Abstract

The Benford’s Law, is a phenomenological law about the frequency distribution of leading digits in many real-life sets of numerical data. 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. To prove it we are using the Numenta Anomaly Benchmark, that contains labeled data with anomalies. We are classifying as flow anomalies the peaks that deviate from the baseline when we apply the Benford’s Law. An important advantage of this method is that malware cannot easily adapt their communication pattern to conform to the logarithmic distribution of first digits. If the Benford’s law works in TCP flows, we will be close to find a general method to detect anomalies in network traffic.
1 Description

Last semester we implemented the Benford’s law [3]. The 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 started looking for a general technique for anomaly detection in Network Flows. Our assumption was 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.

This semester, we implemented the Evaluating Real-time Anomaly Detection Algorithms – the Numenta Anomaly Benchmark (NAB) [9], in order to compare and evaluate different algorithms for detecting anomalies in streaming data. Using NAB we identify two phases where we want to work.

1. Use NAB with our SiLK flows, to identify anomalies in our data.

2. Run our Bendford’s Algorithm with their flows data, that is already labeled with real anomalies. With this approach we can prove if our algorithm work with TCP flow inter-arrival times.

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:

$$P_d = \log_{10} \left(\frac{d + 1}{d}\right), \sum_{d=1}^{9} P(d)$$

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. See Figure 1.

3 Methodology

In order to use NAB with our SiLK flows, to identify anomalies in our data.

1. We start the installation process of NAB in HULK. After numerous attempts of trying, we cannot install it in HULK.

2. Then we start installing NAB locally in my computer. We install it, but only work’s once before it crash. After numerous attempts to make it work we cannot make it work.

We documented the followed steps in GitHub [10].

Then we decide to move to the second phase that is run our Benford’s Algorithm with the NAB flows data, that is already labeled with real anomalies.

1. We adapted our Benford’s program to receive the format provided by NAB. We include the program in the Appendix.

2. We graphed the results in Plotly [8].

4 Results

We start comparing the expected curve provided by Benford’s Law (Figure 1) with the results obtained from the data labeled as no anomalies (Figure 2).

We notice some differences that should not be there assuming the data don’t have anomalies. To find more information about this differences we search the deviations from the Benford theoretical curve.

In the graph of the Deviations from the Benford Theoretical Curve, (Figure 3), there’s a gap between the days from September 5 and some hours of September 9 in the provided data. We didn’t notice that until we make this graph.

Also, looking the graph we need to find information to define the real anomalies. We want to prognosticate how close or far the deviation need to be from zero to be counted as a TP, FP, FN, TN.
5 Conclusion

In order to prove that the Benford’s Law can detect anomalies in Network Flows we need to continue validating the algorithm with labeled data. NAB provide a lot of files with data, but we don’t get enough time to inspect all the files. Also we need to define when we classify the results of the Deviations
An important advantage of this method is that malware cannot easily adapt their communication pattern to conform to the logarithmic distribution of first digits.

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 theoretical 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 and build an alerting system to notify the anomalies as soon as they are detected.

Acknowledgement

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


7 Appendix

#! /usr/bin/env python

# Research Investigation : Spring 2015–2016
# Techniques for Anomaly Detection using Benford’s Law
#
# Using the data provided by NAB (https://github.com/numenta/NAB)
# we will analyze the results of the Benford’s Law.
#
import math
from sys import argv

def significantNumber(num): # receive 0.0024
    number = num.split(' ')[1] # '0' '0024'
aDot = number[1] # '0024'
    for d in range(0, len(aDot)):
        if aDot[d] != '0':
            return aDot[d]
        else:
            pass

def howManyNumbers(listCounter):
    allNum = [] # List to put all the numbers
    diNum = [] * 10 # List to put counts of numbers
    for i in range(0, len(listCounter)):
        allNum.append(int(listCounter[i])) # Change strings for numbers
    totalNum = len(allNum) # Total of numbers
    for c in range(1,10):
        diNum.insert(c, 1.0 * allNum.count(c)/totalNum)
    return diNum
```python
def benford(listCounter):
    allNum = [] # List to put all the numbers
    diNum = [] * 10 # List to put counts of numbers
    for i in range(0, len(listCounter)):
        allNum.append(int(listCounter[i])) # Change strings for numbers
    for c in range(1,10):
        diNum.insert(c, allNum.count(c))
    return diNum

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

# main
if (len(argv) == 4):
    filename = argv[1]
    columnName = int(argv[2]) - 1
    windowSize = int(argv[3])

    with open(filename) as f:
        content = f.read().splitlines()
    f.close()

    value = [] # For the values
    fDigit = [] # For the first significant digit

    # Take the only values of the line
    # 1. Starts in line 1 because line 0 contain column title. (range(1 ... )
    ## timestamp,value,anomaly_score,raw_score,label,S(t)_reward_low_FP_rate,S(t)_reward_low_FN_rate,S(t)_standard
```
# 2. The value is positioned in the second column. (The split (’,’)[1]) columnNumber should be 1.

2015-09-01 13:45:00, 3.06, 0.0301029996659, 1.0, 0.0, 0.0, 0.0, 0.0

for i in range(1, len(content)):  # 1.
    if float(content[i].split(’,’)[columnNumber]) != int(0):
        value.append(content[i].split(’,’)[columnNumber])
    else:
        pass

# To that values, just select the first significant digit
for j in range(0, len(value)):
    if value[j].isdigit() == True:  # For int values
        if value[j][0].isdigit() == True:
            fDigit.append(value[j][0])
        else:
            print "Alert Anomaly: value = %d" % (value[j])
    else:  # For float values
        if value[j].split(’.’)[0] != ’0’:  # Save the left dot number for a number like 12.003
            fDigit.append(value[j].split(’.’)[0][0])  # From 12.003 only save the 1
        else:  # Save the right dot number for a number like 0.0034
            fDigit.append(significantNumber(value[j]))

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

### ONLY BENFORD GRAPH
#bVal = benford(fDigit)
#for n in range(0, len(bVal)):
#    print n+1, bVal[n], expected[n]
###
for i in range(len(fDigit) - windowSize):
    valuesFD = howManyNumbers(fDigit[i:i+windowSize])
    print(content[i].split(',')[0], rms(valuesFD, expected)
    # The timestamp is in the first column, that's why
c CONTENT[i].split(',')[0]

else:
    print"
    print "Need to provide the name of the file that have the data"
    print "and in what column is the value data"
    print "Example:"
    print "timestamp, value, anomaly_score, raw_score, label, S(t)_reward, low_FP_rate"
    print "The value column is the second"
    print ""
    print "Format to write the missing information"
    print "$ python benford-NAB.py fileName columnName windowSize"
    print "$ python benford-NAB.py numenta_occupancy_6005.csv 2 100"
    print ""