Extracting the payload from a pcap file using Python

Vera Worri Follow
Dec 25, 2016 · 9 min read

(I am working on mac Sierra, Python 2.7.12, and the Pycharm IDE).

I am in the process of making a sniffing app to pull redundant copies of submission forms or other Ethernet traffic. It is part of Unhackable Server Project.

I have already started the project with pyshark but I could only get the http headers, not the payloads so I had to switch tactics.

One of my earlier posts was about executing terminal commands through python. The reason I had need of this was because I switched from using

Pyshark to sniff my network to going straight to the source; tshark. I just wanted to make it a but more elegant while getting as much information as possible.

```
import os


os.system("tshark  -T fields -e  data.data -e frame.time -w
Eavesdrop_Data.pcap > Eavesdrop_Data.txt -F pcap -c 1000")
```

I need to capture the packets being posted to the server.

> *n.b.: You can get in trouble if you use this to capture information that is not yours*

When you run this, it saves two files in the directory, a Pcap file and a text file after it captures 1000 packets. The output is a time stamp and whatever data is captured.

My goal now is to follow to TCP stream and extract the files in the packets.

> *To check the packets you saved while writing this code, I recommend you download Wireshark and give it root access by running this:* /Applications/Wireshark.app/Contents/MacOS/Wireshark *in your terminal.*

I am using this article from Codingsec.net as my guide but my needs are a bit simpler and more straight forward. I also have to be careful because the code I am referencing is written in Python 3 and I am using Python 2.

I downloaded scapy using the terminal. I had to restart PyCharm to get it to reload the interpreter and get it to import into the code.

I started to look over the reference code and adopt some of it.

```
import os
import pcapy as p
from scapy.all import *
a = " "
os.system("tshark  -T fields  -e frame.time -e  data.data -w
Eavesdrop_Data.pcap > Eavesdrop_Data.txt -F pcap -c 1000")

data = "Eavesdrop_Data.pcap"
a = rdpcap(data)
```

So, this is gleaned form the afore mentioned code. This was just to dip my toe and see exactly what the different modules were doing.

Firstly, I had to re-download scapy for some reason it didn't properly download earlier so I uninstalled it and reinstalled it using the terminal. Then I had to download Pcapy which could not run without dumbnet. Ugh! I spent the whole day just finding and installing packages. But soon, I got this to work. The output looked like this:

Woot! No errors!

> *N.B. I installed dumbnet with my terminal from git.*

Now, I moved to the next few lines of the refference code:

```
from scapy.all import *
a = " "
os.system("tshark  -T fields  -e frame.time -e  data.data -w
Eavesdrop_Data.pcap > Eavesdrop_Data.txt -F pcap -c 1000")


data = "Eavesdrop_Data.pcap"
a = rdpcap(data)
sessions = a.sessions()
print sessions
```

and the output is:

The dict keeps going all the way

Woot! Still no error. I like to use the print command just to check what is happening and that the data remains intact.

Code:

```
from scapy.all import *
a = " "
os.system("tshark  -T fields  -e frame.time -e  data.data -w
Eavesdrop_Data.pcap > Eavesdrop_Data.txt -F pcap -c 1000")

data = "Eavesdrop_Data.pcap"
a = rdpcap(data)
sessions = a.sessions()
for session in sessions:
    print sessions
```

output:

Input:

```
from scapy.all import *
a = " "
#os.system("tshark  -T fields  -e frame.time -e  data.data -w
Eavesdrop_Data.pcap > Eavesdrop_Data.txt -F pcap -c 1000")


#I commented out the t-shark so i could just reuse the same data

data = "Eavesdrop_Data.pcap"
a = rdpcap(data)
sessions = a.sessions()
for session in sessions:
    http_payload = ""
    for packet in sessions[session]:
        print packet
```

Now, at this point I'm thinking how can I make sure that I am getting the payload from all the http requests (not https) I am getting? So I changed the

command line file and added another underneath it to parse out the http packets from the file captured in the first command line. I then printed out the result.

Code:

```
from scapy.all import *
data = "Eavesdrop_Data.pcap"
a = rdpcap(data)
os.system("tshark  -T fields -e _ws.col.Info -e http -e frame.time -e
"
          "data.data -w Eavesdrop_Data.pcap > Eavesdrop_Data.txt -c
1000")
os.system("tshark -r Eavesdrop_Data.pcap -Y http -w
Eavesdrop_Data_http.pcap")
sessions = a.sessions()
i = 1
for session in sessions:
    http_payload = ""
    for packet in sessions[session]:
        print packet
```

At this point, we are still only using scapy.

Output:

I did get some plain text which is what I am after!

So, now I must figure out how to isolate text/plain content types. I first put in a try and accept function inside my for loop to test if the http packets have a TCP. Then I matched source ports (sports) and destination ports (dports) to port 80. Port 80 is the default listening port for http.

My code now looks like this:

```
from scapy.all import *
data = "Eavesdrop_Data.pcap"
a = rdpcap(data)
#os.system("tshark  -T fields -e _ws.col.Info -e http -e frame.time -
e  "
#          "data.data -w Eavesdrop_Data.pcap > Eavesdrop_Data.txt -c
1000")
os.system("tshark -r Eavesdrop_Data.pcap -Y http -w
Eavesdrop_Data_http.pcap")
sessions = a.sessions()
i = 1
for session in sessions:
    http_payload = ""
    for packet in sessions[session]:
        try:
            if packet[TCP].dport == 80 or packet[TCP].sport == 80:
                print packet[TCP].payload
        except:
            pass
```

and I get this in my terminal:

It looks a lot less scary that before and as you can see, the plain text is still there.

I have to figure out how to get the http header so I can sift through the content types. If you scroll down in out example code, you will see that the author has created a class to handle this. His class is geared towards images.

I started by copying and pasting his function and adapting it to work for text (everywhere it says image, change it to text). This is what it looks like:

```
import re


def HTTPHeaders(http_payload):
        try:
            #isolate headers
            headers_raw =
http_payload[:http_payload.index("\r\n\r\n") + 2]
            regex = r"(?P&lt;'name&gt;.*?): (?
P&lt;value&gt;.*?)\r\n'"
            headers =  dict(re.findall(regex, headers_raw))
        except:
            return None
        if 'Content-Type' not in headers:
            return None
        return headers

def extractText(headers, http_payload):
        text = None
        text_type = None
        if 'text' in headers['Content-Type']:
            text_type = headers['Content-Type'].split("/")[1]
            text = http_payload[http_payload.index("\r\n\r\n")+4:]
            return text,text_type
```

This is my output:

None
None
None
None
None
None
None
None
None
None
None
…

…

all the way down.

As we have seen before, there should be headers in the output. I isolated the problem to the regular expression in the try function. Time to learn something new… again. What is the best way?

After scouring the internet, I found the most amazing api! I was able to test the regex given in the sample code with the “headers_raw”. In the meantime, I leaned out the code and just kept all the code I needed (got rid of the text_type functions).

Afterwards, I set out to find what was happening. I tested with print statements. I followed the data, printing it out in each step, each loop, each function and found the the problem was the headers_raw line. The error was that the substring could not be found. With this, I removed it and used part of a code generated from https://regex101.com/ (translated to python 2.7).

Now it looks like:

```
import re
import zlib

def HTTPHeaders(http_payload):
    try:
        # isolate headers
        matches = re.finditer(r'\r\n\r\n', http_payload)

        for matchNum, match in enumerate(matches):
            matchNum = matchNum + 1
```

```
            headers_raw = http_payload[:match.end()]
            print headers_raw
        regex = ur"/?P&lt;'name&gt;.*?/: /?P&lt;value&gt;.*?/\n"
        headers = dict(re.findall(regex, headers_raw, re.UNICODE))
        return headers
    except:
        return None
    if 'Content-Type' not in headers:
        return None
    return headers

def extractText(headers, http_payload):
        text = None
        try:
            if 'text/html' in headers['Content-Type']:
                text = http_payload[http_payload.index("\n\n")+4:]
                try:
                    if "Content-Encoding" in headers.keys():
                        if headers['Content-Encoding'] == "gzip":
                            text = zlib.decompress(text)
                    elif headers['Content-Encoding'] == "deflate":
                        text = zlib.decompress(text)
                except: pass
        except:
            return None
        return text
```

The output looks like:

The match worked perfectly but I soon found out that even though I solved my header_raw problem, something was up with the regex that parses the headers into dictionaries.

Again, I shed a tear because I have no clue about regexes. I rolled up my sleeves and got to work.

This was going to be a lot more complicated than just finding the headers. I did more research, googled and searched Stackoverflow as well as

Regex101 looking for a solution. Regex101 has a library of regexes users have created. Searching for json , I found this: (?:[\n]{0,1})(\w+)(?:\ *=\ *)([^\n]*)(?:[\n]{0,1})

When I put it in my code, as-is, it gave me this:

Ooops! It's supposed to be a dictionary form of :

If you look carefully at the output, you can see that there are some rogue \r values in there on top of that, the regex has cut out a lot of the information. If we add \r to our \n in the regex like this: **“(?:[\r\n]{0,1})(\w+)(?:\ *=\ *)([^\r\n]*)(?:[\r\n]{0,1})”**

and our output looks like this:

It's getting there....

Another adjustment I needed to make was to change the equals sign to a colon to adapt the regex to the format of my input (the original had = to signify key value pairs).

This:

```
(?:[\r\n]{0,1})(\w+)(?:\ *:\ *)([^\r\n]*)(?:[\r\n]{0,1})
```

Gives me this:

By this time, I understand how regex works a little so when I realized that the code was dropping the first part of anything hyphenated, I knew I needed to escape the hyphen and add a \w+ to add the word before the hyphen.

```
regex = ur"(?:[\r\n]{0,1})(\w+\-\w+)(?:\ *:\ *)([^\r\n]*)(?:[\r\n]
{0,1})"
```

which got me this:

If you look at the output, you will notice that now, it is ignoring everything that isn't hyphenated. So I needed to add an "or" in order to grab the

missing key/value combinations(or is denoted with a |).

```
regex = ur"(?:[\r\n]{0,1})(\w+\-\w+|\w+)(?:\ *:\ *)([^\r\n]*)(?:
[\r\n]{0,1})"
```

Woooot! It is not in order, but that does not matter to me. At this point, I went back to see if the original parsing for the headers_raw would work because the for loop for the matching seemed a bit cumbersome. It worked!

In the end, my code was made of two python files. Here they are:

Eavesdrop.py

```
import os
from scapy.all import *

from getHTTPHeaders import HTTPHeaders, extractText
```

```
data = "Eavesdrop_Data.pcap"
a = rdpcap(data)
os.system("tshark  -T fields -e _ws.col.Info -e http -e frame.time -e
"
          "data.data -w Eavesdrop_Data.pcap > Eavesdrop_Data.txt -c
1000")


sessions = a.sessions()
carved_texts = 1
for session in sessions:
    http_payload = ""
    for packet in sessions[session]:
        try:
            if packet[TCP].dport == 80 or packet[TCP].sport == 80:
                http_payload += str(packet[TCP].payload)
        except:
            pass
        headers = HTTPHeaders(http_payload)
    if headers is None:
        continue
    text = extractText(headers,http_payload)
    if text is not None:
         print (text)
```

getHTTPHeaders.py

```
import re
import zlib

def HTTPHeaders(http_payload):
```

```
    try:
        # isolate headers
        headers_raw = http_payload[:http_payload.index("\r\n\r\n") +
2]
        regex = ur"(?:[\r\n]{0,1})(\w+\-\w+|\w+)(?:\ *:\ *)([^\r\n]*)
(?:[\r\n]{0,1})"
        headers = dict(re.findall(regex, headers_raw, re.UNICODE))
        print headers
        return headers
    except:
        return None
    if 'Content-Type' not in headers:
        return None
    return headers

def extractText(headers, http_payload):
        text = None
        try:
            if 'text/plain' in headers['Content-Type']:
                text =
http_payload[http_payload.index("\r\n\r\n")+4:]
                try:
                    if "Accept-Encoding" in headers.keys():
                        if headers['Accept-Encoding'] == "gzip":
                            text = zlib.decompress(text,
16+zlib.MAX_WBITS)
                    elif headers['Content-Encoding'] == "deflate":
                        text = zlib.decompress(text)
                except: pass
        except:
            return None
        return text
```

In the data, there is only one plain text payload…. and the code found it!

This was a test in patience and a labor of love . I have been trying to solve this problem for a few months. Keep in mind that when I started, I had no network programming experience. I hope this helps someone out there. If you have any suggestions, please comment down or find me on my Github

Python Programming Hacking Wireshark Pcap

256 claps

WRITTEN BY

Vera Worri

Follow

See responses (4)

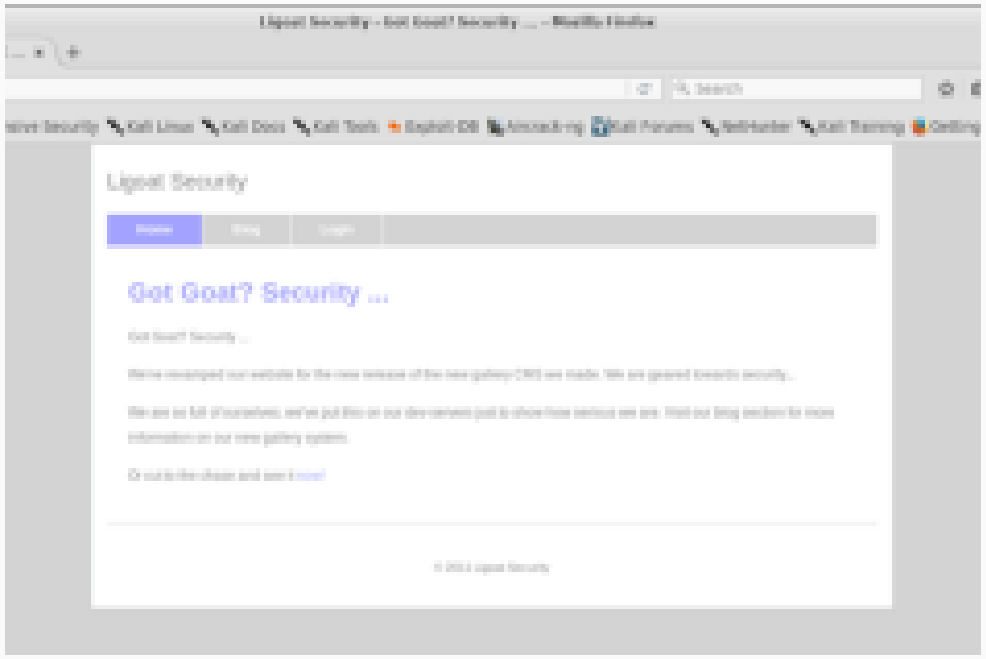
More From Medium

Related reads

VulnHub — Kioptrix: Level 3

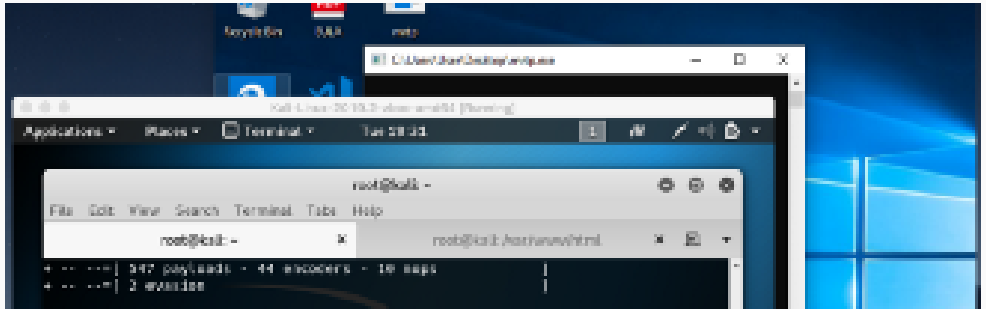
Mike Bond
Jun 1, 2018 · 14 min read

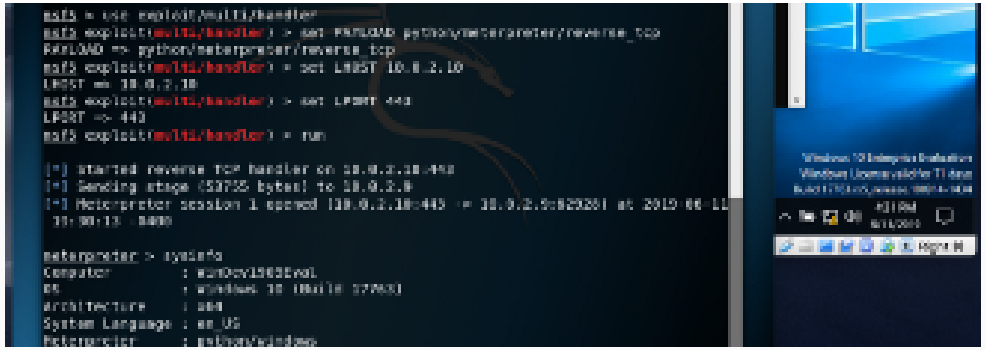
268

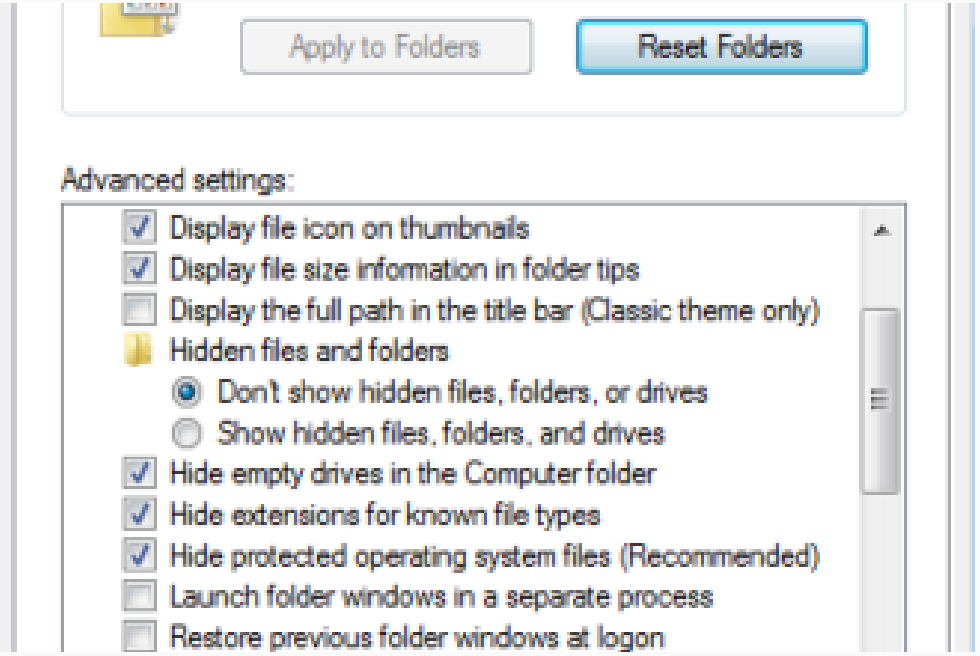


Related reads

Antivirus Evasion with Python







Marcelo Sacchetin in InfoSec Write-ups
Jun 11 · 6 min read ★

600

Related reads

Basic Static Analysis (Part 1)

Tstillz
Nov 19, 2018 · 11 min read ★

199