Anomaly Detection in Network Flows

Benford's Law

Bianca I. Colón Rosado
colon.bianca@gmail.com
Computer Science Department
University of Puerto Rico - Río Piedras

Advisor:
Humberto Ortiz Zuazaga
humberto@hpcf.upr.edu
Computer Science Department
University of Puerto Rico - Río Piedras

September 2015

Abstract

A general method for detecting anomalies in network traffic is an important unresolved problem. Using Network Flows it should be possible to observe most anomaly types by inspecting traffic flow. However, to date, there has been little progress on extracting the range of information present in the complete set of traffic flows in a network. Anomaly detection is a method that searches for unusual and out of the ordinary activity in traffic flow packets. In this research, however, we are classifying as flow anomalies the peaks that deviate from the baseline when we apply the Benfords law. We collected flow data using SiLK from the UPR's Science DMZ, a high-performance network for data science. We start using the subspace method to detect anomalies in the following different types of traffic flows: bytes, packets and IP-flow counts.

1 Description

Benford's law also called the First-Digit Law, is a phenomenological law about the frequency distribution of leading digits in many real-life sets of numerical data. That law states that in many naturally occurring collections of numbers the small digits occur disproportionately often as leading significant digits.

We are looking for a general technique for anomaly detection in Network Flows. Our assumption is that Benford's law applies to the TCP flow inter-arrival times as well, and therefore, simpler approaches for computer network traffic analysis can be applied, specifically with the aim of fault and intrusion detection. We expect that intentional attacks alter the first digit distribution of the inter-arrival times can simply be detected without the need of packet header inspection.

The remainder of this report is as follows. In Section 2, we give a general overview on Benford's Law and the anomaly detection concept.

2 Background

2.1 Benford's Law

According to Benford's law of anomalous numbers the frequency of the digit d, appearing as the first significant digit in a collection of numbers, is no uniform as expected intuitively, rather it follows closely the logarithmic relation: Pd = log 10 { 1 + (1/d)}, where d = 1,2,3,4,5,6,7,8,9 and {ZP(d)=1}.

Sets which obey the law the number 1 would appear as the most significant digit about 30% of the time while larger digits would occur in that position less frequently: 9 would appear less than 5% of the time. If all digits were distributed uniformly, they would each occur about 11.1% of the time.

2.2 Anomaly Detection

Anomaly detection is a method that searches for unusual and out of the ordinary activity in traffic flow packets. The general process of working and find anomalies with NetFlow includes capturing, sampling, generating, exporting, collecting, analyzing, visualizing and compare. In this research, however, we are classifying as flow anomalies those packets with an inexplicable amount of bytes.

3 Methodology

We installed a set of flow tools on a computer in the UPR **ScienceDMZ**, this refers to a computer subnetwork that is structured to be secure, but without the performance limits that would otherwise result from passing data through a stateful firewall. The Science DMZ is designed to handle high volume data transfers, typical with scientific and high-performance computing, by creating a special DMZ to accommodate those transfers. Found tools and how we use it in [1].

Using PySilk we make a program (included in the appendix) that extract the start time and the inter-arrival time of each flow. The for flow data analysis we use Plotly.

4 Results

It took one hour to analyze one week of flows. This has billions of coordinates from the inter-arrival time. If we compare Benford's law with the distribution of leading digits in the inter-arrival time of the flows, then we can identify an anomaly as significant discrepancies between them. We can see that there are anomalies but can't identify what type of anomalies, or with what computer. These anomalies are the peaks that deviate from the baseline. To get details of what type of anomalies is, we need to consult other techniques. See Figure 1.

Another way to analyze our flows using the Benford's law is comparing the same day each week. If we take Monday, March 23,2015 and Monday, March 30, 2015 we will expect that the results be almost the same, and if not they could be anomalies. See Figure 2.

5 Conclusion

The Benford's Law was effective with our flows. An important advantage of this method is that malware cannot easily adapt their communication pattern to conform to the logarithmic distribution of first digits. We need to validate the method with labeled or simulated data, and build an alerting system to notify of anomalies as soon as they are detected. Finding a general method for detecting anomalies in flows is hard. But with these techniques we can identify when we have anomalies.

Figure 1: Deviations from the Benford theoretical curve in a week

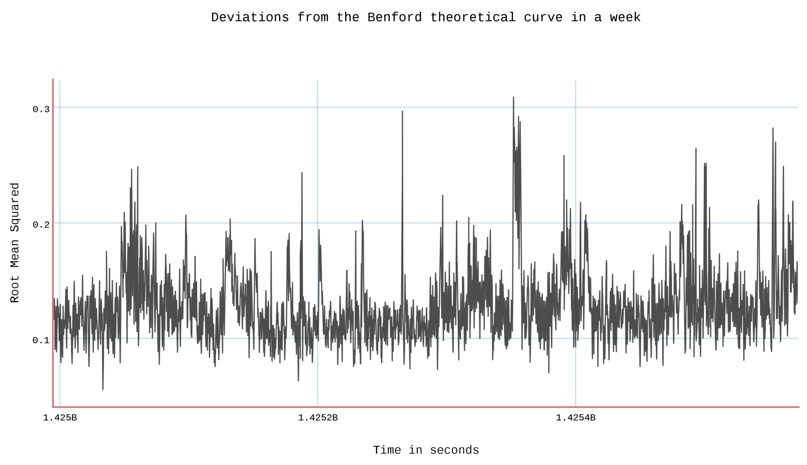
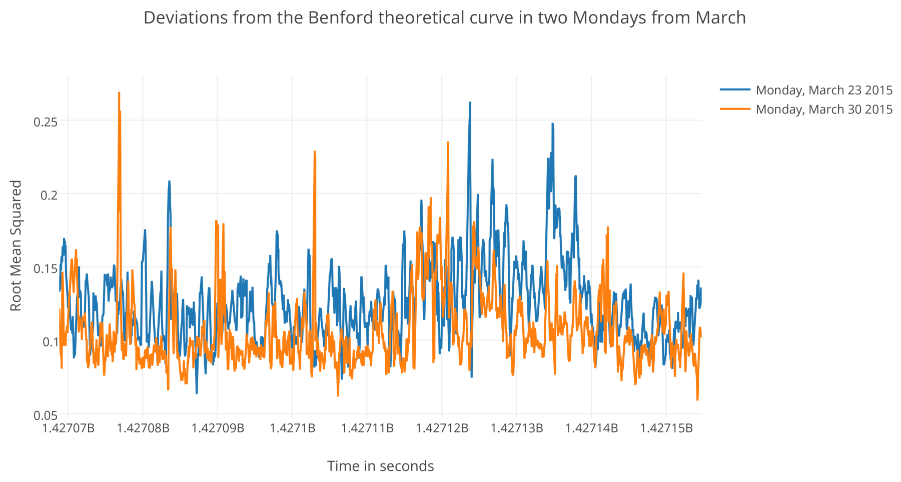


Figure 2: Deviations from the Benford theoretical curve comparing two days



6 Future Work

We want to analyze more the utility of this law, we want to compare the results of the Deviations from the Benford theorical curve with the octets in the same flows.

Also, we will explore new approaches to find new techniques. Implement these techniques for anomaly detection to our collection of flows from UPR's network, and compare results with the results of current techniques. If those techniques are effective we can use it in real time flow collection.

Acknowledgement

This work is supported by the scholarship Academics and Training for the Advancement of Cybersecurity Knowledge in Puerto Rico (ATACK-PR) supported by the National Science Foundation under Grant No. DUE-1438838.

References

[1] Bianca Colón-Rosado, Humberto Ortiz-Zuazaga. Techniques for Anomaly Detection in Network Flows. 2014. `http://figshare.com/articles/Techniques_for_Anomaly_Detection_in_Network_Flows/1424475` `http://ccom.uprrp.edu/~humberto/megaprobe/tag/flows.html`

[2] Iván García, Humberto Ortiz-Zuazaga. Techniques for Anomaly Detection in Network Flows. 2013. `http://ccom.uprrp.edu/~humberto/research/anomaly-detection.pdf`

[3] Bingdong Li, Jeff Springer, George Bebis and Mehmet Hadi Gunes. A survey of network flow applications. *Journal of Network and Computer Applications*, 2013. doi:10.1016/j.jnca.2012.12.020

[4] Lakhina Anukool, Mark Crovella, and Christophe Diot. Characterization of network-wide anomalies in traffic flows. doi=10.1.1.92.7738, 2004.

[5] Laleh Arshadi and Amir Hossein Jahangir. Benford's law behavior of internet traffic. Journal of Network and Computer Applications, 2013. doi:10.1016/j.jnca.2013.09.007.

[6] Ron Bandes, Timothy Shimeall, Matt Heckathorn, Sidney Faber (2014) Using SiLK for Network Traffic Analysis. CERT Coordination Center. 2014. `https://tools.netsa.cert.org/silk/` `http://tools.netsa.cert.org/silk/analysis-handbook.pdf`

[7] FlowBAT. Applied Network Defense, LLC (2014) http://www.flowbat.com/

[8] Plotly. https://plot.ly/

7 Appendix

```
#!/usr/bin/env python

# Use print functions (Compatible with Python 3.0;
  Requires 2.6+)

from __future__ import print_function

# Import the PySiLK bindings
import silk

import math
from collections import defaultdict

def check_flowstarts(flowstarts):
    "check the leading digit distribution for a set of
  flow start times"
    counts = defaultdict(int)
    denom = len(flowstarts)

    for i in range(1, denom):
        last = flowstarts[i-1]
        cur = flowstarts[i]
        if cur == last:
            continue

        digit = int(("%.3e" % (cur - last))[0])
        counts[digit] += 1

    digits = range(0,10)
    freq = [1.0 * counts[d] / denom for d in digits]
```

```python

    return digits[1:10], freq[1:10]

def rms(freq, expected):
    sum = 0
    for i in range(len(freq)):
        sum += (freq[i] - expected[i])**2
    return math.sqrt(sum)

### main
starTimeList= list()
for filename in silk.FGlob(type="all", start_date="2015/02/27", end_date="2015/03/05", site_config_file="/data/conf-v9/silk.conf", data_rootdir="/data/flowsDMZv9/scratch/flow/rwflowpack/"):
        #print(filename)


# https://tools.netsa.cert.org/silk/pysilk.html#RWRec-Object for rec in silk.silkfile_open(filename,silk.READ):
#rec.stime

# The duration of the flow rec in seconds, a float. The default duration_secs value is 0.
#rec.duration_secs

# The start time of the flow rec as a number of seconds since the epoch time, a float. Epoch time is 1970-01-01 00:00:00.
#print(rec.stime_epoch_secs)
                    starTimeList.append(rec.stime_epoch_secs)

starTimeList.sort()

#for i in range(1, len(starTimeList)):
#        #if(i%2 != 0):
```

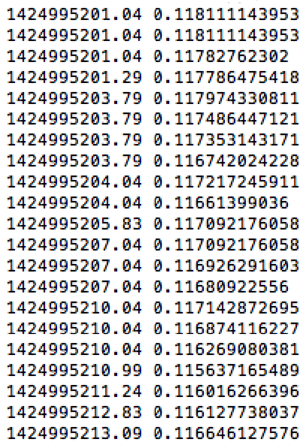
```
#                    print(starTimeList[i] - starTimeList[i-1])

expected = [ math.log(1.0/d+1.0,10) for d in range(1,10)]

xs = []
ys = []

for i in range(len(starTimeList) - 1000):
    digits, freq = check_flowstarts(starTimeList[i:i+1000])
    #xs.append(starTimeList[i])
    print (starTimeList[i], rms(freq,expected))
    #ys.append(rms(freq, expected))
    #if i > 100000:
    #    break
```

Figure 3: The output of leyBenford.py



```
1424995201.04 0.118111143953
1424995201.04 0.118111143953
1424995201.04 0.11782762302
1424995201.29 0.117786475418
1424995203.79 0.117974330811
1424995203.79 0.117486447121
1424995203.79 0.117353143171
1424995203.79 0.116742024228
1424995204.04 0.117217245911
1424995204.04 0.11661399036
1424995205.83 0.117092176058
1424995207.04 0.117092176058
1424995207.04 0.116926291603
1424995207.04 0.11680922556
1424995210.04 0.117142872695
1424995210.04 0.116874116227
1424995210.04 0.116269080381
1424995210.99 0.115637165489
1424995211.24 0.116016266396
1424995212.83 0.116127738037
1424995213.09 0.116646127576
```

```
$ python leyBenford.py > leyBenSilk.txt
$ awk '!(NR % 100)' leyBenSilk.txt > leyGraph.txt
```