Automating Packet Analysis with Python

Joe McManus

mcmanus@automox.com

Who?

- Currently: CISO of Automox & Sr. Researcher CERT/SEI/CMU
- Past: Professor @ CU, Director of Security @ SolidFire, Head of R&D @ Webroot.

- MS @ Carnegie Mellon
- BS @ U of MD
- PhD From CU 2019(?, working on it)

What?

- Analyzing Packets with Python
- Use Scapy to pull out DNS queries, URLs, etc
- Use Pandas for time series
- Use Plotly for graphs

Why?

- During incident response you tend to do the same steps.
- Wireshark is neat, but time consuming
- Some things are better left to automation
- Python is fun
- **<u>Packets don’t lie!</u>**

Where?

- In a Virtual Machine
  - Fedora 28
- But you can do this anywhere
  - Requires python3-scapy
    - Prettytable
    - Pandas
    - Plotly
    - scapy_http
    - networkx

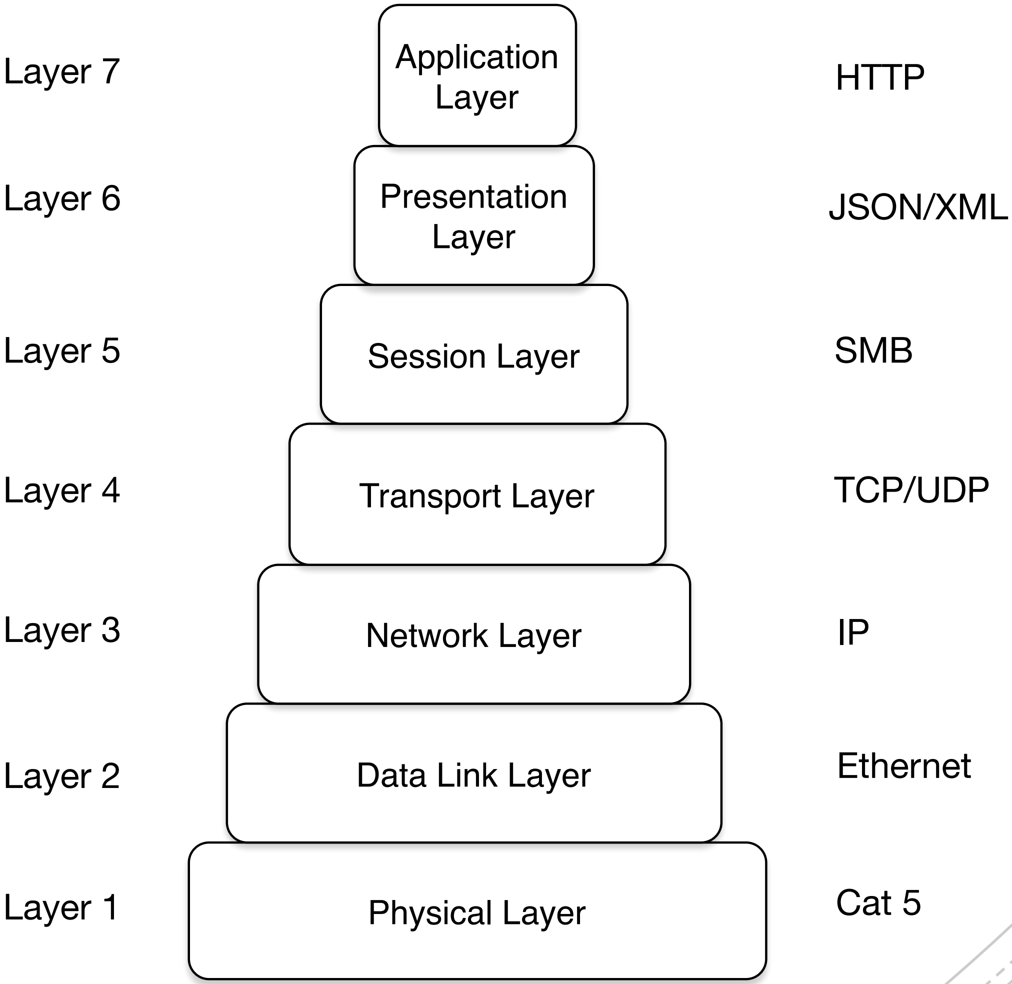
When?

- Anytime you have a PCAP
    - Incident Response
    - Troubleshooting
    - Application Security Analysis

Basics

- Scapy works at the network layer.
- Lets review the 7 OSI Model

OSI Model



Layer 7	Application Layer	HTTP
Layer 6	Presentation Layer	JSON/XML
Layer 5	Session Layer	SMB
Layer 4	Transport Layer	TCP/UDP
Layer 3	Network Layer	IP
Layer 2	Data Link Layer	Ethernet
Layer 1	Physical Layer	Cat 5

Scapy

- Scapy is a Python module, but also a stand alone application.
- pip3 install scapy

Scapy

- Scapy works at liked the layers of the OSI model.
- You go from Layer 2 up.
- To find an IP address you start at layer 3.
- To print the source IP : print(pkt[IP].src)
- Destination IP: print(pkt[IP].dst)

Scapy

- What layer of the OSI Model is DNS?
- To print a DNS record you would check to see if the packet has the layer.
- Then print the lookup out.
- `if pkt.haslayer(DNS)`

```
 print((pkt.getlayer(DNS).qd.qname)
 .decode("utf-8")
```

Hands On!

- Lets get started with code.
- First we need to create a pcap file.
  - `sudo tcpdump -c2000 -w example.pcap`
  - Open Chromium and browse the web for a few minutes.

Code Snippets

- You can download examples here: http://bit.ly/pficexamples
- I find code in slides hard to follow.

Coding

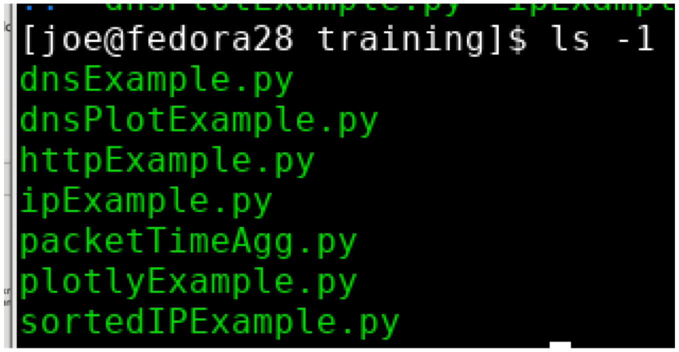
- You can download a complete swiss army knife called PacketExaminer which uses all of these examples.
- wget http://bit.ly/packetex
- Use your favorite editor:
  - vi
  - nano (sudo yum install nano -y)
  - less packetexaminer.py

Coding

- If you want to download all of the examples in advance:

http://bit.ly/pficcode

```
unzip pficcode
cd packetexaminer-master/training/
```



```
[joe@fedora28 training]$ ls -1
dnsExample.py
dnsPlotExample.py
httpExample.py
ipExample.py
packetTimeAgg.py
plotlyExample.py
sortedIPExample.py
```

Imports

```
from scapy.all import *
from prettytable import PrettyTable
from collections import Counter, defaultdict
```

Read the file

- Next we tell scapy to read the pcap file
- `packets = rdpcap('example.pcap’)`

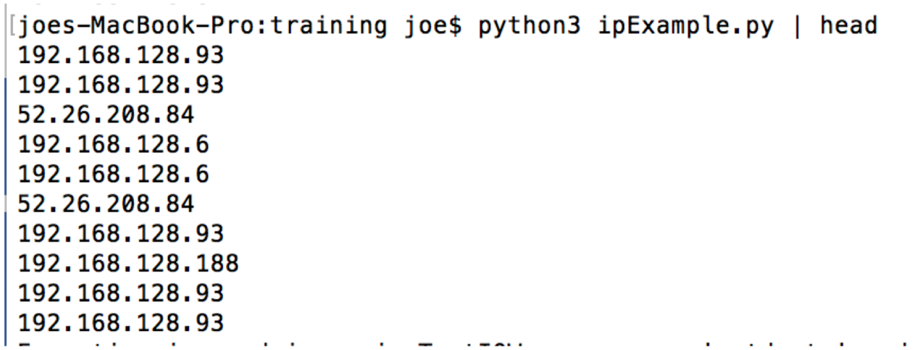
Process the Packets

- To just read each packet and print the source use a loop.

```
for pkt in packets:
    if IP in pkt:
        try:
            print(pkt[IP].src)
        except:
            pass
```

Try it!

- Read your PCAP and print out each IP.
- Spend about 5 minutes.
- http://bit.ly/pficex1



```
joes-MacBook-Pro:training joe$ python3 ipExample.py | head
192.168.128.93
192.168.128.93
52.26.208.84
192.168.128.6
192.168.128.6
52.26.208.84
192.168.128.93
192.168.128.188
192.168.128.93
192.168.128.93
```

Count

- Obviously that was a bad way to view data.
- Lets add it to a list then run it through a counter

```
srcIP=[]
for pkt in packets:
    if IP in pkt:
        try:
            srcIP.append(pkt[IP].src)
        except:
            pass
```

Analyze the Data

- Use a counter to create a count

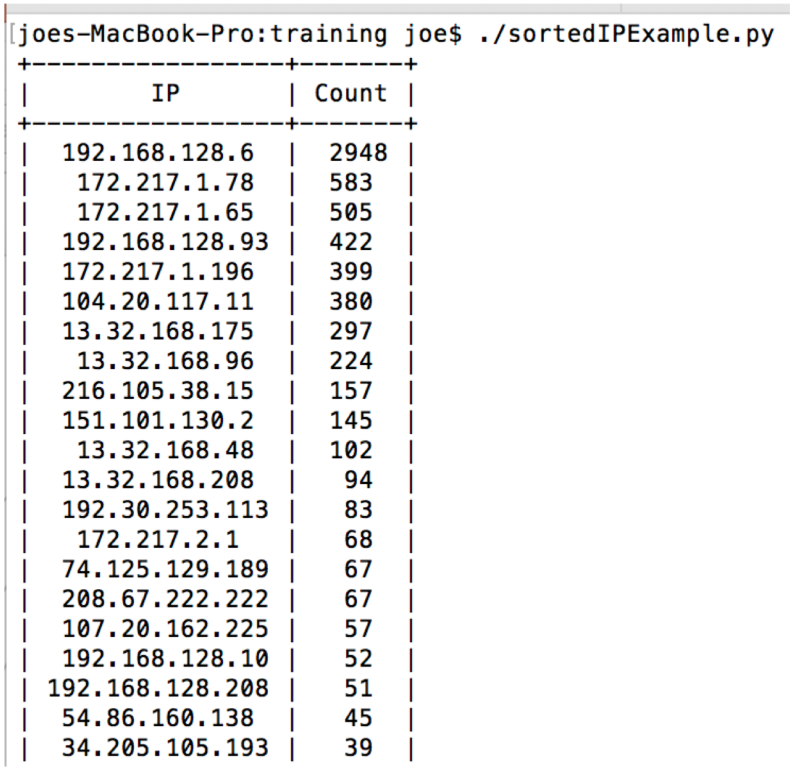
```
cnt=Counter()

for ip in srcIP:
    cnt[ip] += 1
```

PrettyTable

- A favorite Python module of mine is PrettyTable.
- We'll use another loop to create a sorted table of results.

```
table= PrettyTable(["IP", "Count"])
for ip, count in cnt.most_common():
   table.add_row([ip, count])
print(table)
```

PrettyTable

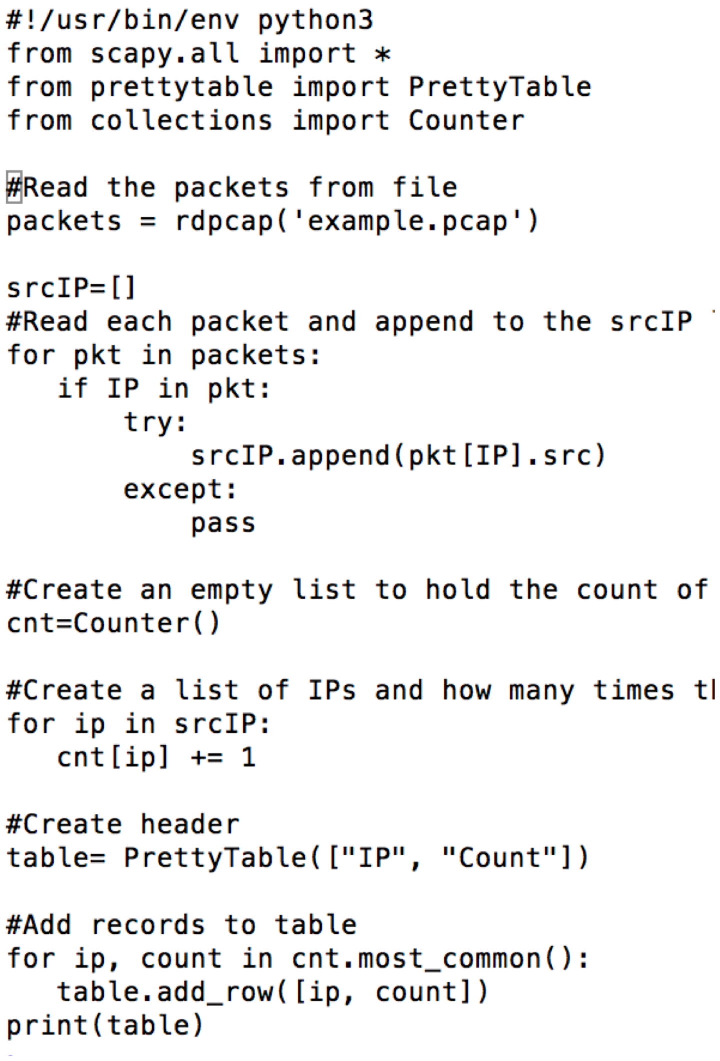


```
[joes-MacBook-Pro:training joe$ ./sortedIPExample.py
+-----------------+--------+
|        IP       | Count  |
+-----------------+--------+
|  192.168.128.6  |  2948  |
|   172.217.1.78  |  583   |
|   172.217.1.65  |  505   |
|  192.168.128.93 |  422   |
|  172.217.1.196  |  399   |
|  104.20.117.11  |  380   |
|  13.32.168.175  |  297   |
|   13.32.168.96  |  224   |
|  216.105.38.15  |  157   |
|  151.101.130.2  |  145   |
|   13.32.168.48  |  102   |
|  13.32.168.208  |   94   |
|  192.30.253.113 |   83   |
|   172.217.2.1   |   68   |
|  74.125.129.189 |   67   |
|  208.67.222.222 |   67   |
|  107.20.162.225 |   57   |
|  192.168.128.10 |   52   |
| 192.168.128.208 |   51   |
|  54.86.160.138  |   45   |
|  34.205.105.193 |   39   |
```

http://bit.ly/pficex2

Try it!

Print a table of results.



```
#!/usr/bin/env python3
from scapy.all import *
from prettytable import PrettyTable
from collections import Counter

#Read the packets from file
packets = rdpcap('example.pcap')

srcIP=[]
#Read each packet and append to the srcIP
for pkt in packets:
   if IP in pkt:
       try:
           srcIP.append(pkt[IP].src)
       except:
           pass

#Create an empty list to hold the count of
cnt=Counter()

#Create a list of IPs and how many times t
for ip in srcIP:
   cnt[ip] += 1

#Create header
table= PrettyTable(["IP", "Count"])

#Add records to table
for ip, count in cnt.most_common():
   table.add_row([ip, count])
print(table)
```

Plotting Data

- I find graphs and charts to be much better tools for looking at network data than a simple table.
- In the past we use Matplotlib.
  - Slow, picky and unattractive.
- Plotly fixes all of this.

Plotly

- To install run:

```
pip3 install plotly
```

- Then just add the import in your program.

```
from scapy.all import *
from collections import Counter
import plotly
```

Building

- Make a copy of your previous script, and we will just add on to it.
- After printing the table add two new lists for X and Y data.

```
xData=[]
yData=[]
```

- Then loop through the IP and X and Y data

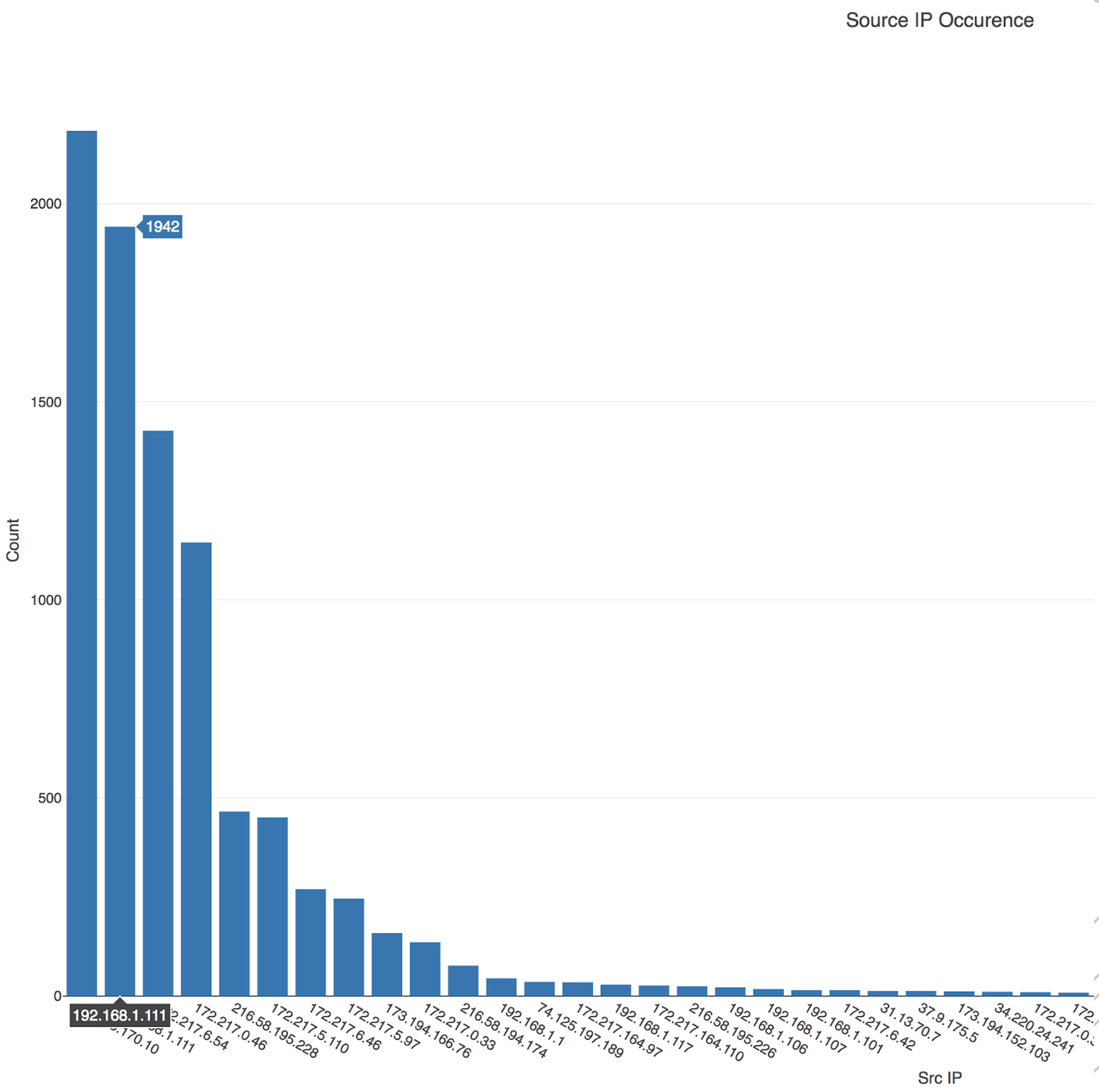
```
for ip, count in cnt.most_common():
   xData.append(ip)
   yData.append(count)
```

Plot the Plotly

- Plotly is a great tool, it opens your system web browser to create interactive graphs.

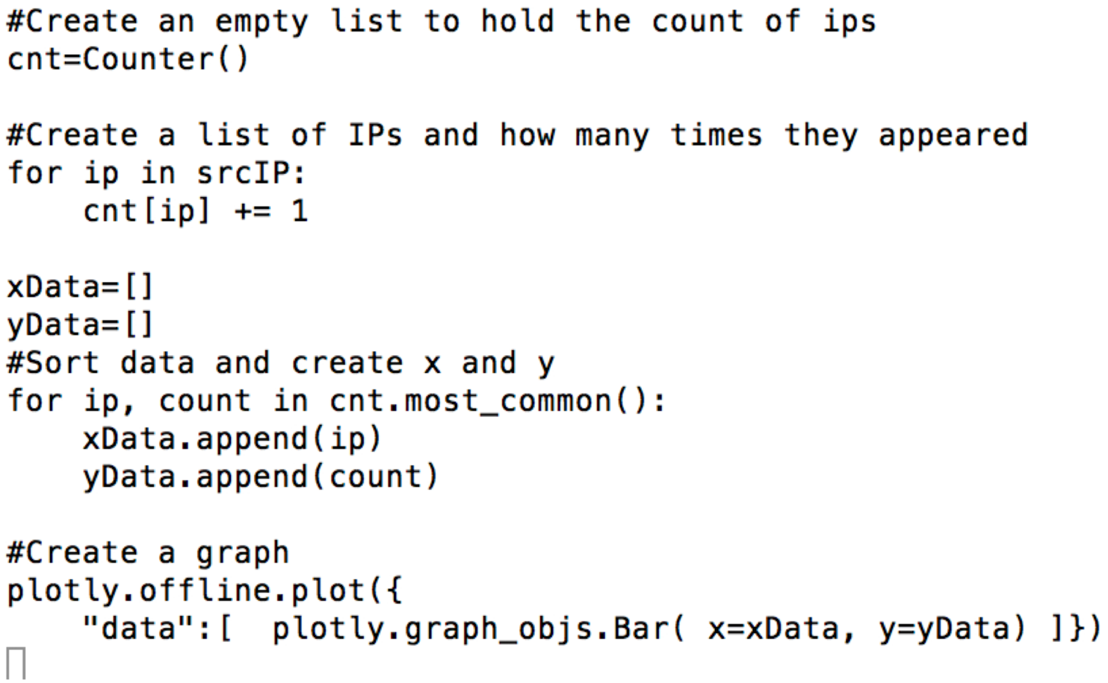
```
plotly.offline.plot({
  "data":[  plotly.graph_objs.Bar( x=xData, y=yData) ]
})
```

Plot the Plotly



Try it!

Add the following to your script.



```
#Create an empty list to hold the count of ips
cnt=Counter()

#Create a list of IPs and how many times they appeared
for ip in srcIP:
    cnt[ip] += 1

xData=[]
yData=[]
#Sort data and create x and y
for ip, count in cnt.most_common():
    xData.append(ip)
    yData.append(count)

#Create a graph
plotly.offline.plot({
    "data":[  plotly.graph_objs.Bar( x=xData, y=yData) ]})
```

http://bit.ly/pficex3

Refine it

Add Labels

```
plotly.offline.plot({
   "data":[plotly.graph_objs.Bar(x=xData, y=yData)],
   "layout":plotly.graph_objs.Layout(
        title="Source IP Occurrence",
       xaxis=dict(title="Src IP"),
       yaxis=dict(title="Count"))})
```

HTTP URLs

- URLs can be scraped from the packets.
- To get the uri use:

```
(pkt[http.HTTPRequest].Path).decode("utf-8")
```

- To get the host use:

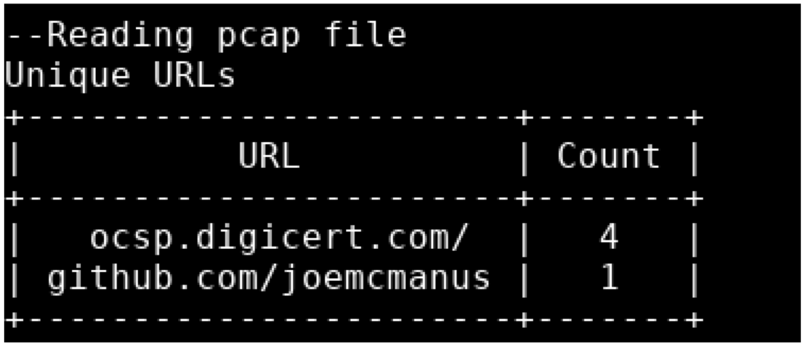
```
(pkt[http.HTTPRequest].Host).decode("utf-8")
```

HTTP URLs

```
if http.HTTPRequest in pkt:
  uri=(pkt[http.HTTPRequest].Path).decode("utf-8")
  host=(pkt[http.HTTPRequest].Host).decode("utf-8")
  url=host+uri
  print(url)
```

Try it!

- http://bit.ly/pficex4



```
--Reading pcap file
Unique URLs
+-----------------------+-------+
|          URL          | Count |
+-----------------------+-------+
|   ocsp.digicert.com/  |   4   |
| github.com/joemcmanus |   1   |
+-----------------------+-------+
```

Plot DNS

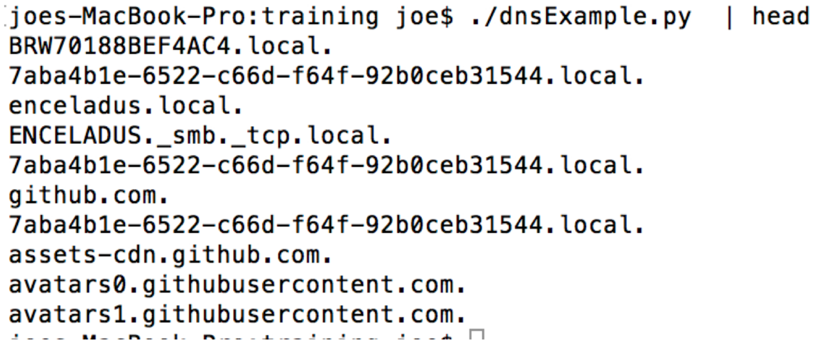
- With a simple change we can plot DNS lookups.
- You can also print a table of DNS lookups.

Plot DNS

```
for pkt in packets:
 if IP in pkt:
   if pkt.haslayer(DNS) and pkt.getlayer(DNS).qr == 0:
     lookup=(pkt.getlayer(DNS).qd.qname).decode("utf-8")
     print(lookup)
```

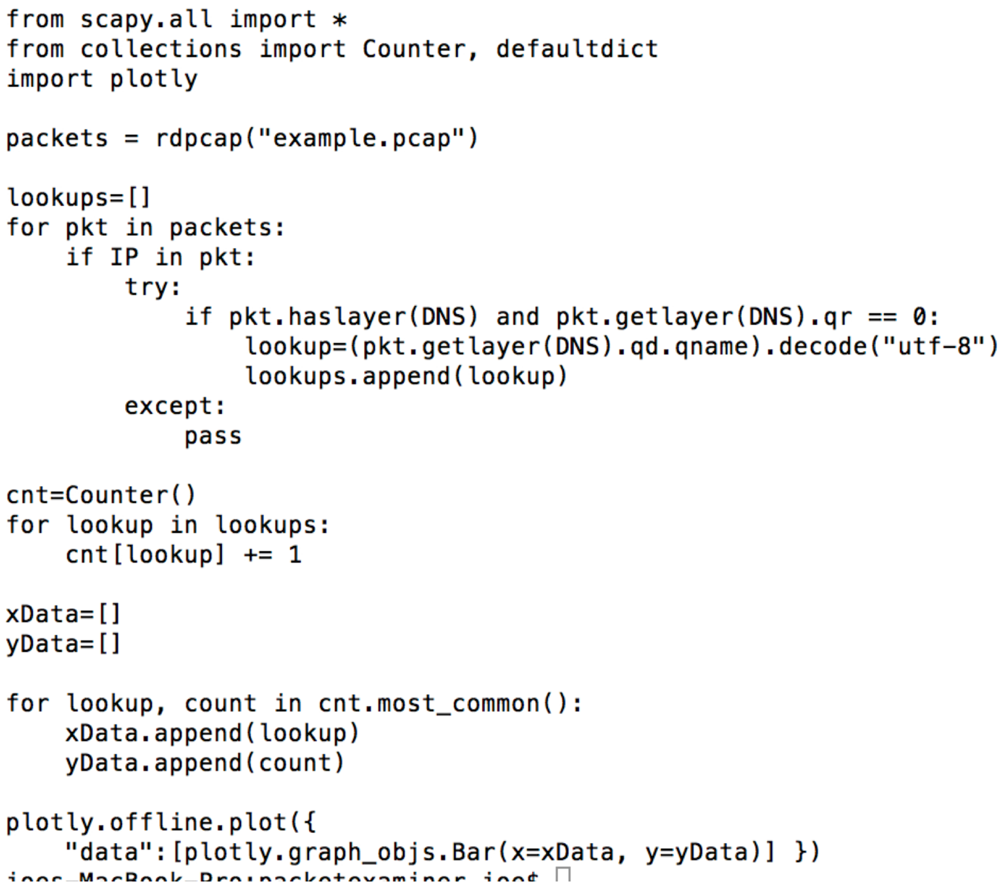
Try it!

- http://bit.ly/pficex5



```
joes-MacBook-Pro:training joe$ ./dnsExample.py  | head
BRW70188BEF4AC4.local.
7aba4b1e-6522-c66d-f64f-92b0ceb31544.local.
enceladus.local.
ENCELADUS._smb._tcp.local.
7aba4b1e-6522-c66d-f64f-92b0ceb31544.local.
github.com.
7aba4b1e-6522-c66d-f64f-92b0ceb31544.local.
assets-cdn.github.com.
avatars0.githubusercontent.com.
avatars1.githubusercontent.com.
joes-MacBook-Pro:training joe$ 
```

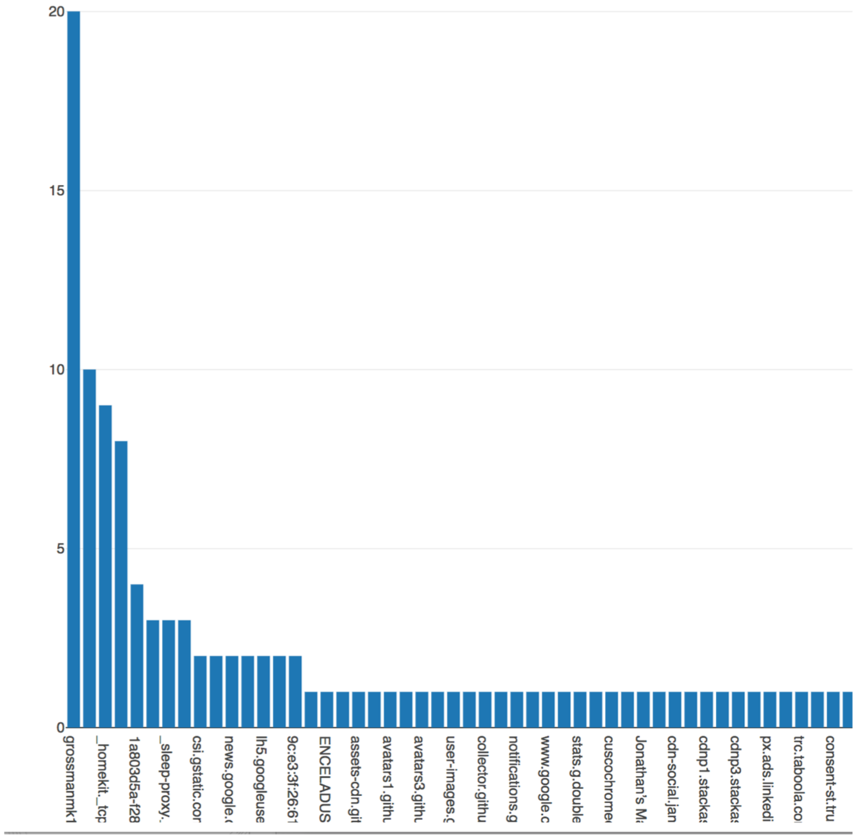
Plot DNS



```
from scapy.all import *
from collections import Counter, defaultdict
import plotly

packets = rdpcap("example.pcap")

lookups=[]
for pkt in packets:
    if IP in pkt:
        try:
            if pkt.haslayer(DNS) and pkt.getlayer(DNS).qr == 0:
                lookup=(pkt.getlayer(DNS).qd.qname).decode("utf-8")
                lookups.append(lookup)
        except:
            pass

cnt=Counter()
for lookup in lookups:
    cnt[lookup] += 1

xData=[]
yData=[]

for lookup, count in cnt.most_common():
    xData.append(lookup)
    yData.append(count)

plotly.offline.plot({
    "data":[plotly.graph_objs.Bar(x=xData, y=yData)] })
joes-MacBook-Pro:packetexaminer joe$ □
```

Try it!

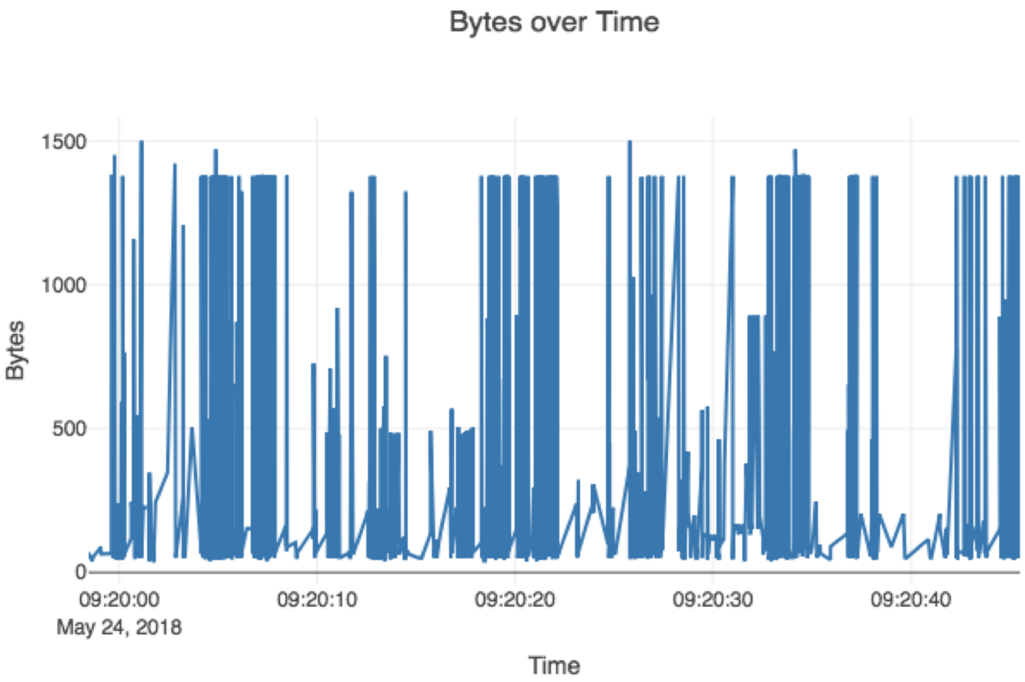
- http://bit.ly/pficex6



Time Series

- You will often want to plot data over time.
- The first thought is to just look at the length of each packet.
- The problem with that is you almost always plot the maximum MTU (usually 1500)

Time Series


Bytes over Time
Bytes
1500
1000
500
0
09:20:00
May 24, 2018
09:20:10
09:20:20
09:20:30
09:20:40
Time

Time Series

- To get around this you want to bin packets over time.
- The package Pandas makes this incredibly easy for us.

Time Series

- Start with the same imports, plus pandas.

```
from scapy.all import *
import plotly
from datetime import datetime
import pandas as pd
```

Time Series

- PCAPs have time in epoch, we need to confer to human readable times.

```
#Read each packet and append to the lists.
for pkt in packets:
   if IP in pkt:
     try:
        pktBytes.append(pkt[IP].len)
        pktTime=datetime.fromtimestamp(pkt.time)
        pktTimes.append(pktTime.strftime("%Y-%m-%d
%H:%M:%S.%f"))

     except:
        pass
```

Time Series

- Next convert the list to a pandas time series.

```
bytes = pd.Series(pktBytes).astype(int)
```

Time Series

- Next convert the timestamp to a date_time for Pandas.

```
times = pd.to_datetime(pd.Series(pktTimes).astype(str),
errors='coerce')
```

Time Series

- Create a Pandas data frame

```
df  = pd.DataFrame({"Bytes": bytes, "Times":times})
```

Time Series

- Create a Pandas timestamp

```
df = df.set_index('Times')
```

Time Series

- Resample the data to 2 second bins

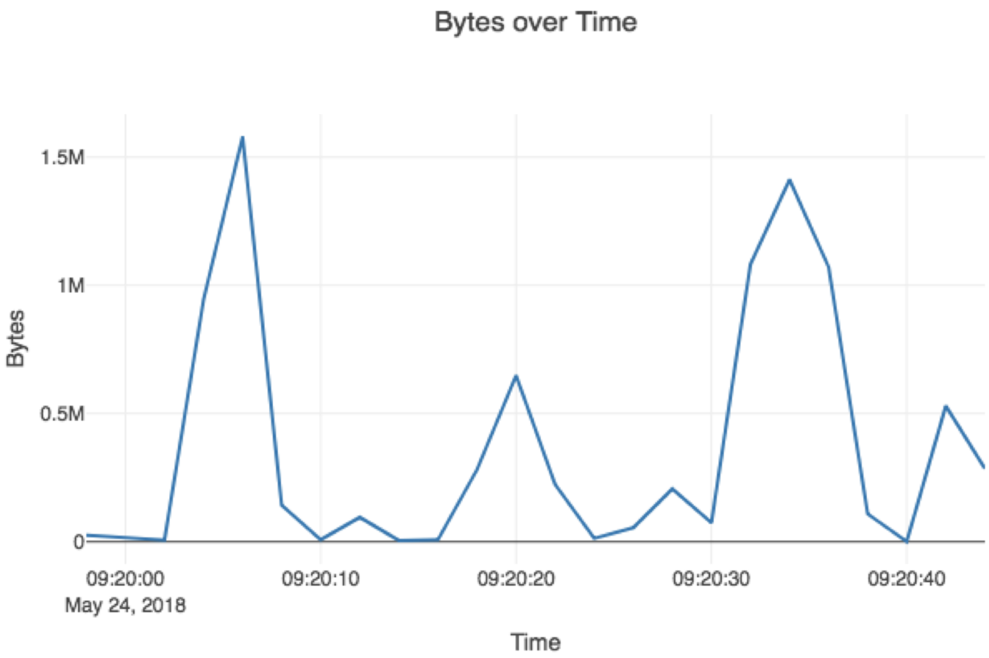
```
df2=df.resample('2S').sum()
print(df2)
```

Time Series

- Print the results

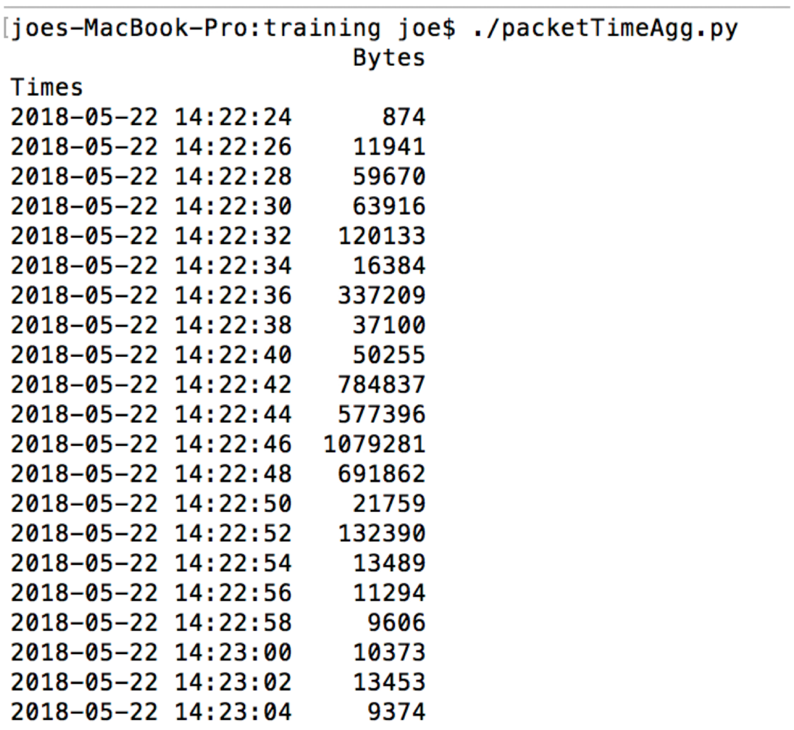
```
plotly.offline.plot({
 "data":[plotly.graph_objs.Scatter(x=df2.index,
        y=df2['Bytes'])],
    "layout":plotly.graph_objs.Layout(
        title="Bytes over Time ",
        xaxis=dict(title="Time"),
        yaxis=dict(title="Bytes"))})
```

Time Series


Bytes over Time
Bytes
1.5M
1M
0.5M
0
09:20:00
May 24, 2018
09:20:10
09:20:20
09:20:30
09:20:40
Time

Try it!

- http://bit.ly/pficex7



```
[joes-MacBook-Pro:training joe$ ./packetTimeAgg.py
                       Bytes
Times
2018-05-22 14:22:24      874
2018-05-22 14:22:26    11941
2018-05-22 14:22:28    59670
2018-05-22 14:22:30    63916
2018-05-22 14:22:32   120133
2018-05-22 14:22:34    16384
2018-05-22 14:22:36   337209
2018-05-22 14:22:38    37100
2018-05-22 14:22:40    50255
2018-05-22 14:22:42   784837
2018-05-22 14:22:44   577396
2018-05-22 14:22:46  1079281
2018-05-22 14:22:48   691862
2018-05-22 14:22:50    21759
2018-05-22 14:22:52   132390
2018-05-22 14:22:54    13489
2018-05-22 14:22:56    11294
2018-05-22 14:22:58     9606
2018-05-22 14:23:00    10373
2018-05-22 14:23:02    13453
2018-05-22 14:23:04     9374
```

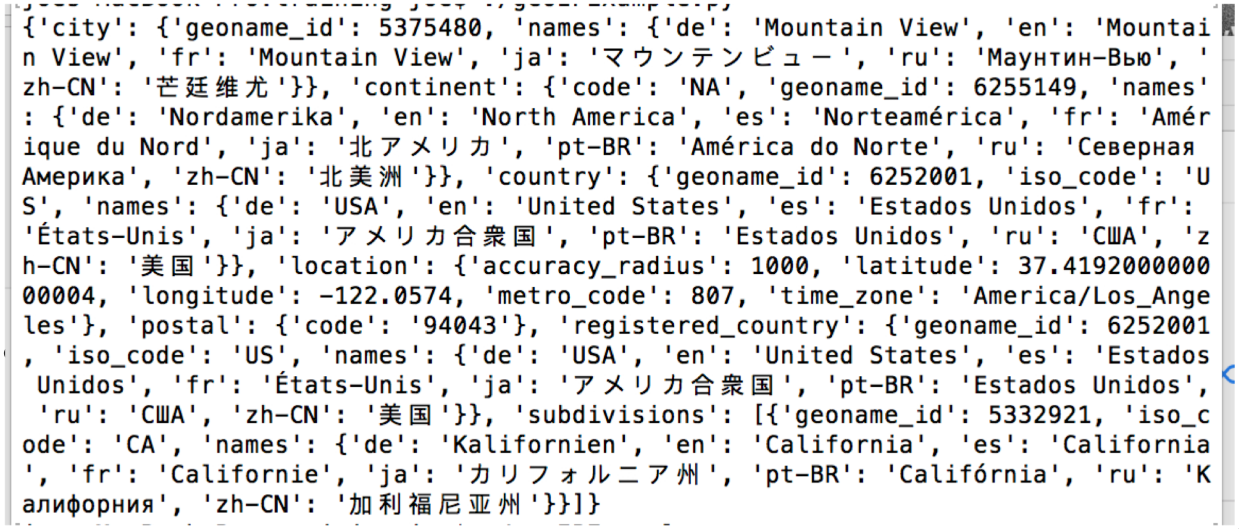
GeoIP

- It can be helpful to batch resolve locations in your data.

```
pip3 install maxminddb-geolite2
```

GeoIP

- The data is in JSON format.



```
{'city': {'geoname_id': 5375480, 'names': {'de': 'Mountain View', 'en': 'Mountai
n View', 'fr': 'Mountain View', 'ja': 'マウンテンビュー', 'ru': 'Маунтин-Вью', '
zh-CN': '芒廷维尤'}}, 'continent': {'code': 'NA', 'geoname_id': 6255149, 'names'
: {'de': 'Nordamerika', 'en': 'North America', 'es': 'Norteamérica', 'fr': 'Amér
ique du Nord', 'ja': '北アメリカ', 'pt-BR': 'América do Norte', 'ru': 'Северная
Америка', 'zh-CN': '北美洲'}}, 'country': {'geoname_id': 6252001, 'iso_code': 'U
S', 'names': {'de': 'USA', 'en': 'United States', 'es': 'Estados Unidos', 'fr':
'États-Unis', 'ja': 'アメリカ合衆国', 'pt-BR': 'Estados Unidos', 'ru': 'США', 'z
h-CN': '美国'}}, 'location': {'accuracy_radius': 1000, 'latitude': 37.4192000000
00004, 'longitude': -122.0574, 'metro_code': 807, 'time_zone': 'America/Los_Ange
les'}, 'postal': {'code': '94043'}, 'registered_country': {'geoname_id': 6252001
, 'iso_code': 'US', 'names': {'de': 'USA', 'en': 'United States', 'es': 'Estados
 Unidos', 'fr': 'États-Unis', 'ja': 'アメリカ合衆国', 'pt-BR': 'Estados Unidos',
 'ru': 'США', 'zh-CN': '美国'}}, 'subdivisions': [{'geoname_id': 5332921, 'iso_c
ode': 'CA', 'names': {'de': 'Kalifornien', 'en': 'California', 'es': 'California
', 'fr': 'Californie', 'ja': 'カリフォルニア州', 'pt-BR': 'Califórnia', 'ru': 'K
алифорния', 'zh-CN': '加利福尼亚州'}}]}
```

GeoIP

- Add a new imports

```
from geoip import geolite2
import json
```

GeoIP

- Add a new imports

```
from geolite2 import geolite2
import json
```

GeoIP

- Access the data

```
reader = geolite2.reader()
match = reader.get(IP)
country=match['country']['names']['en']
```

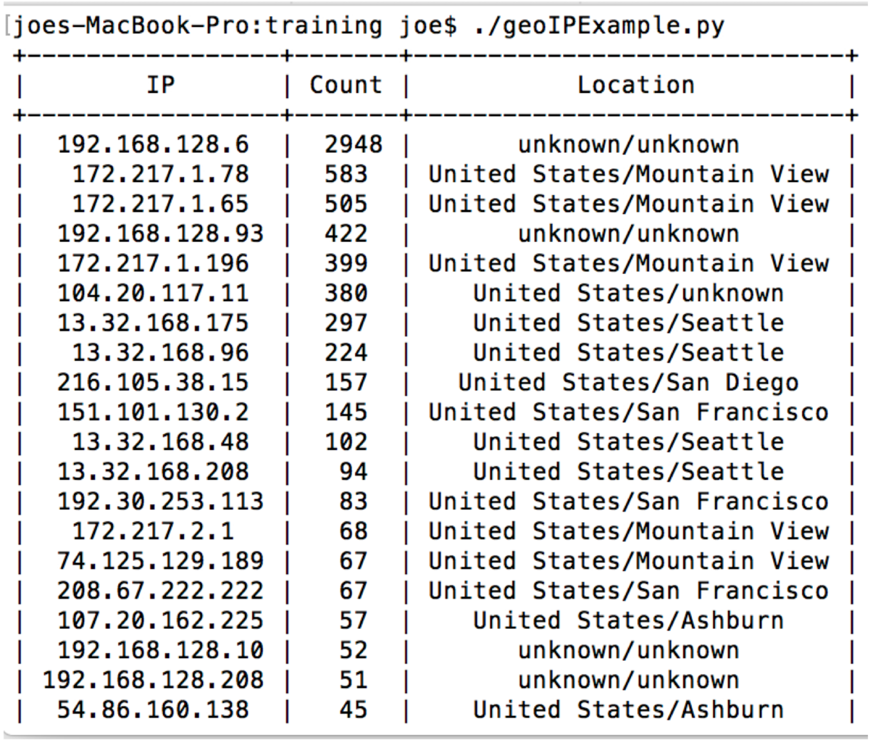
GeoIP

- Use a lot of try/except to handle issues.

```
if match:
        try:
            country=match['country']['names']['en']
        except:
            country="unknown"
        try:
            city=match['city']['names']['en']
        except:
            city="unknown"
    else:
        country="unknown"
        city="unknown"
```

Try it!

- Add location to your script.
- http://bit.ly/pficex8

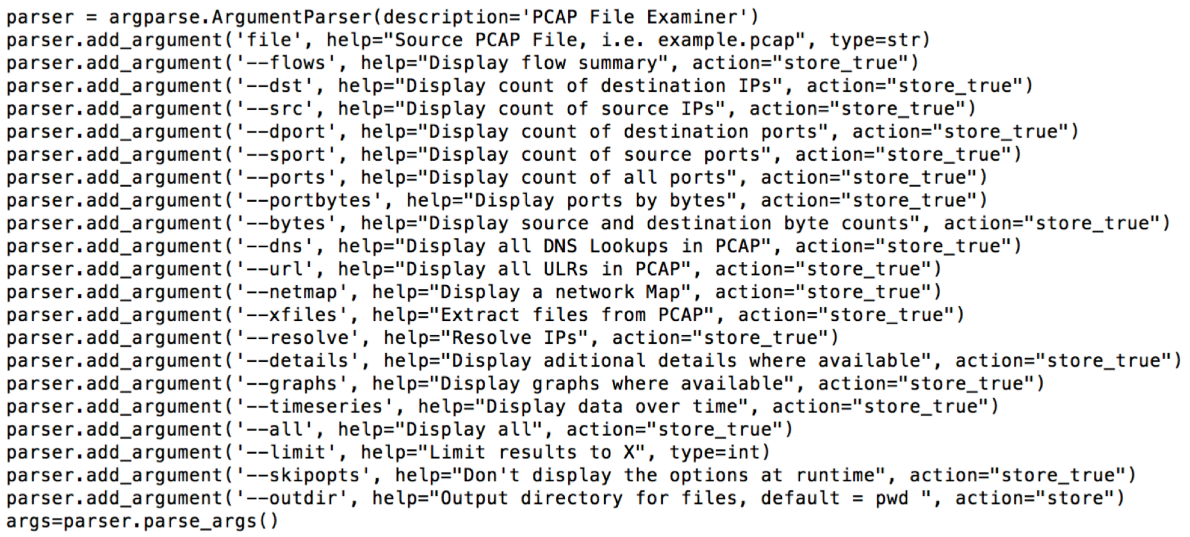


```
[joes-MacBook-Pro:training joe$ ./geoIPExample.py
+-----------------+-------+-----------------------------+
|        IP       | Count |           Location          |
+-----------------+-------+-----------------------------+
|  192.168.128.6  |  2948 |       unknown/unknown       |
|   172.217.1.78  |  583  | United States/Mountain View |
|   172.217.1.65  |  583  | United States/Mountain View |
|  192.168.128.93 |  422  |       unknown/unknown       |
|  172.217.1.196  |  399  | United States/Mountain View |
|  104.20.117.11  |  380  |    United States/unknown    |
|  13.32.168.175  |  297  |    United States/Seattle    |
|   13.32.168.96  |  224  |    United States/Seattle    |
|  216.105.38.15  |  157  |   United States/San Diego   |
|  151.101.130.2  |  145  | United States/San Francisco |
|   13.32.168.48  |  102  |    United States/Seattle    |
|  13.32.168.208  |   94  |    United States/Seattle    |
|  192.30.253.113 |   83  | United States/San Francisco |
|   172.217.2.1   |   68  | United States/Mountain View |
|  74.125.129.189 |   67  | United States/Mountain View |
|  208.67.222.222 |   67  | United States/San Francisco |
|  107.20.162.225 |   57  |    United States/Ashburn    |
|  192.168.128.10 |   52  |       unknown/unknown       |
| 192.168.128.208 |   51  |       unknown/unknown       |
|  54.86.160.138  |   45  |    United States/Ashburn    |
```

Tips

- I hate hardcoding filenames.
- You create a parser object and add options.
- `parser=argparse.ArgumentParser(description='Example Command Line Parser')`
- `parser.add_argument(‘filename', action="store")`
- For troubleshooting, use:
- `print(parser.parse_args())`

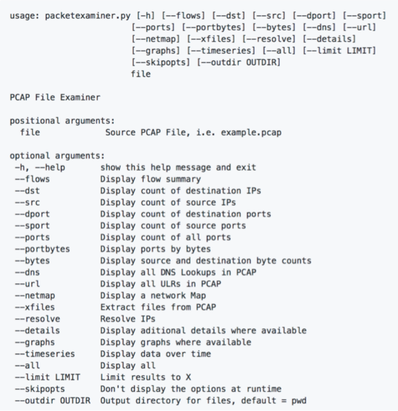
Tips



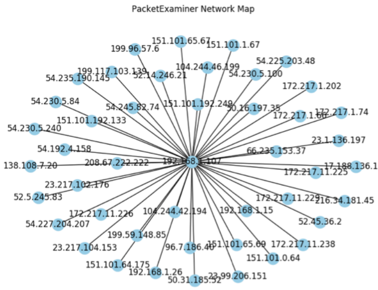
```
parser = argparse.ArgumentParser(description='PCAP File Examiner')
parser.add_argument('file', help="Source PCAP File, i.e. example.pcap", type=str)
parser.add_argument('--flows', help="Display flow summary", action="store_true")
parser.add_argument('--dst', help="Display count of destination IPs", action="store_true")
parser.add_argument('--src', help="Display count of source IPs", action="store_true")
parser.add_argument('--dport', help="Display count of destination ports", action="store_true")
parser.add_argument('--sport', help="Display count of source ports", action="store_true")
parser.add_argument('--ports', help="Display count of all ports", action="store_true")
parser.add_argument('--portbytes', help="Display ports by bytes", action="store_true")
parser.add_argument('--bytes', help="Display source and destination byte counts", action="store_true")
parser.add_argument('--dns', help="Display all DNS Lookups in PCAP", action="store_true")
parser.add_argument('--url', help="Display all ULRs in PCAP", action="store_true")
parser.add_argument('--netmap', help="Display a network Map", action="store_true")
parser.add_argument('--xfiles', help="Extract files from PCAP", action="store_true")
parser.add_argument('--resolve', help="Resolve IPs", action="store_true")
parser.add_argument('--details', help="Display aditional details where available", action="store_true")
parser.add_argument('--graphs', help="Display graphs where available", action="store_true")
parser.add_argument('--timeseries', help="Display data over time", action="store_true")
parser.add_argument('--all', help="Display all", action="store_true")
parser.add_argument('--limit', help="Limit results to X", type=int)
parser.add_argument('--skipopts', help="Don't display the options at runtime", action="store_true")
parser.add_argument('--outdir', help="Output directory for files, default = pwd ", action="store")
args=parser.parse_args()
```

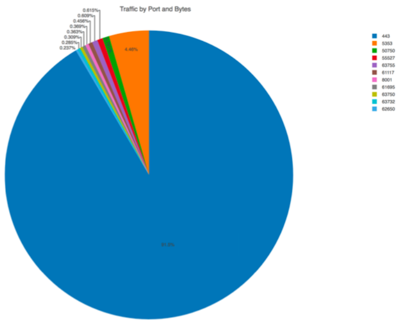
PacketExaminer

- I’ve bundled all of this and more in to a open source tool for DFIR called PacketExaminer.
- https://github.com/joemcmanus/packetexaminer



```
usage: packetexaminer.py [-h] [--flows] [--dst] [--src] [--dport] [--sport]
                         [--ports] [--portbytes] [--bytes] [--dns] [--url]
                         [--netmap] [--xfiles] [--resolve] [--details]
                         [--graphs] [--timeseries] [--all] [--limit LIMIT]
                         [--skipopts] [--outdir OUTDIR]
                         file

PCAP File Examiner

positional arguments:
  file              Source PCAP File, i.e. example.pcap

optional arguments:
 -h, --help        show this help message and exit
 --flows           Display flow summary
 --dst             Display count of destination IPs
 --src             Display count of source IPs
 --dport           Display count of destination ports
 --sport           Display count of source ports
 --ports           Display count of all ports
 --portbytes       Display ports by bytes
 --bytes           Display source and destination byte counts
 --dns             Display all DNS Lookups in PCAP
 --url             Display all ULRs in PCAP
 --netmap          Display a network Map
 --xfiles          Extract files from PCAP
 --resolve         Resolve IPs
 --details         Display aditional details where available
 --graphs          Display graphs where available
 --timeseries      Display data over time
 --all             Display all
 --limit LIMIT     Limit results to X
 --skipopts        Don't display the options at runtime
 --outdir OUTDIR   Output directory for files, default = pwd
```





Multi Platform Cloud Patching & Management

AUTOMOX

Questions?

- Any questions?
- mcmanus@automox.com
- www.linkedin.com/in/networkforensics/
- github.com/joemcmanus