

**for row in c:**

**print("[-] Table: "+str(row))**

**except:**

**pass**

**finally:**

**conn.close()**

**dirList = os.listdir(os.getcwd())**

**for fileName in dirList:**

**printTables(fileName)**

运行我们的脚本，我们列举了移动备份目录里的所有的数据库模式。当脚本找到多个数据库，我们将整理输出我们关心的特定的数据库。注意文件名为d0d7e5fb2ce288813306e4d4636395e047a3d28包含了一个SQLite数据库里面有一个名为messages的表。该数据库包含了存储在iPhone备份中的文本消息列表。

**investigator$ python listTables.py**

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

**[\*] Database: 3939d33868ebfe3743089954bf0e7f3a3a1604fd**

**[-] Table: (u'ItemTable',)**

**[\*] Database: d0d7e5fb2ce288813306e4d4636395e047a3d28**

**[-] Table: (u'\_SqliteDatabaseProperties',)**

**[-] Table: (u'message',)**

**[-] Table: (u'sqlite\_sequence',)**

**[-] Table: (u'msg\_group',)**

**[-] Table: (u'group\_member',)**

**[-] Table: (u'msg\_pieces',)**

**[-] Table: (u'madrid\_attachment',)**

**[-] Table: (u'madrid\_chat',)**

**[\*] Database: 3de971e20008baa84ec3b2e70fc171ca24eb4f58**

**[-] Table: (u'ZFILE',)**

**[-] Table: (u'Z\_1LABELS',)**

**<..SNIPPED..>**

虽然现在我们知道SQLite数据库文件d0d7e5fb2ce288813306e4d4636395e047a3d28包含了文本消息，我们想要能够自动的对不同的备份进行调查。为了执行这个，我们编写了一个简单的函数名为isMessageTable()，这个函数将连接数据库并枚举数据库模式信息。如果文件包含名为messages的表，则返回True，否则函数返回False。现在我们有能力快速扫描目录下的上千个文件并确认包含messages表的特定数据库。

**def isMessageTable(iphoneDB):**

**try:**

**conn = sqlite3.connect(iphoneDB)**

**c = conn.cursor()**

**c.execute('SELECT tbl\_name FROM sqlite\_master WHERE type==\"table\";')**

**for row in c:**

**if 'message' in str(row):**

**return True**

**except:**

**return False**

现在，我们可以定位文本消息数据库了，我们希望可以打印包含在数据库中的内容，如时间，地址，文本消息。为此，我们连接数据库并执行以下命令：

‘select datetime(date,\‘unixepoch\’), address, text from message WHERE address>0;’我们可以打印查询结果到屏幕上。注意，我们将使用一些异常处理，如果isMessageTable()返回的数据库不是我们需要的文本信息数据库，它将不包含数据，地址，和文本的列。如果我们去错了数据库，我们将允许脚本捕获异常并继续执行，直到找到正确的数据库。

**def printMessage(msgDB):**

**try:**

**conn = sqlite3.connect(msgDB)**

**c = conn.cursor()**

**c.execute('select datetime(date,\'unixepoch\'),address, text from message WHERE address>0;')**

**for row in c:**

**date = str(row[0])**

**addr = str(row[1])**

**text = row[2]**

**print('\n[+] Date: '+date+', Addr: '+addr + ' Message: ' + text)**

**except:**

**pass**

包装这些函数在一起，我们可以构建最终的脚本。我们将添加一个选项解析来执行iPhone备份的目录。接下来，我们将列出该目录的内容并测试每一个文件直到找到文本信息数据库。一旦我们找到这个文件，我们可以打印数据库的内容在屏幕上。

# coding=UTF-8  
**import** os  
**import** sqlite3  
**import** optparse  
  
**def isMessageTable**(iphoneDB):  
 **try**:  
 conn = sqlite3.connect(iphoneDB)  
 c = conn.cursor()  
 c.execute('SELECT tbl\_name FROM sqlite\_master WHERE type==\"table\";')  
 **for** row **in** c:  
 **if** 'message' **in** str(row):  
 **return** True  
 **except**:  
 **return** False  
  
**def printMessage**(msgDB):  
 **try**:  
 conn = sqlite3.connect(msgDB)  
 c = conn.cursor()  
 c.execute('select *datetime*(date,\'unixepoch\'),address, text from message WHERE address>0;')  
 **for** row **in** c:  
 date = str(row[0])  
 addr = str(row[1])  
 text = row[2]  
 **print**('\n[+] Date: '+date+', Addr: '+addr + ' Message: ' + text)  
 **except**:  
 **pass  
  
def main**():  
 parser = optparse.OptionParser("usage%prog -p <iPhone Backup Directory> ")  
 parser.add\_option('-p', dest='pathName', type='string',help='specify skype profile path')  
 (options, args) = parser.parse\_args()  
 pathName = options.pathName  
 **if** pathName == None:  
 **print** parser.usage  
 exit(0)  
 **else**:  
 dirList = os.listdir(pathName)  
 **for** fileName **in** dirList:  
 iphoneDB = os.path.join(pathName, fileName)  
 **if** isMessageTable(iphoneDB):  
 **try**:  
 **print**('\n[\*] --- Found Messages ---')  
 printMessage(iphoneDB)  
 **except**:  
 **pass  
  
if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

对iPhone备份目录运行这个脚本，我们可以看到一些存储在iPhone备份中的最近的文本消息。

**investigator$ python iphoneMessages.py -p ∼/Library/Application\ Support/MobileSync/Backup/192fd8d130aa644ea1c644aedbe23708221146a8/**

**[\*] --- Found Messages ---**

**[+] Date: 2011-12-25 03:03:56, Addr: 55555554333 Message: Happy**

**holidays, brother.**

**[+] Date: 2011-12-27 00:03:55, Addr: 55555553274 Message: You didnt**

**respond to my message, are you still working on the book?**

**[+] Date: 2011-12-27 00:47:59, Addr: 55555553947 Message: Quick**

**question, should I delete mobile device backups on iTunes?**

**<..SNIPPED..>**

**本章总结**

再次祝贺！在本章调查数字结构时我们已经编写了不少工具了。通过调查Windows注册表和回收站，藏在元数据中的结构，应用程序存储的数据库我们又增加了一些有用的工具到我们的工具库中。希望你建立在本章的例子基础上回答你将来调查中的问题。