Laboratory: 6

Entropy Spoofing

Introduction

This report includes DDoS attack entropy detection spoofing. Here we perform entropy spoofing and compare the time-series graph for non-spoofed entropy files and spoofed entropy files and compare their differences. Various scripts are used in each stage to convert the files. Such as, “pcap2entropy.py” is used to generate the entropy file from pcap files, “entropy_spoofV3.py” is used to take histogram file and an attack record time file as input and generate spoofed histogram files, plot.py is used to plot the time-series graph and calculate_entropyV2.py is used to generate entropy files from the histogram files.

The basic working of how an entropy spoofing would be carried out in this report is: Perform some flood attack --> Capture the traffic and attack times --> Use pcap files to generate transcript files --> Use transcript files to generate histogram files --> Use the histogram files and attack record files to generate spoofed histogram files --> Use the spoofed histogram files to generate spoofed entropy files--> Plot the spoofed entropy files for comparing.

Setup and Process

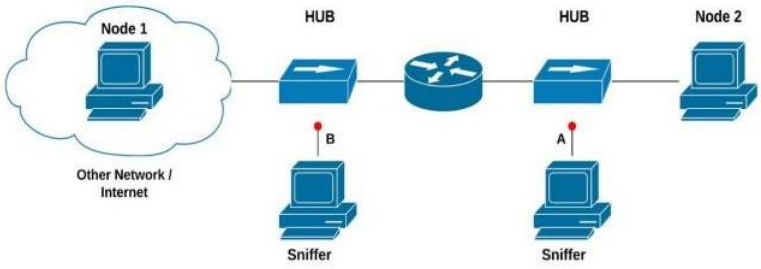


Figure1: Network set-up

Here, as shown above, node 1 (192.168.10.10) is machine from lab in clemson and node 2 (192.168.20.13) is machine from lab in charleston. All the machines are connected over the HUB and the red dots are the sniffing points (192.168.10.9). One Command & Control machine (192.168.10.111) is also connected over the sub-net that is used to observe and connect to the bots.

We first ssh to our Command & Control machine. First, we try to plot the graph for entropy file before spoofing and entropy file after spoofing. For now we do this on the entropy files which are pre-generated.

“outputTime0604.entr” is the entropy file before spoofing and “spoofed_1_outputTime0604.entr” is the entropy file after spoofing. We plot the time-series graph for both of them by using the following commands:

./plot.py -d outputTime0604.entr 1

./plot.py -d spoofed_1_outputTime0604.entr 1

You see the graphs generated in the Figure: 2 and Figure: 3

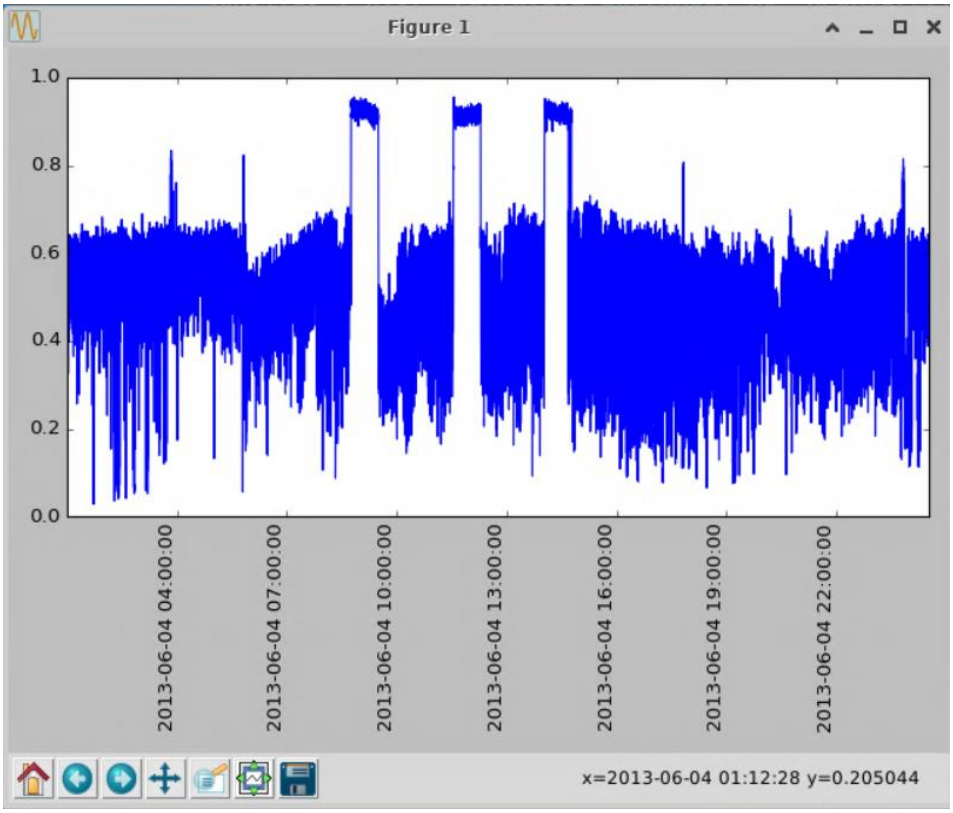


Figure 2: Entropy before spoofing

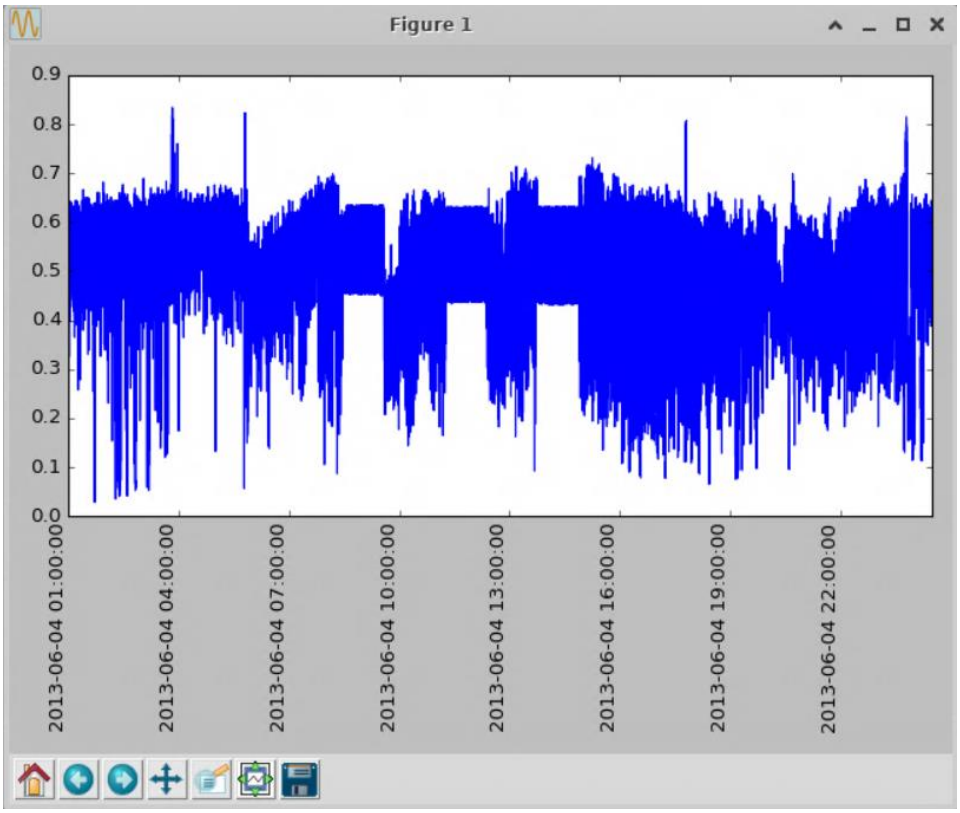


Figure 3: Entropy after spoofing

We can see how the attack could be detected easily in the case without spoofing from the figure 2, where we can say that everything over 0.9 would trigger the alarm that there is a DDoS attack. Where as, in figure 3, no such threshold can be used as a detection strategy since the graph doesn't show a spike during the attack. Hence making the entropy spoofing an efficient way to avoid and hide the detection of an attack.

Now, we move on to performing entropy spoofing by generating spoofed histogram file by ourself. It can take up to 20-25 mins in performing this.

For this, the attacks we performed in the previous labs are captured and the pcap file is used to generate the histogram file "outputTime0604.hist" Using "pcap2hist.py" script.

Once that's done. We now generate spoofed histogram file by using the following command (Figure: 4):

 # ./entropy_spoofV3.py outputTime0604.hist shriyas.hist.out complete_attack_times 1.25 -c 1

```
root@cnc:~/DDoS_Lab6/Appendix# ./entropy_spoofV3.py outputTime0604.hist shriyas.
hist.out complete_attack_times 1.25 -c 1
Namespace(FA=None, attack_times='complete_attack_times', columns=['1'], ent=Fals
e, file_name='outputTime0604.hist', margin=[0.15], output='shriyas.hist.out', sc
ale='1.25', spoofPacketCount=False)
Copying outputTime0604.hist to temp.txt
Performing attack number 1
Average cycle: 5.94748091603
Average time: 0.000912345648855
Performing attack number 2
Average cycle: 5.79839786382
Average time: 0.0010273271028
Performing attack number 3
Average cycle: 5.97771775827
Average time: 0.00102144767049
('It took ', 724, ' seconds')
End of process!!
root@cnc:~/DDoS_Lab6/Appendix#
```

Figure 4: Generating spoofed histogram

Our output is stored in "shriyas.hist". Now, this file is used to further generate the entropy file. The entropy file generated from "shriyas.hist' would be the spoofed entropy file which we will use for plotting the graph. The following command is used to generate the entropy file (figure: 5):

```
root@cnc:~/DDoS_Lab6/Appendix# ./calculate_entropyV2.py shriyas.hist shriyas
End of process.
('It took ', 682, ' seconds')
```

Figure 5: Generating spoofed entropy file

The output is now stored in shriyas.entr, which we use to generate the time-series graph (figure: 6) by using the following command:

/plot.py -d shriyas.entr 1

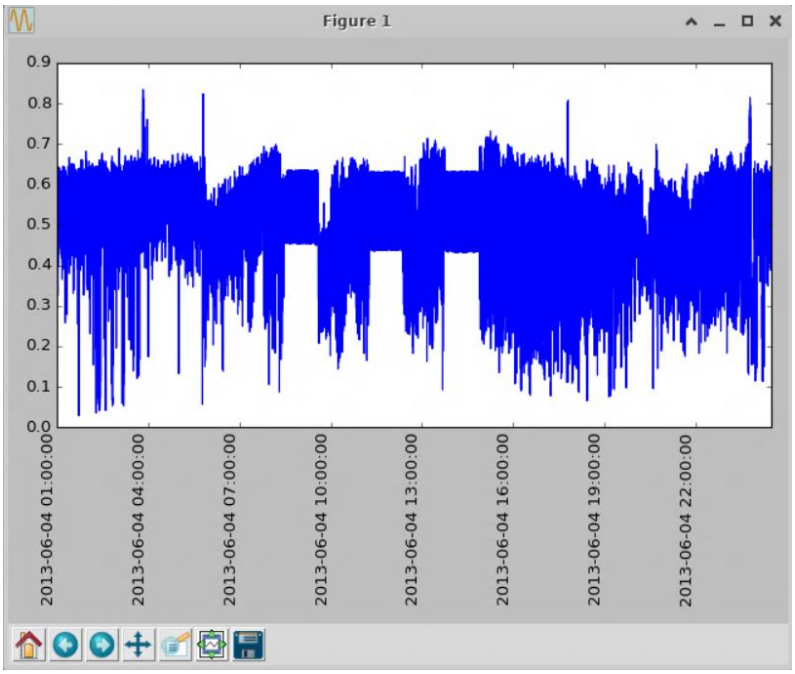


Figure 6: Spoofed Entropy

We can also plot the non-spoofed entropy file by generating entropy file from non-spoofed histogram file. Command I used to for generating non-spoofed entropy file is as follows (Figure: 7):

```
root@cnc:~/DDoS_Lab6/Appendix# ./calculate_entropyV2.py outputTime0604.hist shri
yasws
End of process.
('It took ', 679, ' seconds')
```

Figure 7: Generating non-spoofed entropy file

The output is saved in "shriyasws.entr" which is then used to plot the time-series graph which is shown in the figure 8 below.

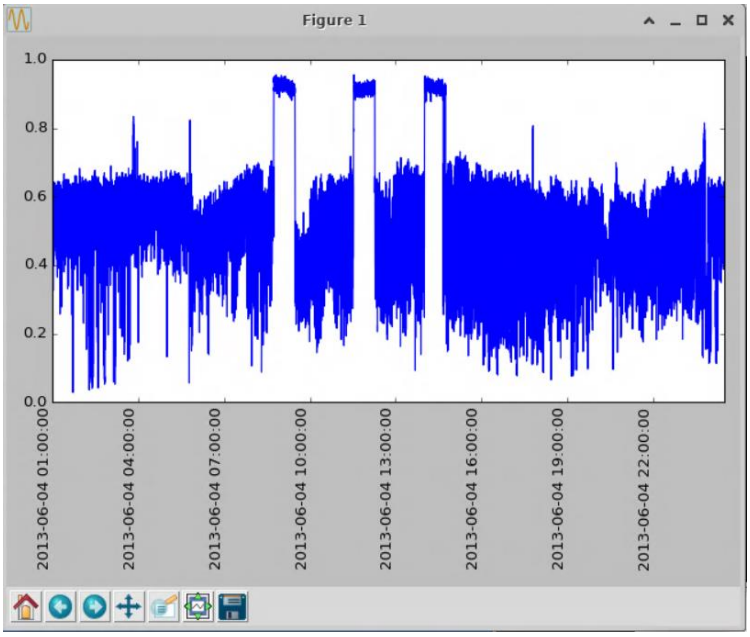


Figure 8: Non-spoofed entropy file

Questions

1. Execute ./entropy_spoofV3.py -h for options and modify the parameters and see if you can get better results.

Following are the options available for the "entropy_spoofV3.py" script.

```
root@cnc:~/DDoS_Lab6/Appendix# ./calculate_entropyV2.py -h
usage: calculate_entropyV2.py [-h] [-c [COLUMNS [COLUMNS ...]]] [-tsi TSI TSI]
                              [-nfed]
                              inputFilename outputFilename

positional arguments:
  inputFilename         The filename of input (histogram) data on which to
                        perform the entropy calculations
  outputFilename        The name of the file in which to write (replace) the
                        calculated entropy output values

optional arguments:
  -h, --help            show this help message and exit
  -c [COLUMNS [COLUMNS ...]], --columns [COLUMNS [COLUMNS ...]]
                        Type of parameter for which to calculate the entropy.
                        Possible naming for columns : 1=src_ip; 2=dst_ip;
                        3=sport; 4=dport; 5=protocol/type; 6=src_dst(tuple);
                        7=sport_dport(tuple); 8=src_dst_sport_dport(tuple)
  -tsi TSI TSI          Time Stamp Interval. This option requires 2 arguments
                        'start time' and 'stop time'. Time entry format is
                        YYYYmmDDHHMMSS
  -nfed                 DO NOT filter empty dictionaries while reading
```

Now we generate the spoofed histogram file using the column value 2 which is destination ip (dst_ip). With a value of 1.1

Here is the command I used for generating it (Figure below):
 # ./entropy_spoofV3.py outputTime0604.hist shriyas2.hist.out complete_attack_times 1.1 -c 2

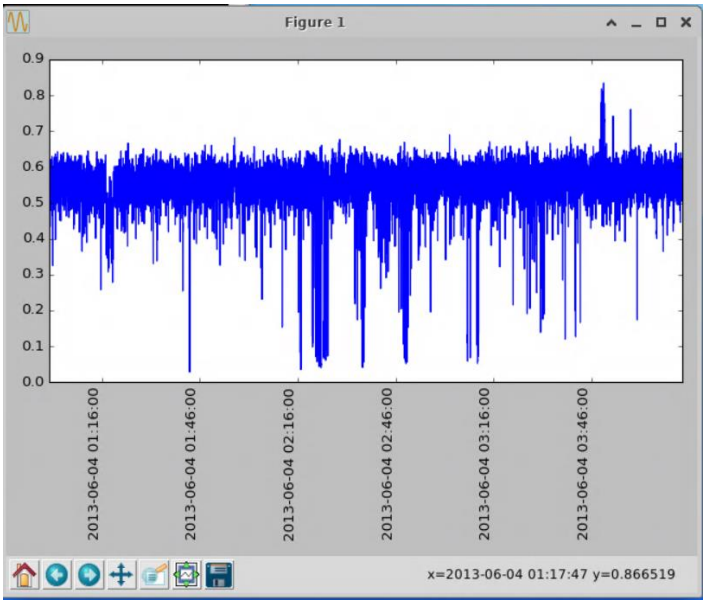
```
root@cnc:~/DDoS_Lab6/Appendix# ./entropy_spoofV3.py outputTime0604.hist shriyas2
.hist.out complete_attack_times 1.1 -c 2
Namespace(FA=None, attack_times='complete_attack_times', columns=['2'], ent=Fals
e, file_name='outputTime0604.hist', margin=[0.15], output='shriyas2.hist.out', s
cale='1.1', spoofPacketCount=False)
Copying outputTime0604.hist to temp.txt
Performing attack number 1
Average cycle: 5.6596313912
Average time: 0.000790303210464
Performing attack number 2
Average cycle: 5.50526662278
Average time: 0.000957189927584
Performing attack number 3
Average cycle: 5.82907600126
Average time: 0.00118995238095
('It took ', 714, ' seconds')
End of process!!
root@cnc:~/DDoS_Lab6/Appendix#
```

This shriyas2.hist is now used to generate spoofed entropy file. Following command is used to perform the same (figure below):

```
root@cnc:~/DDoS_Lab6/Appendix# ./calculate_entropyV2.py shriyas2.hist shriyas2
End of process.
('It took ', 499, ' seconds')

root@cnc:~/DDoS_Lab6/Appendix#
```

Plotting the result will give out the following graph (figure below):



When you compare this with the figure 6. We see that both of them represent spoofed entropy but the values and parameters are tweaked in the graph above. When we spoof using the destination port and with value 1.1 it proves to give out better results since we can see from the graph above there is no abnormal spike or evident pattern that could be used for detecting the attack. Whereas, figure 6 has a slight possibility of detecting the attack using the similar evident pattern between 7:00 – 16:00

2. Discuss the pros and cons of using entropy to detect DDoS attacks.

Pros:

- If entropy method is used for DDoS attack detection, the judgement of an attack is not made too soon right after detecting abnormal network condition but instead network is monitored for a period of time for observing whether the abnormal network condition persistently lasts for a certain period, which proves to be more efficient.
- When we take continuous packets into consideration, the entropy of it can be calculated on them and the entropy value gives out the random distribution of the source IP addresses and when the attack occurs, we can detect the change of the source IP distribution through monitoring the entropy change and thus giving a reason for keeping or discarding the packet. This also proves to be more efficient with respect to traditional detection cause the false positives are comparatively less.

Cons:

- Entropy during an attack can be spoofed which could evade the entropy-based detection algorithm, as we can see from the entropy spoofed time-series graph how the attacks are clouded with the normal traffic to avoid any attention over them.
- An attacker can defeat this detection algorithm by producing flooding attack and simulating the monitors' expected normal data flow and adjust the entropy values during a flooding attack, after knowing some packets attributes' entropy value.

References

[1]: https://link.springer.com/content/pdf/10.1007%2F978-3-540-77048-0_35.pdf