CONF42

PYTHON

Build Your First Cyber Forensic Application using Python

Gajendra Deshpande
https://gcdeshpande.github.io

Outline of the Talk

- Introduction to digital crimes, digital forensics, the process of investigation, and the collection of evidence.
- Setting up Python for forensics application development
- Built-in functions and modules for forensic tasks
- Forensic Indexing and searching
- Hash functions for forensics
- Forensic Evidence extraction
- Meta data forensics
- Using Natural Language Tools in Forensics
- Summary

Cyber Crime Statistics

- The Internet Crime Report for 2019, released by USA's Internet Crime Complaint Centre (IC3) of the Federal Bureau of Investigation, has revealed top 4 countries that are victims of internet crimes.
  - USA-4,67,361; UK-93,796; Canada-33,000; India-27,248 (2018 data)
- According to RSA report (2015), mobile transactions are rapidly growing and cyber criminals are migrating to less protected 'soft' channels.
- According to report by Norton (2015), an estimated 113 million Indians lost about Rs. 16,558 on an average to cybercrime
- According to an article published in Indian Express on 19th November 2016, *Over 55% Millennials in India Hit by Cybercrime*.
- A recent study by the CheckPoint Research has recorded over 1,50,000 cyber-attacks every week during the COVID-19 pandemic. There has been an increase of 30% in cyber-attacks compared to previous weeks.

Digital Forensics

- Forensic science is the use of scientific methods or expertise to investigate crimes or examine evidence that might be presented in a court of law.
- Cyber Forensics is investigation of various crimes happening in the cyber space.
- Examples of cyber-attacks include phishing, ransomware, fake news, fake medicine, extortion, and insider frauds.
- According to DFRWS, Digital Forensics can be defined as "the use of scientifically derived and proven method toward the preservation, collection, validation, identification, analysis, interpretation, documentation, and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations."

Digital Forensics Investigation Process

- Identification
- Collection
- Validation
- Examination
- Preservation
- Presentation

Daubert Standard and Python

- In United States federal law, the Daubert standard is a rule of evidence regarding the admissibility of expert witness testimony. A party may raise a Daubert motion, a special motion in limine raised before or during trial, to exclude the presentation of unqualified evidence to the jury.
- **Illustrative factors:** The Court defined "scientific methodology" as the process of formulating hypotheses and then conducting experiments to prove or falsify the hypothesis, and provided a set of illustrative factors (i.e., not a "test"). Pursuant to Rule 104(a), in *Daubert* the U.S. Supreme Court suggested that the following factors be considered:
  - Has the technique been tested in actual field conditions (and not just in a laboratory)?
  - Has the technique been subject to peer review and publication?
  - What is the known or potential rate of error?
  - Do standards exist for the control of the technique's operation?
  - Has the technique been generally accepted within the relevant scientific community?

Source: https://en.wikipedia.org/wiki/Daubert_standard

Daubert Standard and Python

- In 2003, Brian Carrier [Carrier] published a paper that examined rules of evidence standards including Daubert, and compared and contrasted the open source and closed source forensic tools. One of his key conclusions was, "Using the guidelines of the Daubert tests, we have shown that open source tools may more clearly and comprehensively meet the guideline requirements than would closed source tools."
- The results are not automatic of course, just because the source is open. Rather, specific steps must be followed regarding design, development, and validation.
  - Can the program or algorithm be explained? This explanation should be explained in words, not only in code.
  - Has enough information been provided such that thorough tests can be developed to test the program?
  - Have error rates been calculated and validated independently?
  - Has the program been studied and peer reviewed?
  - Has the program been generally accepted by the community?

Source: Python Forensics by Chet Hosmer

Setting up Python for forensics application development

- Factors
  - Your background and organization support
  - Choosing third party libraries
  - IDEs and their features
- Installation (Operating System)
- Right version of Python
- Graphical vs Shell

Built-in Functions and Modules

dir(__builtins__)

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Source: https://stackoverflow.com/questions/45528559/retrieve-all-builtin-functions

Python 3.6 (64-bit)

```
Python 3.6.6 (v3.6.6:4cf1f54eb7, Jun 27 2018, 03:37:03) [MSC v.1900
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more informatio
n.
>>> baseAddress = "127.0.0"
>>> hostAddresses = range(10)
>>> ipRange = []
>>> for i in hostAddresses:
...   ipRange.append(baseAddress+str(i))
...
>>> for ipAddr in ipRange:
...   print(ipAddr)
...
127.0.00
127.0.01
127.0.02
127.0.03
127.0.04
127.0.05
127.0.06
127.0.07
```

Python 3.6 (64-bit)

```
Python 3.6.6 (v3.6.6:4cf1f54eb7, Jun 27 2018, 03:37:03) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> pwd=os.getcwd()
>>> dir=os.listdir(pwd)
>>> for names in dir:
...   print(names)
...
...
DLLs
Doc
include
Lib
libs
LICENSE.txt
NEWS.txt
python.exe
python.pdb
python3.dll
python36.dll
python36.pdb
pythonw.exe
pythonw.pdb
Scripts
```

Forensic Indexing and Searching

- You can use simple file search and index() function

main.py

```
1   searchWords = set()
2
3   fileWords = open('keywords.txt')
4   for line in fileWords:
5       searchWords.add(line.strip())
6   print(searchWords)
7
8   if ('Python' in searchWords):
9       print("Found word")
10  else:
11      print("not found")
12
```

Console Shell

```
{'one', 'most', 'the', 'world', 'is', 'Python', 'popular', 'languages', 'in'}
Found word
>
```

Whoosh: Forensic Indexing and Searching

- Whoosh was created and is maintained by Matt Chaput. It was originally created for use in the online help system of Side Effects Software's 3D animation software Houdini.
- Pythonic API.
- Pure-Python.
- Fielded indexing and search.
- Fast indexing and retrieval
- Pluggable scoring algorithm (including BM25F), text analysis, storage, posting format, etc.
- Powerful query language.
- Pure Python spell-checker

Whoosh: Forensic Indexing and Searching

```
>>> from whoosh.index import create_in
>>> from whoosh.fields import *
>>> schema = Schema(title=TEXT(stored=True), path=ID(stored=True), content=TEXT)
>>> ix = create_in("indexdir", schema)
>>> writer = ix.writer()
>>> writer.add_document(title=u"First document", path=u"/a",
...                     content=u"This is the first document we've added!")
>>> writer.add_document(title=u"Second document", path=u"/b",
...                     content=u"The second one is even more interesting!")
>>> writer.commit()
>>> from whoosh.qparser import QueryParser
>>> with ix.searcher() as searcher:
...     query = QueryParser("content", ix.schema).parse("first")
...     results = searcher.search(query)
...     results[0]
...
{"title": u"First document", "path": u"/a"}
```

Source: https://whoosh.readthedocs.io/en/latest/quickstart.html

Hash Functions for Forensics

main.py

```
1   import hashlib  # import hashlib module
2
3   print('\nExample for SHA256')
4   m = hashlib.sha256()
5
6   m.update(b"Python is a")
7   m.update(b" great programming language.")
8   print('Output 1 :', m.digest())
9
10  x = hashlib.sha256(b"Python is a great programming
    language.")
11  print('Output 2 :', x.digest())
12
13  print(x.digest()==m.digest())
14
```

Console Shell

```
Example for SHA256
Output 1 : b'8\xc3\xa2\xe80\xf5\xd1\xfb</\x1a\x94\xcfu\xb09\xf1
xc0\x82B\xeb~C\x81\xcdzW\x14)1\x11\xfd'
Output 2 : b'8\xc3\xa2\xe80\xf5\xd1\xfb</\x1a\x94\xcfu\xb09\xf1
xc0\x82B\xeb~C\x81\xcdzW\x14)1\x11\xfd'
True
>
```

Source: https://www.journaldev.com/16035/python-hashlib

Hash Functions for forensics

main.py

```python
import hashlib  # import hashlib module

print('\nExample for SHA256')
m = hashlib.sha256()

m.update(b"Python is a")
m.update(b" great programming language.")
print('Output 1 :', m.digest())

x = hashlib.sha256(b"Python is a great programming language. ")
print('Output 2 :', x.digest())

print(x.digest()==m.digest())

```

Console Shell

```
Example for SHA256
Output 1 : b'8\xc3\xa2\xe80\xf5\xd1\xfb</\x1a\x94\xcfu\xb09\xf1\
0\x82B\xeb~C\x81\xcdzW\x14)1\x11\xfd'
Output 2 : b'\xff]x\x95\x059\x9e\xb3y\x10\xe3\x93\xe0$Ip\xc7\xb3
x882\xc5a\xb2\xcb\xeet\x82t\xb9V\xde'
False
>
```

Forensic Evidence Extraction

- Pillow is the friendly PIL fork by Alex Clark and Contributors. PIL is the Python Imaging Library by Fredrik Lundh and Contributors.
- The Python Imaging Library adds image processing capabilities to your Python interpreter.
- This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.
- The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

Source: https://pillow.readthedocs.io/en/stable/

Forensic Evidence Extraction

```
from PIL.EXIFTags import TAGS
EXIFTAGS = TAGS.items()
print(EXIFTags)
```

```
from PIL.ExifTags import GPSTAGS
gps=GPSTAGS.items()
print(gps)
```

```
from PIL import Image
from PIL.ExifTags import TAGS, GPSTAGS
pilImage=Image.open("C:\\Users\\gcdeshpande\\Desktop\\image.png")
ExifData = pilImage._getexif()
imgExif = ExifData.items()
print(imgExif)
```

Pyscreenshot

- Pyscreenshot tries to allow to take screenshots without installing 3rd party libraries. It is cross-platform but mainly useful for Linux based distributions.
- Features: Cross-platform wrapper; Capturing the whole desktop or an area; pure Python library; Supported Python versions: 2.7, 3.6, 3.7, 3.8; Interactivity is not supported; Mouse pointer is not visible.
- Performance is not a target for this library, but you can benchmark the possible settings and choose the fastest one.

Installation

$ python3 -m pip install Pillow pyscreenshot

Source: https://github.com/ponty/pyscreenshot

Pyscreenshot – Full Screen

```
# pyscreenshot/examples/grabfullscreen.py

"Grab the whole screen"
import pyscreenshot as ImageGrab

# grab fullscreen
im = ImageGrab.grab()

# save image file
im.save("fullscreen.png")
```

Pyscreenshot – Part of Screen

```
# pyscreenshot/examples/grabbox.py

"Grab the part of the screen"
import pyscreenshot as ImageGrab

# part of the screen
im = ImageGrab.grab(bbox=(10, 10, 510, 510))  # X1,Y1,X2,Y2

# save image file
im.save("box.png")
```

Pyscreenshot - Performance

```
$ python3 -m pyscreenshot.check.speedtest

n=10
------------------------------------------------------
default                 1    sec        (  102 ms per call)
pil
mss                     2.1  sec        (  214 ms per call)
scrot                   1    sec        (  101 ms per call)
maim                    1.5  sec        (  147 ms per call)
imagemagick             2.5  sec        (  247 ms per call)
pyqt5                   4.4  sec        (  442 ms per call)
pyqt                    3.5  sec        (  352 ms per call)
pyside2                 5    sec        (  495 ms per call)
pyside                  3.5  sec        (  350 ms per call)
wx                      3.3  sec        (  329 ms per call)
pygdk3                  2.3  sec        (  225 ms per call)
mac_screencapture
mac_quartz
gnome_dbus              1.7  sec        (  166 ms per call)
gnome-screenshot        2.3  sec        (  231 ms per call)
kwin_dbus
```

Pyscreenshot – Performance

```
$ python3 -m pyscreenshot.check.speedtest --childprocess 0

n=10
------------------------------------------------------
default                 0.16 sec        (   16 ms per call)
pil
mss                     0.17 sec        (   17 ms per call)
scrot                   1    sec        (  104 ms per call)
maim                    1.5  sec        (  145 ms per call)
imagemagick             2.5  sec        (  246 ms per call)
pyqt5                   1.1  sec        (  110 ms per call)
pyqt                    1    sec        (  104 ms per call)
pyside2                 1.2  sec        (  121 ms per call)
pyside                  1    sec        (  104 ms per call)
wx                      0.33 sec        (   32 ms per call)
pygdk3                  0.2  sec        (   19 ms per call)
mac_screencapture
mac_quartz
gnome_dbus              1.5  sec        (  152 ms per call)
gnome-screenshot        2.3  sec        (  230 ms per call)
kwin_dbus
```

Pyscreenshot – Force backend

```
import pyscreenshot as ImageGrab
im = ImageGrab.grab(backend="scrot")
```

```
# best performance
import pyscreenshot as ImageGrab
im = ImageGrab.grab(backend="mss", childprocess=False)
```

Metadata Forensics

- Mutagen is a Python module to handle audio metadata.
- It supports ASF, FLAC, MP4, Monkey's Audio, MP3, Musepack, Ogg Opus, Ogg FLAC, Ogg Speex, Ogg Theora, Ogg Vorbis, True Audio, WavPack, OptimFROG, and AIFF audio files. All versions of ID3v2 are supported, and all standard ID3v2.4 frames are parsed. It can read Xing headers to accurately calculate the bitrate and length of MP3s. ID3 and APEv2 tags can be edited regardless of audio format. It can also manipulate Ogg streams on an individual packet/page level.
- Mutagen works with Python 3.6+ (CPython and PyPy) on Linux, Windows and macOS, and has no dependencies outside the Python standard library.

Source: https://mutagen.readthedocs.io/en/latest/

Metadata Forensics

Installation: python3 -m pip install mutagen

The File functions takes any audio file, guesses its type and returns a FileType instance or None

```
>>> import mutagen
>>> mutagen.File("11. The Way It Is.ogg")
{'album': [u'Always Outnumbered, Never Outgunned'],
 'title': [u'The Way It Is'], 'artist': [u'The Prodigy'],
 'tracktotal': [u'12'], 'albumartist': [u'The Prodigy'],'date': [u'2004'],
 'tracknumber': [u'11'],
>>> _.info.pprint()
u'Ogg Vorbis, 346.43 seconds, 499821 bps'
>>>
```

Metadata Forensics

The File functions takes any audio file, guesses its type and returns a FileType instance or None

```
from mutagen.flac import FLAC

audio = FLAC("example.flac")
audio["title"] = u"An example"
audio.pprint()
audio.save()
```

The following example gets the length and bitrate of an MP3 file.

```
from mutagen.mp3 import MP3

audio = MP3("example.mp3")
print(audio.info.length)
print(audio.info.bitrate)
```

Metadata Forensics

PyPDF2 - Pure-Python library built as a PDF toolkit. It is capable of:

- extracting document information (title, author, …)
- splitting documents page by page
- merging documents page by page
- cropping pages
- merging multiple pages into a single page
- encrypting and decrypting PDF files
- It is a useful tool for websites that manage or manipulate PDFs.

Source: https://pypi.org/project/PyPDF2/

Metadata Forensics

- pefile is a multi-platform Python module to parse and work with Portable Executable (PE) files. Most of the information contained in the PE file headers is accessible, as well as all the sections' details and data.
- The structures defined in the Windows header files will be accessible as attributes in the PE instance. The naming of fields/attributes will try to adhere to the naming scheme in those headers. Only shortcuts added for convenience will depart from that convention.
- pefile requires some basic understanding of the layout of a PE file — with it, it's possible to explore nearly every single feature of the PE file format.

Source: https://github.com/erocarrera/pefile

Metadata Forensics

Some of the tasks that pefile makes possible are:

- Inspecting headers
- Analyzing of sections' data
- Retrieving embedded data
- Reading strings from the resources
- Warnings for suspicious and malformed values
- Basic butchering of PEs, like writing to some fields and other parts of the PE
- This functionality won't rearrange PE file structures to make room for new fields, so use it with care.
- Overwriting fields should mostly be safe.
- Packer detection with PEiD's signatures
- PEiD signature generation

Using Natural Language Tools

- Examine the text for evidence using NLP concepts
- NLTK, spaCy, Textacy
- Stanza, Polyglot
- inltk, indic-nlp

Summary

- It is very important to follow the standard procedure laid by law enforcement agencies during investigation process.
- There are many open source as well as commercial tools for digital forensics. Learning to develop your own tool is advantageous.
- Many tools written in Python are pure Python implementations. And most importantly Python and Open Source tools comply with Daubert Standard.

Thank You!

Widescreen Test Pattern (16:9)