Techniques for Anomaly Detection in Network Flows

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

April 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 flows. 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 those packets with an inexplicable amount of bytes. We collected flow data using SiLK from the UPR's Science DMZ, a high-performance network for data science. We analized the flows with FlowBAT. No real anomaly was detected because we need to collect more flow data to establish patterns and find anomalies.

1 Description

This work seeks to detect anomalies in network traffic by inspecting the network flow data. For our purposes, a **computer network** is a telecommunication network that allows computers to exchange data, which is transferred in the form of packets. An **anomaly** is something that deviates from what is standard, normal, or expected. A **network flow** is a summary of a sequence of packets sent from a source computer to a destination, which may be another host, a multicast group, or a broadcast domain. A flow could consist of source IP, destination IP, source port, destination port, packets, bytes, flags, source time, etc.

Anomaly detection is the identification of flows which do not conform to an expected pattern or other flows in a dataset. The general process of working and find anomalies with NetFlow includes capturing, sampling, generating, exporting, collecting, analyzing, visualizing and compare [2].

2 Methodology

In all the papers that we explored, we didn't find a general method to detect anomalies. But it is necessary to look back and learn what perspectives have been achieved, and what methods have been used and are more effective in order to move forward.

We have a collection of v5 flows from the HPCF for a single day in September 2011 which were used in [1]. We can compare any new techniques we develop with the results from that work.

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.

One of our flow tools is **SiLK** [5]. We configure SiLK in this subnetwork because it supports the efficient collection, storage, and analysis of network flow data, enabling network security analysts to rapidly query large historical traffic data sets. With SiLK we collected IPv4 and IPv6 flows. The other tool is **FlowBAT** [6] is a graphical flow-based analysis tool. We use this tool to detect the anomalies efficiently.

2.1 SiLK

System for Internet-Level Knowledge (SiLK), is a collection of traffic analysis tools developed by the CERT Network Situational Awareness Team (CERT NetSA) to facilitate security analysis of large networks.

To see our flows we have to export the data, for that we has to say SiLK where the data is. Then we can view this data with rwfilter.

```
$ export SILK_DATA_ROOTDIR=/data/dmz-flows/silk-v9/flow

$ rwfilter --start-date=2015/02/26:13 --end-date
    =2015/02/26:16 --sensor=S0,S1 --type=all --proto=all
     --print-volume --threads=4 --passpdestination=
    stdout --site-config-file=/data/conf-v9/silk.conf |
    rwcut
```

2.2 First 10 IP-numbers

In the paper that we explored [1], their methods successfully detected anomalies. We try to use that methods to detect anomalies in our flows from ScienceDMZ.

The C++ program in [1] that identifies first 10 IP-numbers that generated the most traffic in a network. The program start by reading a file which contains IP numbers with its respective octets. Then, it sums the octets for repeated IP-numbers. Finally, it orders the list of IP numbers, placing those with more octets at the final and we obtain the first 10 IP numbers with the most octets.

```
// Programmer: Ivan O. Garcia
// Edited: Ricardo Lopez Torres
// Date: August 28, 2013
// Purpose: To read a file called "test.out" to extract
// all sources IP with its octets to see what IP
// address have the most traffic.
// Program: ip-octet-sort.cpp


#include <iostream>
#include <iomanip>
```

```
#include <fstream>
#include <string>
#include <map>
#include <vector>
#include <algorithm>

using namespace std;

        // object holding IP Octet pairs
struct ipOctetPair
{
    string IP;
    int octets;

        // overload of < operator
        // allows sorting ipOctetPairs by octet count
    bool operator<(const ipOctetPair &iopair) const {
        return octets < iopair.octets;
    }
};

int main() {

// Reading 'test.out' file.
  // Creating an input file variable.
   ifstream in ;     // Variable to access the file.


    in.open("test.out"); //Opening file.

    // To hold srcIP and dstIP.
    string srcIP, dstIP ;

    // To hold respective data.
  int prot, srcPort, dstPort,
      octets, packets ;

  // Skip the header line.
```

```
  string line ;
  getline(in, line) ;

    //Uses 'map' function to match 'srcIP' with 'oc
    map<string,int> matches ;


// matches["0.0.0.0"]= 0 ;
// To have something in the 'map'.  tets'.

  // To iterate each file's line.
  while(!in.eof()) {

    // Primming read.
    in >> srcIP >> dstIP >> prot >> srcPort
       >> dstPort >> octets >> packets ;

          //To iterate the 'map'.
         map<string,int>::iterator it ;

// if 'srcIP' is in the list: it = address of 'srcIP'
         it = matches.find(srcIP) ;

    //new 'srcIP'='srcIP' in 'map': do the following.
      if(it!=matches.end() && srcIP == (it->first)){

              //To hold old 'octets' count.
              int oct_temp ;
              // new 'octets' + old 'octets'.
              oct_temp = (it->second) + octets;

         // Erase old 'key' and its value from 'map'.
              matches.erase(it) ;

              //Places new 'key' and its value.
              matches[srcIP] = oct_temp ;
              }

```

```
        else
        //Places new 'key' and its value in the map.
            matches[srcIP] = octets;
  } // End of 'while' cycle.

// vector to store ipOctetPair objects
vector<ipOctetPair> iopairs;

// place all elements of matches into iopairs vector
for (map<string,int>::iterator it = matches.begin();
                    it != matches.end(); ++it) {

    ipOctetPair iopair;
    iopair.IP = it->first;
    iopair.octets = it->second;

    iopairs.push_back(iopair);
}

// sort by octet - see operator<() in ipOctetPair
sort(iopairs.begin(), iopairs.end());


// iterate over sorted elements in iopairs vector
for (vector<ipOctetPair>::iterator it = iopairs.begin()
   ;
                    it != iopairs.end(); ++it) {

    std::cout << setw(18) << it->IP << setw(12)
                   << it->octets << std::endl ;
}

// Closing file
  in.close() ;

return 0 ;

} // End of 'main' function.
```

To save the data that you want to analize in ".out" file to use the **First 10 IP-numbers** program you need to run this command. For version four flows *−−ip−version=4* and for version six flows *−−ip−version=6*.

```
$ export SILK_DATA_ROOTDIR=/data/dmz-flows/silk-v9/flow
$ rwfilter --start-date=2015/04/10 --ip-version=4 --
   print-statistics --pass=stdout --site-config=/data/
   flowsDMZv9/scratch/flow/rwflowpack/silk.conf --type=
   all --data-rootdir=/data/flowsDMZv9/scratch/flow/
   rwflowpack/ | rwcut --fields=1,2,5,3,4,7,6 >test.out

Files 66. Read 258496. Pass 248639. Fail 9857.

$ less test.out
```

Figure 1: The output of less test.out should be

```
            sIP|                                     dIP|pro|sPort|dPort|     bytes|   packets|
 136.145.231.17|                         221.235.189.249|  6|   22|49152|      3145|        20|
 136.145.231.56|                           222.161.4.148|  6|   22|24525|      4527|        28|
 136.145.231.39|                           222.161.4.148|  6|   22|27457|      4355|        25|
 136.145.231.15|                           222.161.4.148|  6|   22|59607|        52|         1|
 136.145.231.16|                           222.161.4.148|  6|   22|33731|      3093|        19|
 136.145.231.11|                           222.161.4.148|  6|   22|10702|      3265|        22|
 136.145.231.17|                           222.161.4.148|  6|   22|27625|      3277|        22|
 136.145.231.18|                         221.180.149.120|  6|   22|42678|      2589|        15|
 136.145.231.13|                         221.235.189.249|  6|   22|39675|      3093|        19|
 136.145.231.17|                         221.235.189.249|  6|   22|59143|      3093|        19|
 136.145.231.11|                           222.161.4.148|  6|   22|10702|       104|         2|
 136.145.231.11|                           130.156.34.15|  6|56534| 9995|      1954|         7|
 136.145.231.13|                         221.235.189.249|  6|   22|49218|      3093|        19|
 136.145.231.18|                           222.161.4.148|  6|   22|36091|      3221|        21|
 136.145.231.18|                         221.180.149.120|  6|   22|47233|      2589|        15|
 136.145.231.11|                           222.161.4.148|  6|   22|10702|        52|         1|
 136.145.231.21|                           91.212.124.14|  6| 5900|51741|       403|         8|
 136.145.231.22|                           91.212.124.14|  6| 5900|51742|       358|         7|
 136.145.231.16|                           222.161.4.148|  6|   22|40832|      3153|        20|
 136.145.231.17|                         221.235.189.249|  6|   22| 4821|      3093|        19|
 136.145.231.13|                         221.235.189.249|  6|   22|58838|      3093|        19|
 136.145.231.51|                           222.161.4.148|  6|   22|25820|      4775|        31|
 136.145.231.17|                           222.161.4.148|  6|   22|39999|      2325|        11|
 136.145.231.17|                         221.235.189.249|  6|   22|13940|      3093|        19|
 136.145.231.13|                         221.235.189.249|  6|   22| 5456|      3093|        19|
 136.145.231.18|                         221.180.149.120|  6|   22|51517|      2589|        15|
```

But our program don't like the pipes "|" after every columns. We need to eliminate that pipes.

```
$ vi test.out
:%s/|/ /
```

After we remove the pipes we can compile the program.

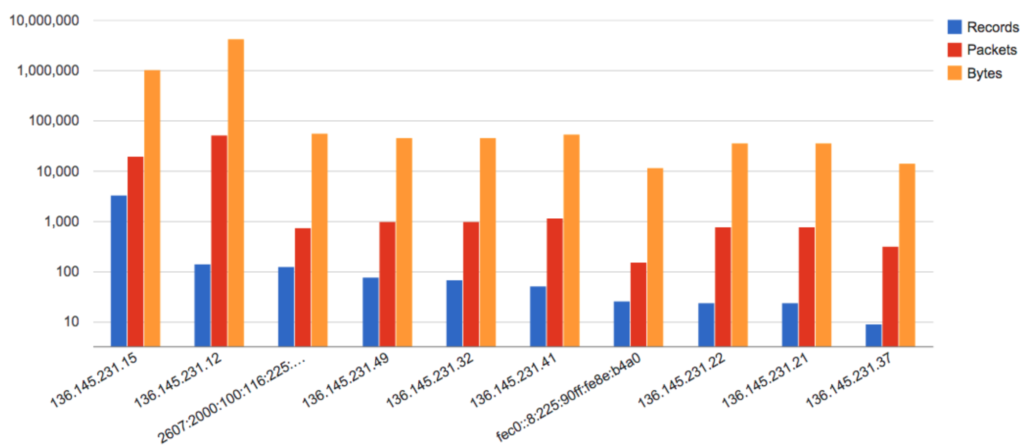
```
$ g++ ip-octet-sort.cpp -o ip-octet-sort
$ ./ip-octet-sort
```

Figure 2: Top 10 IP′s in IPv4 & IPv6

Source IP	Records
136.145.231.11	1162027
136.145.231.33	1598613
136.145.231.36	1763883
136.145.231.19	2026952
136.145.231.15	2029427
136.145.231.13	2812466
136.145.231.22	3841398
136.145.231.35	9160381
136.145.231.41	18587215
136.145.231.37	69744320

Source IP	Records
2607:2000:100:116:225:90ff:fe8e:b61c	638272
2607:2000:100:116:225:90ff:fe8e:b4a0	789106
2607:2000:100:116:92e2:baff:fe5a:7ded	817865
2607:2000:100:116:e9f5:a850:8fcf:ba58	1274992
2607:2000:100:116:c24a:ff:fe09:49a8	2497012
2607:2000:100:116:92e2:baff:fe5a:7685	2548186
2607:2000:100:116:5074:8c32:5dd2:d949	2793566
2607:2000:100:116:25a7:dbb7:884:f0d0	4586989
2607:2000:100:116:bc40:e5f0:a5b4:4903	20291696
2607:2000:100:116:cdcd:7853:2837:de0e	39725518

Figure 3: Top 10 in FlowBAT of IPv4 & IPv6



3 Future Work

In a future, we will explore the second technique used in [1], the Benford's Law [4], and futuremore new approaches to find new techniques. Implement this techniques for anomaly detection to our collection of flows from UPR's network, and compare results with the results of current techniques.

4 Conclusion

In conclusion we can saw ipv4 and ipv6 flows using SiLK. Find a general method for detecting anomalies in flows is hard, but we will continue working with our flows to find new techniques. Looking into flows may be the only way to monitor security in future networks.

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

[2] 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

[3] Lakhina Anukool, Mark Crovella, and Christophe Diot. Characterization of network-wide anomalies in traffic flows. citeseerx.ist.psu.edu/viewdoc/download doi=10.1.1.92.7738&rep=rep1&type=pdf, 2004.

[4] 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.

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

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