Forensic Investigations with Python

In February 2005, Wichita police forensic investigator Mr. Randy Stone unraveled the nal clues of a 30-year-old mystery. A couple of days earlier, KSAS Television station had handed the police a 3.5” floppy disk they had received from the infamous BTK (Bind, Torture, Kill) Killer. Responsible for at least 10 murders from 1974 to 1991, the BTK Killer eluded capture while repeatedly taunting the police and his victims. On February 16th, 2005, the BTK Killer sent the television station a 3.5” disk with communication instructions. Among these instructions, the disk contained a file named **Test.A.rtf**. (Regan, 2006). While the file contained instructions from the BTK Killer, it also contained something else: *metadata*. Embedded in the Microsoft proprietary Rich Text Format (RTF), the file contained the *first name* of the BTK Killer and the *physical location* at which the user had last saved the file.

This narrowed the investigation to a man named **Denis** at the local Wichita Christ Lutheran Church. Mr. Stone verified that a man named **Denis Rader** served as a church officier at the Lutheran Church (Regan, 2006). With this information, police requested a warrant for a DNA sample from the medical records of Denis Rader’s daughter (Shapiro, 2007). The DNA sample confirmed what Mr. Stone already knew — **Denis Rader was the BTK Killer**. A 31-year investigation that had exhausted 100,000 man hours ended with Mr. Stone’s examination of metadata (Regan, 2006).

RECOVER DELETED ITEMS

Case Study

Anonymous’ Metadata Fail

On December 10, 2010, the hacker group **Anonymous** posted a press release outlining the motivations behind a recent attack named *Operation Payback* (Prefect, 2010). Angry with the companies that had dropped support for the Web site WikiLeaks, Anonymous called for retaliation by performing a distributed denial of service (DDoS) attack against some of the parties involved. The hacker posted the press release *unsigned* and without attribution. Distributed as a Portable Document Format (PDF) file, the press release contained metadata. In addition to the program used to create the document, the PDF metadata contained the name of the author, **Mr. Alex Tapanaris**. Within days, Greek police arrested Mr. Tapanaris (Leyden, 2010).

File can be downloaded from http://www.wired.com/images_blogs/threatlevel/2010/12/ANONOPS_The_Press_Release.pdf (http://www.wired.com/images_blogs/threatlevel/2010/12/ANONOPS_The_Press_Release.pdf).

Part of content is read by general editor and in binary format as follows:

PDF-1.4
%Ã¤Ã¼Ã¶Ã<9f>
2 0 obj
<</Length 3 0 R/Filter/FlateDecode>>
stream
x<9c>¥[I<8b>,¹^Q¾¿_QgC×hËL%^T^E]Ý<95>`ß^F^Z|0>y^CÃØÌ\ü÷X^T^QRª2Û<98>^Gý
ª2µ<84>"¾X^Uå®þò<9f>^_¿^\ù7Ó5]r
<97>ßþvùãï.ÿâçîòÛ?~<¾~Ló5_^V¿\§Ë×_/?mþ^RÃåëï^?º9^?÷7^Wîñæ¢^K.Á·è¦{¸¹ùþV
¾,ô^L<87>%|2ÝóÍå»O7·ÂÓw|ú^NSpè<83>^^^?Ð^?<9f>úv½ÿùë^O?<9e>_?~^^Q7ç¥Ò^V^
RÑö¼¿ÕM¶ûL<9f>¼ó¾|<98><80>^DX9ùPÖö^Qþ$~2É<93>p^?Ã<8f><87>^['w<8d>´u¸xÞÚ
^WfD<97>ÊaaÉ^Yø<93>î<93>+ü^@jry<97>ï^KPôæ#^\Ï#^C^_÷·^YF^_nç<9d>lÆ2ø(^K^
VÚ^WàVáS<80><83>Ï7<92>Âv^?Ë7~[¶q°^MHÆ!k§új+s<80>9@Ý<93>H^DÎ'xÿ<82>^Z?¥k
^X<90>Søûð^Aù^GÇ<9f>`Õ<93><95>òR888Ø^LìÇCùD`^H7>^\<9d><85>>g^Z2áI^RÀËßS
$^Fø<99><89>Xh®§¥Êcæ^B<9d>¿"6Ü^ZNâGÚ<84>^^^W¹<95><93>^AßÖ{å«Ï<85>}ExÞ3^
A8Ð¯üÖ^SC£÷<Ó®<87>ÛD3^Oi~W^Y^D~^DH<99>e×^T^@ISE^R¿ò^KkJ!æ^@E1_<97>^]·aÕ
¯^?¾DÝ´ÎEØ<9d> ^]iûÖ ^GQ¥<<8a>7ÿàã^QüO<88>F
4âÀá<8c>U<82>AR<8e>û
4¸ÌDN%·^Y Té<81>ú×©EQP]>Y^Rø.Ó<87>ºG¶kÔ1HQ.ï^^<87>Ú9Íù:÷|úÀ^C^PE+ÙÁø^]^
A¾h6ÞrÅ^?öï²B^N=Hó*/Þ^Eìï^LÿÔ¼§<89>ðpÆs<9b>W<95>õfo<92>³ÿìy!´¬<9e>çéR++
^ZbÃ¬OÚ÷D»º{¿â>^Eê9Êw5if^QÖ©<8d>H<9b>D<98>LcË&ä<81>à<89>þª<8d>3ô<86><8a
><83>À^TÒw^Z{¤N#¹<9f>)T±<9e>¹·wÌX¢·<9c><85><8d>^A*^P1.VÏº<90>ÅðKªÓ^@D³>
T¾ù%Êc^D7
ý^Eºi6^L<9f>Ê¦^[^M<9b>^FÛ^VÂ"Í<87>^WsÖ3ã^],²¬^Fÿ^W<9f>h^X+<80>º'¿^_k^L<
<83>¿^[ÐcØ/áôôl±P=Þï<8d>Z5k«ã)ÌyTÃ^V
8!<87>Ú-<8c>¨=´·{y<9f>^A$ù^^^^ç^F^C^]<9a>J^K¾ù<95>Y$Q<8d>ÑßêV<8a>³^B;<9
e>^Dhì<8d><88>^O<93>°ËÓ^BÖ<83>mêTyx¡/8<8e>-¾gó<89>ì
<8b>^PáIwQ¨^@ó<8d>ë<8d>´J^TµÆ)F¬pBö§^P<87>Ì·Æµn´<89>²%ºªñu2n3À^M^F^^eÐ±
Yè<84>x&óºì¼ì!^Vë<81>U^YÔ<8f>AÄQÍ¤<8d><9e>¼zÖ<88>þPB^NQµú~^[xB<8a>'<9b>
^M3²<98>ÂM^F©dS¦û<9e>xÐ´®;^OZâ¹L<81>S^T÷
æ<87>ü(»S^N^S¢<93>^QD²Á¼X^]³<8e>q¢Â>³FjÆvú :Å¾¬<8c><8a>2<98>^PNûÍÆ'<93>
ßÑ=ða^HÖI}òÇ8É jäÍ<89>ÕÈ^Y¶<Y^G7¢ë}?#úwÆoñ´!<99>£V¥é^_¶<94>Ì<90>K1^P<83
>]^l»e^YadÌõØl^UÕ<8d>sdpèt<97>^ATNô+åiçt=ÅO<85>^Uì=%>Ú$tB^P½e7<88>Ær<90
>IìK!2ÝîSÆÀ<97>_ÃÒ<82>áìÛu¶ö!^@^Y8É®6^[Ù<9b>ÍLX¤ËyØÜ$q^RñäI&b|Å¾.^G^Sô^
@óÕDl^R<84>i^D¨ö¯Ø<9e>¬é<8a>^S<91>Ñ*ê_^LÁ<88><9a>hòÐ<96>tÉ%ÚÇ<9f>Ö@?äT<
86>7^UH#Þ^\£g^G<86>3ô,;Ûü<9d><80><8d>s_p}Æ<9c>ZN<80>ÇËuLoÊÍlâ<91>N^Dy£9
6<96>ãYgP<8a>ø^VÄõ.ì8¹
<80>^OÔûâ^N!^R
¼ßÓúT=<80>^D¡t<8c>~Bõ
^Mírh^]y}IÆ<9f>ûÁ&êTæ@^Ví^Q'ì^B³$^^XPz½]Ë½Pù0^B²ÄÇg^Hê^@q<86><9f>)ïsè²[
°.<83>4Ã7Ù

METADATA

Using PyPDF2 to Parse PDF Metadata

Let's use Python to quickly recreate the forensic investigation of a document that proved useful in the arrest of a member of the hacker group Anonymous.

Note

use pip to install PyPDF2

```
shell > pip install PyPDF2
```

In [9]:

```
# Row data information, dictionary data
from PyPDF2 import PdfFileReader
pdf_toread = PdfFileReader(open("src/ANONOPS_The_Press_Release.pdf", "rb"))
pdf_info = pdf_toread.getDocumentInfo()
print(str(pdf_info))
```

```
{'/Creator': 'Writer', '/Author': 'Alex Tapanaris', '/Producer': '
OpenOffice.org 3.2', '/CreationDate': "D:20101210031827+02'00'"}
```

In [3]:

```
# Row data information, dictionary data
from PyPDF2 import PdfFileReader
pdf_toread = PdfFileReader(open("../../../../../../Desktop/bank.pdf", "rb"))
pdf_info = pdf_toread.getDocumentInfo()
print(str(pdf_info))
```

```
{'/Producer': 'TeXmacs 1.99.4 + Hummus', '/Author': 'cch', '/Title
': 'bank.tm', '/Creator': 'TeXmacs 1.99.4'}
```

In [10]:

```
# Row data information, dictionary data
from PyPDF2 import PdfFileReader
pdf_toread = PdfFileReader(open("../../../../../../Downloads/掃描0002.pdf", "r
b"))
pdf_info = pdf_toread.getDocumentInfo()
print(str(pdf_info))
```

```
{'/Producer': ''}
```

In [ ]:

```
!ls ../../../../../../Downloads/*.pdf
```

Not easy to read the dict-type data, use python to finalize the data:

```
Dict Type:
   data={key1:value1, key2:val2,...}
-> data[key1] = value1
```

Python solution:

```
data = (dict type ...)
   for item in data:
       print('[+] ' + item + ':' + data[item])
                        ↑              ↑
                       key            value
```

In [10]:

```
import PyPDF2

from PyPDF2 import PdfFileReader

def printMeta(fileName):
    pdfFile = PdfFileReader(open(fileName, 'rb'))
    docInfo = pdfFile.getDocumentInfo()
    print('[*] PDF MetaData For: ' + str(fileName))
    for metaItem in docInfo:
        print('[+] ' + metaItem + ':' + docInfo[metaItem])


def pdfRead(fileName):
    """
    Usage:
    printRead(File)

    For example,
        pdfRead('where/file.pdf')
    """
    printMeta(fileName)
```

In [11]:

```
help(pdfRead)
```

```
Help on function pdfRead in module __main__:

pdfRead(fileName)
    Usage:
    printRead(File)

    For example,
        pdfRead('where/file.pdf')
```

In [7]:

```
#pdfRead('SeniorCitizens.pdf')
pdfRead('src/ANONOPS_The_Press_Release.pdf')
```

```
[*] PDF MetaData For: src/ANONOPS_The_Press_Release.pdf
[+] /Creator:Writer
[+] /Author:Alex Tapanaris
[+] /Producer:OpenOffice.org 3.2
[+] /CreationDate:D:20101210031827+02'00'
```

Analyze Web Images

In [1]:

```
import urllib.request, urllib.error, urllib.parse
import optparse
from urllib.parse import urlsplit
from os.path import basename
from bs4 import BeautifulSoup
from PIL import Image
from PIL.ExifTags import TAGS


def findImages(url):
    print('[+] Finding images on ' + url)
    urlContent = urllib.request.urlopen(url).read()
    soup = BeautifulSoup(urlContent)
    imgTags = soup.findAll('img')
    return imgTags


def downloadImage(imgTag):
    try:
        print('[+] Dowloading image...')
        imgSrc = imgTag['src']
        imgContent = urllib.request.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 list(info.items()):
                decoded = TAGS.get(tag, tag)
                exifData[decoded] = value
            exifGPS = exifData['GPSInfo']
            if exifGPS:
                print('[*] ' + imgFileName + \
                 ' contains GPS MetaData')
    except:
        pass


def main():
    imgTags = findImages(url)
    for imgTag in imgTags:
        imgFileName = downloadImage(imgTag)
        testForExif(imgFileName)
```

In [3]:

```
url="http://flicker.com/photos/dvids/4999001925/sizes/o"
main()
```

```
[+] Finding images on http://flicker.com/photos/dvids/4999001925/s
izes/o

/Users/cch/anaconda/lib/python3.5/site-packages/bs4/__init__.py:18
1: UserWarning: No parser was explicitly specified, so I'm using t
he best available HTML parser for this system ("lxml"). This usual
ly isn't a problem, but if you run this code on another system, or
in a different virtual environment, it may use a different parser
and behave differently.

The code that caused this warning is on line 184 of the file /User
s/cch/anaconda/lib/python3.5/runpy.py. To get rid of this warning,
change code that looks like this:

 BeautifulSoup([your markup])

to this:

 BeautifulSoup([your markup], "lxml")

  markup_type=markup_type))

[+] Dowloading image...
[+] Dowloading image...
[*] 4999001925_ab6da92710_o.jpg contains GPS MetaData
[+] Dowloading image...
[+] Dowloading image...
```

Parsing Firefox Sqlite3 Databases

Link: Mozilla Firefox, sqlite DB, http://forensicswiki.org/wiki/Mozilla_Firefox (http://forensicswiki.org/wiki/Mozilla_Firefox)

Firefox stores these databases in a default directory located at

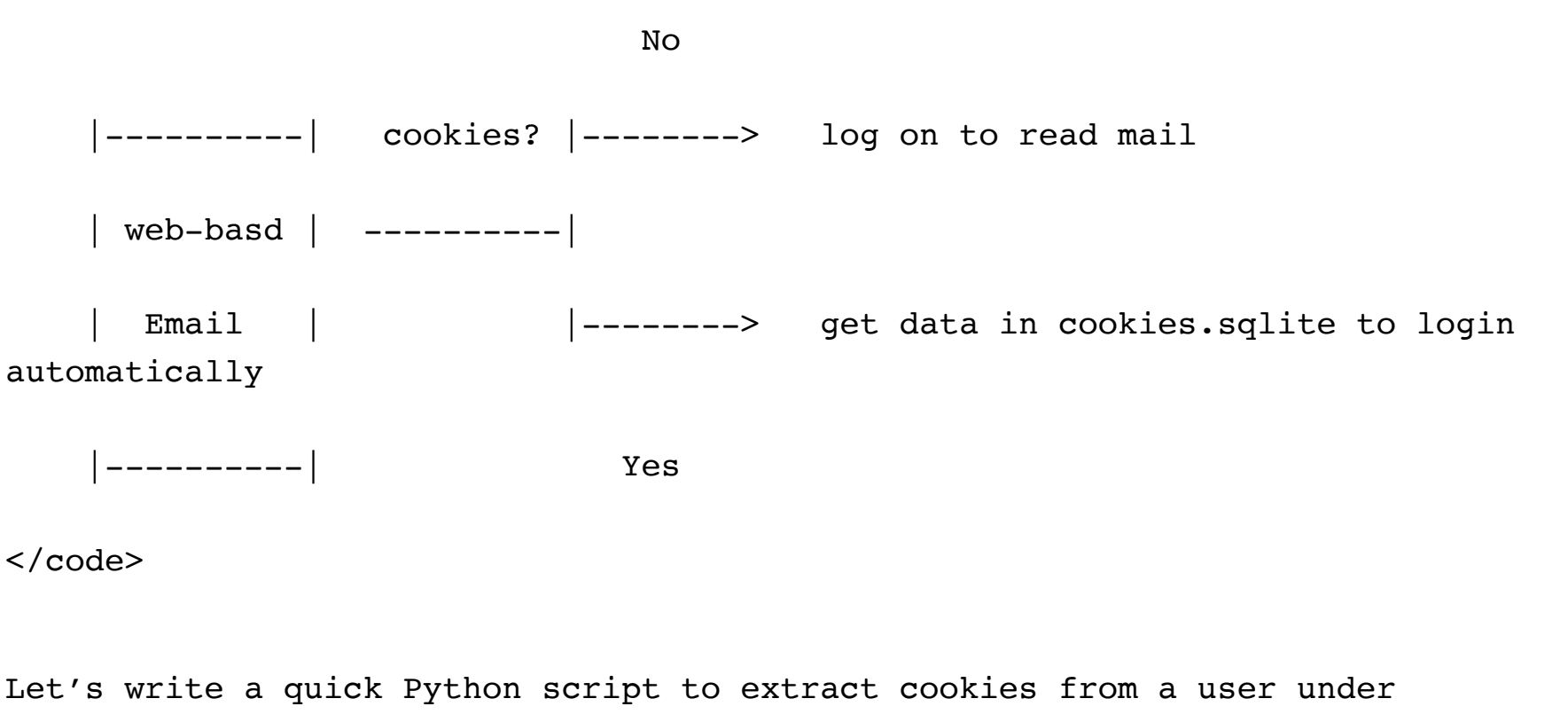
- Windows: `C:\Documents and Settings\\Application Data\Mozilla\Firefox\Profiles\<profile folder>\`
- MacOS: `/Users/<USER>/Library/Application Support/Firefox/Profiles/<profile>/`

The data files in sqlite format include:

```
addons.sqlite formhistory.sqlite signons.sqlite places.sqlite
content-prefs.sqlite healthreport.sqlite  webappsstore.sqlite
cookies.sqlite       permissions.sqlite   extensions.sqlite
```

Why Cookies

Consider, for example, when a user logs onto a web-based email: if the browser could not maintain cookies, the user would have to log on in order to read every individual email. Firefox stores these cookies in a database named cookies.sqlite. If an investigator can extract cookies and reuse them, it provides the opportunity to log on to resources that require authentication.



</code>

Let’s write a quick Python script to extract cookies from a user under investigation. We connect to the database and execute our SELECT statement. In the database, the moz_cookies maintains the data regarding cookies. From the moz_cookies table in the cookies.sqlite database, we will query the column values for host, name, and cookie value, and print them to the screen.

Though the sqlite is created as binany, part of downloads.sqlite should give the details of table what it owns:

```
...
CREATE TABLE moz_downloads (id INTEGER PRIMARY KEY, name TEXT, source TEXT, target TEXT, tempPath TEXT, startTime INTEGER, endTime INTEGER, state INTEGER, referrer TEXT, entityID TEXT, currBytes INTEGER NOT NULL DEFAULT 0, maxBytes INTEGER NOT NULL DEFAULT -1, mimeType TEXT, preferredApplication TEXT, preferredAction INTEGER NOT NULL DEFAULT 0, autoResume INTEGER NOT NULL DEFAULT 0)
...
```

i.e. the details of created table and its columns are listed as follows:

```
Table
  moz_downloads
  Columns: type/value
      id: INTEGER (PRIMARY KEY),
      name: TEXT,
      source: TEXT,
      target: TEXT,
      tempPath: TEXT,
      startTime: INTEGER,
      endTime: INTEGER,
      state: INTEGER,
      referrer: TEXT,
      entityID TEXT,
      currBytes INTEGER NOT NULL DEFAULT 0,
      maxBytes INTEGER NOT NULL DEFAULT -1,
      mimeType TEXT,
      preferredApplication TEXT,
      preferredAction INTEGER NOT NULL DEFAULT 0,
      autoResume INTEGER NOT NULL DEFAULT 0
```

In [ ]:

Investingate the Sqlite Data

- use Sqlite library:

```
shell> sqlite3 cookies.sqlite
SQLite version 3.13.0 2016-05-18 10:57:30
Enter ".help" for usage hints.
sqlite> select * from moz_cookies;
1|google.com||__utma|29003808.1143615086.1386662234.1386662234.1386662234.1|.mail.

2|google.com||__utmb|29003808.2.9.1386662236726|.mail.google.com|/intl/|1386664036

...
```

- by Pyhon's sqlite3 library as follows:

In [7]:

```
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]))

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')
```

In [2]:

```
printDownloads('firefox_profile/downloads.sqlite')
```

```
[*] --- Files Downloaded ---
[+] File: python-nmap-0.1.4.tar.gz from source: http://xael.org/norman/pyt
hon/python-nmap/python-nmap-0.1.4.tar.gz at: 2012-06-20 02:53:09
```

In [ ]:

```
printCookies('Cookies.sqlite')
```

An investigator may also wish to enumerate the browser history. Firefox stores this data in a database named places.sqlite. Here, the moz_places table gives us valuable columns that include information about when (date) and where (address) a user visited a site. While our script for printHistory() only takes into account the moz_places table, the ForensicWiki Web site recommends using data from both the moz_places table and the moz_historyvisits table as well to get a live browser history (Forensics Wiki, 2011).

In [13]:

```
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)
```

In [ ]:

```
printHistory('places.sqlite')
```

Analyze-Browsing-History,--Recognize-Ourselves (History_Analyze.ipynb)

Practice on Google Chrome Browser

In [ ]:

```
f = open('C:\Users\[username]\AppData\Local\Google\Chrome\User Data\Default\History'
, 'rb')
data = f.read()
f.close()
f = open('your_expected_file_path', 'w')
f.write(repr(data))
f.close()
```

In [9]:

```
import 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)
```

Let’s use the last example and our knowledge of regular expressions to expand the previous function. While browser history is in nitely valuable, it would be useful to look deeper into some of the speci c URLs visited. Google search queries contain the search terms right inside of the URL, for example. In the wireless section, we will expand on this in great depth. However, right now, let’s just extract the search terms right out of the URL. If we spot a URL in our history that contains Google, we will search it for the characters q= followed by an &. This speci c sequence of characters indicates a Google search. If we do nd this term, we will clean up the output by replacing some of the characters used in URLs to pad whitespace with actual whitespace. Finally, we will print out the corrected output to the screen. Now we have a function that can search the places.sqlite le for and print out Google search queries.

In [ ]:

```
printGoogle('places.sqlite')
```

In [ ]:

INVESTIGATING ITUNES MOBILE BACKUPS

In April 2011, security researcher and former Apple employee Pete Warden disclosed a privacy issue with the popular Apple iPhone/Ipad iOS operating system (Warden, 2011). After a significant investigation, Mr. Warden revealed proof that the Apple iOS operating system actually tracked and recorded the GPS coordinates of the device and stored them in a database on the phone called consolidated.db (Warden, 2011). Inside this database, a table named Cell-Location contained the GPS points the phone had collected. The device determined the location information by triangulating off the nearest cell-phone towers in order to provide the best service for the device user. However, as Mr. Warden suggested, this same data could be used maliciously to track the entire movements an iPhone/iPad user. Furthermore, the process used to backup and store a copy of the mobile device to a computer also recorded this information. While the location-recording information has been removed from the Apple iOS operating system functionality, the process Mr. Warden used to discover the data remains. In this section, we will repeat this process to extract information from iOS mobile device backups. Specifically, we will extract all the text messages out of an iOS backup using a Python script.

~/Library/Application Support/MobileSync/Backup

In [9]:

```
import os

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 iphoneMessage(pathName):
    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
```

Try

```
    iphoneMessage('where-the backup-dircetory')
```

In [7]:

```
import plistlib, os.path, datetime, pytz, tzlocal, sys


_, start_days, end_days = sys.argv
start_days = int(start_days)
end_days = int(end_days)



apple_epoch = datetime.datetime(
        2001, 1, 1, 0, 0, 0, 0,
        tzinfo=pytz.timezone('UTC')
)
def parse_apple_time(apple_time):
        return apple_epoch + datetime.timedelta(
                seconds=float(apple_time)
        )


local_tz = tzlocal.get_localzone()
def to_local(when):
        return when.astimezone(
                local_tz
        )


class Item:
        def __init__(self, data):
                self.when = parse_apple_time(data['lastVisitedDate'])
                self.title = data.get('title', '')
                self.address = data['']
```

```
        def __str__(self):
                return '%s\t%s' % (to_local(self.when), self.title)




def load_history():

        history = plistlib.load(
                open(os.path.expanduser('~/library/Safari/History.plist'), 'rb')
        )['WebHistoryDates']

        items = []

        for data in history:

                items.append(Item(data))


        return items


history = load_history()
history = filter( lambda _: datetime.date.today() + datetime.timedelta(days=start_da
ys) <= to_local(_.when).date() <= datetime.date.today() + datetime.timedelta(days=en
d_days), history )
history = sorted( history, key=lambda _: _.when )


#for item in sorted(filter(load_history(), lambda _: to_local(_.when).day() < dateti
me.date.today()), key=lambda _: _.when):
#       print(item)

for item in history:
        print(item)



#print('\n'.join('%s\t%s' % ((datetime.datetime(2001, 1, 1, 0, 0, 0, 0,tzinfo=pytz.t
imezone('UTC')) + datetime.timedelta(seconds=float(_['lastVisitedDate'])))).astimezon
e(tzlocal.get_localzone()),_.get('title','')) 
#       for _ in plistlib.load(open(os.path.expanduser('~/library/Safari/History.pli
st'), 'rb'))['WebHistoryDates'])
#)
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call last
)
<ipython-input-7-c6bc5410aba2> in <module>()
      3
      4 _, start_days, end_days = sys.argv
----> 5 start_days = int(start_days)
      6 end_days = int(end_days)
      7

ValueError: invalid literal for int() with base 10: '-f'
```

In [8]:

```
import os
import plistlib

def main():

   fileName=os.path.expanduser('Downloads.plist')

   if os.path.exists(fileName):

      pl=plistlib.readPlist(fileName)

      print('\nThe plist full contents is %s\n' % pl)

      if 'aString' in pl:
         print('The aString value is %s\n' % pl['aString'])
      else:
         print('There is no aString in the plist\n')

   else:
      print('%s does not exist, so can\'t be read' % fileName)
```

In [ ]:

```
main()
```

In [10]:

```
pl = plistlib.readPlist("Downloads.plist")
item = pl["aKey"]
```

```
/Users/cch/anaconda/lib/python3.5/site-packages/ipykernel/__main__.py:1: D
eprecationWarning: The readPlist function is deprecated, use load() instea
d
  if __name__ == '__main__':
```

```
---------------------------------------------------------------------------
-
KeyError                                  Traceback (most recent call last
)
<ipython-input-10-a14b08e5811a> in <module>()
      1 pl = plistlib.readPlist("Downloads.plist")
----> 2 item = pl["aKey"]

KeyError: 'aKey'
```

In [ ]: