Techniques for Anomaly Detection in IPv4 and IPv6 Network Flows

Grace M. Rodríguez Gómez

grace.rodriguez6@upr.edu

University of Puerto Rico – Rio Piedras Campus

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

Abstract:

In a growing demand for web applications, it's imperative to make sure services in the net are secure. One method that researchers are exploring to improve web security is anomaly detection in traffic flows. In this report, we examine how efficient are SiLK tools to detect flow anomalies and analyze IPv4 and IPv6 flow data. Then, we make and implementation that converts the IPv6 addresses to coordinates to make a 3-dimensional graph. With the help of these methods we were able to have graphical formats that help us see the amount of IPv6 addresses, both from the University Of Puerto Rico's network and outside, and their connectivity.

1. Introduction:

A flow is defined as a set of IP packets passing an observation point in the network during a certain time interval [9]. These packets form a sequence of the flows source address to the destinations IP address. Flow information is analyzed because it can help discover external and internal activities such as network misconfiguration and policy violations and it helps protect the users privacy. One way to analyze flow data is with anomaly detection. 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 a flow anomaly those packets with an inexplicable amount of data (bytes).

The tools that we used to analyze flow data collected from the University of Puerto Rico – Rio Piedras Campus (UPR-RP)'s network is System for Internet-Level Knowledge (SiLK). The first flow data that was analyzed with SiLK tools was IPv4 flows, but afterwards IPv6 flow data was also analyzed. IPv6 (Internet Protocol version 6) is the latest revision of the Internet protocol and was created to deal with the long-anticipated problem of IPv4 address exhaustion [6].

2. Past work:

There were two papers, Ivan Garcia's [4] and Bianca's[2], that were explored to gather information about anomaly detection in network flow. In the first paper [4], he

analyses how the subspace method is used to detect anomalies in flow data. He also makes an implementation, which separates and identifies IP numbers that are found to generate the most traffic in a network. Bianca's paper investigates the different approaches that exist for anomaly detection, such as counting flow/packets/octets, comparing to averages and machine learning.

3. Methods:

3.1 Analyzing flow data with *flow-tools*

In order to get familiarized with analyzing flow data, the data was first anlayzed using **flow-tools. Flow-tools** is a library and a collection of programs used to collect, send, process, and generate reports from NetFlow data [3]. The flow data obtained was from the UPR Network and using the command

flow-cat /data/dmz-flows/Flows-v5/2015/2015-01/2015-01-01/* | flow-stat -f9

we were able to find the top ten users with the most flows in the date of 2015-01-01.
The **flow-cat** utility processes files and/or directories of files in the flow-tools format. [3] The next line is the path where the file that we want to process is. The output of **flow-cat** is going to be passed to flow-stat. The **flow-stat** utility generates usage reports for flow data sets by IP address, IP address pairs, ports, packets, bytes, interfaces, next hops, autonomous systems, ToS bits, exporters, and tags. The last specification, **–f ,** means we're going to instruct to generate a usage report based on source IP address (-f9) and to report the totals in percent/total form.

3.2 Starting to use *SiLK tools* to analyze flow information.

SiLK tools are a collection of traffic analysis tools developed by the CERT Network Situational Awareness Team (CERT NetSA) to facilitate security analysis of large networks [10]. We started to use SiLK-tools instead of *flow-tools* because SiLK tools supports IPv6 flow and flow-tools doesn't. We want to extend this research to analyze IPv4 and IPv6 flow data.

At first, learning how to use SiLK tools turned out to be challenging because it was the first time we were using it for this research. After getting more familiarized with SiLK's Analysis Suite and reading the handbook *Using SiLK for Network Analysis* [1], we manage to learn some useful commands that help us analyze flow data in a more organize way.

The data that was tested first to learn how to use the commands in SiLK was the data that is provided as reference data for use the tool suite. This sample data is derived from anonymized enterprise packet header traces obtained from Lawrence Berkeley National Laboratory and ICS. It can be found in this link: https://tools.netsa.cert.org/silk/referencedata.html. The first command that was used to learn how to organize data was **rwcount. Rwcount** summarize (aka group or bin) SiLK Flow records across *time*, producing textual output with counts of bytes, packets, and flow records for each time bin. [10]. The first command with **rwcount** was quite simple:

rwcount LBNL_nonscan/SiLK-LBNL-05/in/2004/10/04/in-S0_2004004.20.

It helps see a review of all the data in that file. Doing **rwcut** for the same file, we can see the attributes of SiLK Flow records in a delimited, columnar, human-readable format. The command is

rwcut LBNL_nonscan/SiLK-LBNL-05/in/2004/10/04/in-S0_2004004.20

```
[gracemarod@hulk ~]$ rwcut LBNL_nonscan/SiLK-LBNL-05/in/2004/10/04/in-S0_20041004.20
                 sIP|                 dIP|sPort|dPort|pro|   packets|     bytes|   flags|               sTime| duration|                eT
ime|sen|
    203.13.173.243|     128.3.45.10|   53| 2124| 17|         1|        64|        |2004/10/04T20:05:02.331|    0.000|2004/10/04T20:05:02.
331|  ?|
     33.115.84.19|  128.3.46.146| 4087| 5554|  6|         1|        48|  S     |2004/10/04T20:11:48.675|    0.002|2004/10/04T20:11:48.
677|  ?|
     33.115.84.19|  128.3.46.146| 4434| 9898|  6|         1|        48|  S     |2004/10/04T20:11:49.497|    0.001|2004/10/04T20:11:49.
498|  ?|
    148.184.175.97|     128.3.44.112|  389| 1549| 17|         1|       201|        |2004/10/04T20:07:18.685|    0.000|2004/10/04T20:07:18.
685|  ?|
    148.184.175.97|     128.3.44.112|    0|    0|  1|         2|       120|        |2004/10/04T20:07:39.974|    0.182|2004/10/04T20:07:40.
156|  ?|
    207.235.115.253|     128.3.44.112| 5190| 4973|  6|         2|        80|    A   |2004/10/04T20:03:57.090|  300.267|2004/10/04T20:08:57.
357|  ?|
    207.235.255.108|     128.3.44.112| 5002| 4974|  6|        10|       767|   PA   |2004/10/04T20:04:00.576|  303.024|2004/10/04T20:09:03.
600|  ?|
    148.184.191.214|     128.3.44.112|    0|  771|  1|         2|       112|        |2004/10/04T20:07:18.788|  115.138|2004/10/04T20:09:13.
926|  ?|
     148.184.171.6|     128.3.44.112|  389| 1560| 17|         1|       197|        |2004/10/04T20:09:14.022|    0.000|2004/10/04T20:09:14.
```

figure 3.1 Display of flow data with rwcut

For the next command, we save some of the sample data in the folder flowTest2.rw. The command **rwsort** sorts SiLK Flow records using a user-specified key comprised of record attributes, and write the records to the named output path or to the standard output [10]. The command

rwsort --xargs --fields=sTime --output-path=flowTest2.rw <<END-OF-LIST

saves the specified set of data into the folder flowTest2.rw. The data that we saved was in-S0_20041004.20, in-S0_20041004.21 and in-S0_20041004.22. The **rwsort** program has an option called --xargs telling it to get a list of input files from stdin. The here-document supplies data to stdin and is specified with double less than symbols (<<), followed by a string that defines the marker that will indicate the end of the here-document data [1].

The folder flowTest.rw also facilitate us to find out useful information about the flow data using **rwfileinfo.** This utility prints information (type, version, etc.) about a SiLK Flow, IPset, Bag, or Prefix Map file [1] The command **rwfileinfo --field=format flowTest2.rw** tells us that the format (id) of those flows in **flowTest2.rw** is FT_RWIPV6ROUTING(0x0c). The other command **rwfileinfo –field=byte-order flowTest2.rw** displays that the byte-order of those flows is Little Endian, which means that the least significant bytes are saved in the smallest address. There are more commands you can write in **--field** to find out more information about the flows we want to analyze. This site gives all of the available fields for **rwfileinfo** https://tools.netsa.cert.org/silk/rwfileinfo.html.

Another command that's similar to **rwfileinfo** but prints information about the sensors, classes, and types specified in the silk.conf file [https://tools.netsa.cert.org/silk/docs.html] is the utility **rwsitinfo**. The SiLK tool **rwsiteinfo** can produce a list of sensors in use for a specific installation, reflecting its configuration. The **rwsiteinfo** command can display all information from the site configuration [1]. With the command **rwsiteinfo --fields=id-sensor,sensor --site-config-file=/home/gracemarod/flowsSilk/SiLK-LBNL-05/silk.conf**, we were able to see the sensors and the sensors id.

figure 3.2 Display of the sensor and sensors-ID in the silk.conf

We put in **--fields** the information we want to display like sensor-id and sensor. We have to specify the path of the silk.conf with **–site-config-file** so it can now where to the silk.conf.

3.3 Using rwfilter to display and analyze data

One of the most common and useful commands is **rwfilter. Rwfilter** selects SiLK Flow records form the data repository and partition the records into one or more 'pass' and/or 'fail' output streams [10]. It can also be used with other commands. Before using the command **rwfilter,** we have set the **SILK_DATA_ROOTDIR** variable to use the SiLK command line tools. In our case, we used this command: **export SILK_DATA_ROOTDIR=/home/gracemarod/LBNL_nonscan/SiLK-LBNL-05.**

Then, we use the command that will give us an output in the “pass” and “fail” streams.

rwfilter --start-date=2004/10/04:20 --end-date=2005/01/08:05 --sensor=S0,S1 --type=all --proto=1,6,17 --print-volume --threads=4 --pass-destination=stdout --site-config-file=silk.conf | rwuniq --fields=proto --sort-output --values=records, bytes, packets

figure 3.3 Data output of rwfilter with specifications

To get the output above, we need to specify the date of collection with **--start-date** and the --**end-date**. The **--sensor** switch is used to select data from specific sensors. The **--type** predicate further specifies data within the selected *CLASS* by listing the *TYPE*s of traffic to process. The switch takes a comma-separated list of types or the keyword all which specifies all types for the specified *CLASS*. The **--proto** switch partitions the flow records into a *pass* stream and a *fail* stream. **--print-volume** prints a four line summary of rwfilter's processing. **--threads** invokes rwfilter with *N* threads reading the input files. The **--pass-destination** switch tells **rwfilter** to write the records that pass the **--proto** test to the file *stdout*. With **--site-config-file, rwfilter** reads the SiLK site configuration from the named file *silk.confif*. Its important to specify the file path of *silk.config* if this file is not in the folder where the command **rwfilter** is being done.

For its part, **rwuniq** summarizes SiLK Flow records by a user-specified key comprised of record attributes and print columns for the total byte, packet, and/or flow counts for each bin [10].

Testing different commands with **rwfilter,** we added the command **rwcut**, which prints the attributes of SiLK Flow records in a delimited, columnar, human-readable format.

rwfilter --start-date=2004/10/04:20 --end-date=2005/01/08:05 --sensor=S0,S1 --type=all --proto=1,6,17 --print-volume --threads=4 --pass-destination=stdout --site-config-file=silk.conf | rwcut

```
rwcut: unrecognized option '--all-destination=all.txt'
Use 'rwcut --help' for usage
[gracemarod@hulk SiLK-LBNL-05]$ rwfilter --start-date=2004/10/04:20 --end-date=2005/01/08:05 --sensor=S0,S1 --type=all --proto=1,6,17 --print-volume --threads=4 --pass-destination=s
tdout --site-config-file=silk.conf -all-destination=all.txt | rwcut
                sIP|                dIP|sPort|dPort|pro|   packets|     bytes|   flags|               sTime| duration|                eT
ime|sen|
     203.13.173.243|      128.3.45.10|   53| 2124| 17|         1|        64|        |2004/10/04T20:05:02.331|    0.000|2004/10/04T20:05:02.
331| S0|
      33.115.84.19|     128.3.46.146| 4087| 5554|  6|         1|        48| S      |2004/10/04T20:11:48.675|    0.002|2004/10/04T20:11:48.
677| S0|
      33.115.84.19|     128.3.46.146| 4434| 9898|  6|         1|        48| S      |2004/10/04T20:11:49.497|    0.001|2004/10/04T20:11:49.
498| S0|
     148.184.175.97|     128.3.44.112|  389| 1549| 17|         1|       201|        |2004/10/04T20:07:18.685|    0.000|2004/10/04T20:07:18.
685| S0|
     148.184.175.97|     128.3.44.112|    0|    0|  1|         2|       120|        |2004/10/04T20:07:39.974|    0.182|2004/10/04T20:07:40.
156| S0|
     207.235.115.253|     128.3.44.112| 5190| 4973|  6|         2|        80|    A   |2004/10/04T20:03:57.090|  300.267|2004/10/04T20:08:57.
357| S0|
     207.235.255.108|     128.3.44.112| 5002| 4974|  6|        10|       767|   PA   |2004/10/04T20:04:00.576|  303.024|2004/10/04T20:09:03.
600| S0|
     148.184.191.214|     128.3.44.112|    0|  771|  1|         2|       112|        |2004/10/04T20:07:18.788|  115.138|2004/10/04T20:09:13.
926| S0|
```

figure 3.4 Adding rwcut to rwfilter so we can read the data in a more organized and understandable way

3.4 Using rwfilter to display and analyze data from the UPR-RP Network

After becoming more familiar in using **rwfilter** to display the desired data, we decided to use flow records from the UPR-RP's Network. We tested with flows captured in the date of 26/02/26. It is important to remember to set the **SILK_DATA_ROOTDIR** variable or **rwfilter** won't work. At the end, we used the command **rwfilter --start-date=2015/02/26T17:07:00 --end-date=2015/02/26T20:24:00 --sensor=S0,S1 --type=all --proto=1,6,17 --print-volume --threads=4 --pass-destination=stdout --site-config-file=/data/conf-v9/silk.conf | rwcut** to see the flow data from the UPR's network.

```
[gracemarod@hulk ~]$ rwfilter --start-date=2015/02/26T17:07:00 --end-date=2015/02/26T20:24:00 --sensor=S0,S1 --type=all --proto=1,6,17 --print-volume --threads=4 --pass-destination=
stdout --site-config-file=/data/conf-v9/silk.conf  | rwcut
rwfilter: Warning: start-date precision greater than hours ignored
rwfilter: Warning: end-date precision greater than hours ignored
                sIP|                dIP|sPort|dPort|pro|   packets|     bytes|   flags|               sTime| duration|                eT
ime|sen|
     136.145.231.37|      224.0.0.1|62210| 8612| 17|         1|        46|        |2015/02/26T17:11:41.426|    0.150|2015/02/26T17:11:41.
576|  0|
     136.145.231.41|    98.139.221.57| 4085|   25|  6|         1|        60|        |2015/02/26T17:11:41.526|    0.050|2015/02/26T17:11:41.
576|  0|
     136.145.231.16|  61.160.224.130|32764|43554|  6|         1|        46|        |2015/02/26T17:16:42.990|    0.100|2015/02/26T17:16:43.
090|  0|
     136.145.231.41|    98.139.221.57| 4086|   25|  6|         1|        60|        |2015/02/26T17:11:42.976|    0.150|2015/02/26T17:11:43.
126|  0|
     136.145.231.37|      224.0.0.1|65095| 8612| 17|         1|        46|        |2015/02/26T17:11:48.526|    0.250|2015/02/26T17:11:48.
776|  0|
     136.145.231.41|   98.136.217.202| 3854|   25|  6|         9|       522|        |2015/02/26T17:16:42.290|    1.300|2015/02/26T17:16:43.
590|  0|
     136.145.231.18|     69.28.67.44|  123|  123| 17|         1|        76|        |2015/02/26T17:11:52.426|    0.200|2015/02/26T17:11:52.
626|  0|
     136.145.231.37|      224.0.0.1|55484| 8612| 17|         1|        46|        |2015/02/26T17:11:55.572|    0.100|2015/02/26T17:11:55.
672|  0|
     136.145.231.13|    146.66.19.107|    0|  778|  1|         1|        88|        |2015/02/26T17:11:57.472|    0.250|2015/02/26T17:11:57.
```

figure 3.5 Using **rwfilter** with **rwcut** to display flow data from the UPR's network

3.5 Analyzing IPv6 flows from the UPR's network with rwfilter and rwcut

Since we want to extend this research to IPv6 flows, we used the command **rwfilter** with **rwcut** to display IPv6 flow data. At first, it was challenging making **rwfilter** work with IPv6 mainly because we couldn't find where the IPv6 data was stored. At the end, with the command **rwfilter --start-date=2015/04/10 --ip-version=6 --print-statistics --pass=stdout --site-config=/data/flowsDMZv9/scratch/flow/rwflowpack/silk.conf --type=all --data-rootdir=/data/flowsDMZv9/scratch/flow/rwflowpack/ | rwcut** we got the result in figure 3.6. The command is mostly the same as the one we used to display data for IPv4 in section

3.5. In this command, we added the **--ip-version=6,** that pass the record if its IP Version is in the specified list. We also added the command **--data-rootdir** that tells **rwfilter** to use */data/flowsDMZv9/scratch/flow/rwflowpack/* as the root of the data repository.

```
[gracemarod@hulk data]$ rwfilter --start-date=2015/04/10 --ip-version=6 --print-statistics --pass=stdout --site-config=/data/flowsDMZv9/scratch/flow/rwflowpack/silk.conf --type=all
 --data-rootdir=/data/flowsDMZv9/scratch/flow/rwflowpack/ | rwcut
                                 sIP|                                 dIP|sPort|dPort|pro|   packets|     bytes|   flags|                 sTime| duration|                 eT
ime|sen|
   2607:2000:100:116:225:90ff:fe8e:b4a0|  2607:2000:100:116:92e2:baff:fe5a:7ded|57401|   22|  6|         6|       450|        |2015/04/10T13:44:31.326|    0.100|2015/04/10T13:44:31.
426|  0|
   2607:2000:100:116:225:90ff:fe8e:b4a0|  2607:2000:100:116:92e2:baff:fe5a:7ded|55993|  443|  6|         9|      1130|        |2015/04/10T13:44:45.120|    0.200|2015/04/10T13:44:45.
320|  0|
  2607:2000:100:116:92e2:baff:fe5a:7ded|   2607:2000:100:116:225:90ff:fe8e:b4a0|   80|56809|  6|         5|       959|        |2015/04/10T13:44:45.170|    0.150|2015/04/10T13:44:45.
320|  0|
  2607:2000:100:116:92e2:baff:fe5a:7ded|   2607:2000:100:116:225:90ff:fe8e:b4a0|  443|55993|  6|         9|      2563|        |2015/04/10T13:44:44.280|    0.200|2015/04/10T13:44:44.
480|  0|
   2607:2000:100:116:225:90ff:fe8e:b4a0|  2607:2000:100:116:92e2:baff:fe5a:7ded|56809|   80|  6|         5|       488|        |2015/04/10T13:44:44.330|    0.150|2015/04/10T13:44:44.
480|  0|
   2002:8891:e71f:8:92e2:baff:fe5a:7ded|   2607:2000:100:116:225:90ff:fe8e:b4a0|   80|45636|  6|         5|       959|        |2015/04/10T13:44:59.330|    0.000|2015/04/10T13:44:59.
330|  0|
   2607:2000:100:116:225:90ff:fe8e:b4a0|   2002:8891:e71f:8:92e2:baff:fe5a:7ded|41424|  443|  6|         9|      1130|        |2015/04/10T13:45:03.330|    0.100|2015/04/10T13:45:03.
430|  0|
   2002:8891:e71f:8:92e2:baff:fe5a:7ded|   2607:2000:100:116:225:90ff:fe8e:b4a0|  443|41424|  6|         9|      2563|        |2015/04/10T13:45:03.330|    0.100|2015/04/10T13:45:03.
```

figure 3.6 Using **rwfilter** and **rwcut** to display IPv6 flow data

3.6 Conversion of IPv6 addresses to numbers of range 0 to 1

To show how to we converted an IPv6 address to a number from 0 to 1 we will demonstrate it with an example. Lets suppose we have this example: 2001:dc0:2001:0:4608**::**25. Adding zeros in the four colons we get: 2001:dc0:2001:0:4608**:0:0:**25 which is the same as 2001dc02001046080025. This is then multiplied by $(2^{16})^i$ where $0 \leq i < 8$.

$2001 * (2^{16})^7 + dc0 * (2^{16})^6 + 2001 * (2^{16})^5 + 0 * (2^{16})^4 + 4608 * (2^{16})^3 + 0 * (2^{16})^2 + 0 * (2^{16})^1 + 25 * (2^{16})^0$

The total of is 4.254076705501262e+37, which divided by 2^{128} equals **0.1250.**

3.7 Implementation

We decided to make a simple program in Python that would read a file with the IPv6 flow data and display the data in a 3d graph. The coordinates for the graph would be the IP Source Address for X, the IP Destination for Y and the Destination (dPort) for Z. The data that was used was the flow data captured from the UPR-RP Network in April 10, 2015.

```
#!/usr/bin/python
#*************************************************************
# flowReader_2.py
# ------------------------
# Grace M. Rodriguez
# May 5, 2015
#*************************************************************
#-- ticketParser.py ---------------------------------------------------------
# This code reads IPv6 flow data from a file with fileinput and converts the address
# in coordinates of range from 0 to 1.
#-----------------------------------------------------------------------------
import fileinput
import math

# /*************************************************************
# Method : readFile()
# Use    : Reads the data from the file written in the command line
#          and saves it in the list "data".
```

```
# Input  : none
# Return : data
# ***************************************************************/
def readFile():
        data = []
        for line in fileinput.input():
                line = line.strip()
                fields = line.split("|")
                fields = [field.strip() for field in fields]
                data.append(fields)
        return data

data = readFile()

# /***************************************************************
# Method : countColons()
# Use    : Takes the address and splits into elements in each colon(:)
#          found. Afterwards, it adds 0 to all the elements in the list
#          if found that they're empty.
# Input  : IPadd (IP address)
# Return : IPadd
# ***************************************************************/
def countColons(IPadd):
        IPadd = IPadd.split(":")
        missing = (8 - len(IPadd)) + 1
        for i in range(0, len(IPadd)):
                if IPadd[i] == "":
                        IPadd[i:i+1] = ["0"] * missing
                if IPadd[-1] == "":
                        IPadd[-1] = "0"
        return IPadd

# /***************************************************************
# Method : poly()
# Use :      It multiplies each element in the list with 2^16, and
#            adds each element together with the index "coeff" in base
#            16. This is to convert the string to int.
# Input  : s (list with blocks of the addresses)
# Return : result
# ***************************************************************/
def poly(s):
        result = 0
        for coeff in s:
                result *= 2**16
                result += int(coeff,16)

        return result
```

```
# /*****************************************************************
# Method : ipv6ToInt()
# Use    : This is used to unite the result of the address after the
#                  necessary 0 are added and convert it to an int.
# Input  : s
# Return : poly()
# *****************************************************************/
def ipv6ToInt(s):
        return poly(countColons(s))

exp128 = 2**128
exp16 = 2**16

#In the list of flow data that we have, we use record[0], which is the
#IP Source Address and record[1] which is the IP Destination Address
#for the dPort, we only have to divide it with 2^16.
for record in data[1:]:
        print
1.0*ipv6ToInt(record[0])/exp128,1.0*ipv6ToInt(record[1])/exp128,float(record[3])/exp16
```

We then made another simple program that takes all of the source and destination's addresses and saved them into a .dot file. This way we could have more visualization in the addresses connectivity.

The first function in the program is literally the same as the **readFile()** in the flowReader.py. The second function, **getAddress()**, gets the IP source addresses and the IP destination addresses and sends them as parameters to the variable DG, which is a function of the method nx.DiGraph. Then with the method nx.write_dot, the addresses are written in a.dot file to make the graph.

```
#*************************************************************
# IPv6graph.py
# -----------------------
# Grace M. Rodriguez
# May 5, 2015
#*************************************************************
#-- IPv6graph.py ------------------------------------------------------------
# This program reads from a file the IP source address and the IP destination
# address and adds to a Digraph variable to turn it into a graph.
#
#-----------------------------------------------------------------------------

import sys
import networkx as nx
from graphviz import Digraph
import fileinput
```

```
# /*****************************************************************
# Method : readFile()
# Use    : Reads the data from the file written in the command line
#            and saves it in the list "data".
# Input  : none
# Return : data
# *****************************************************************/
def readFile():
        data = []
        for line in fileinput.input():
                line = line.strip()
                fields = line.split("|")
                fields = [field.strip() for field in fields]
                data.append(fields)
        return data

data = readFile()

DG=nx.DiGraph()

# /*****************************************************************
# Method : getAddress()
# Use    : Gets the all the IP source addresses and IP destination
#            addresses (flow[0] and flow[1]) and sends them as
#            parameters to nx.DiGraph().
# Input  : none
# Return : none
# *****************************************************************/
def getAddress():
        for flow in data[1:]:
                print "Source ad: ", flow[0], "\tDestination ad: ", flow[1]
                DG.add_edge(flow[0],flow[1])
        nx.write_dot(DG,"graph_v6Flow.dot")

getAddress()
```

In order for IPv6graph.py to work pygraphviz, graphviz and networkx have to be installed in the computer.

Also, since we're using the library **fileinput** to process lines form input streams, we have to write the name of the file we want the data read in the command line. For example, for flowReader_2.py the command line would look like this

$ python flowReader_2.py ipv6Flow.txt

The flowReade_2.py can be found at https://github.com/gracemarod/loveChocolateAndCats/tree/master/Techniques_for_Anomaly_Detection_in_IPv4_and_IPv6_Network_Flows .

4. Results:

At the end, we managed to get a file with IPv6 flow data from the UPR's network. In only one day, in the time between 1:44pm to 11:56pm, we got 9858 flows. This is what the head of the file looks like:

	sIP	dIP	sPort	dPort	pro
1					
2	2607:2000:100:116:225:90ff:fe8e:b4a0	2607:2000:100:116:92e2:baff:fe5a:7ded	57401	22	6
3	2607:2000:100:116:225:90ff:fe8e:b4a0	2607:2000:100:116:92e2:baff:fe5a:7ded	55993	443	6
4	2607:2000:100:116:92e2:baff:fe5a:7ded	2607:2000:100:116:225:90ff:fe8e:b4a0	80	56809	6
5	2607:2000:100:116:92e2:baff:fe5a:7ded	2607:2000:100:116:225:90ff:fe8e:b4a0	443	55993	6
6	2607:2000:100:116:225:90ff:fe8e:b4a0	2607:2000:100:116:92e2:baff:fe5a:7ded	56809	80	6
7	2002:8891:e71f:8:92e2:baff:fe5a:7ded	2607:2000:100:116:225:90ff:fe8e:b4a0	80	45636	6
8	2607:2000:100:116:225:90ff:fe8e:b4a0	2002:8891:e71f:8:92e2:baff:fe5a:7ded	41424	443	6
9	2002:8891:e71f:8:92e2:baff:fe5a:7ded	2607:2000:100:116:225:90ff:fe8e:b4a0	443	41424	6
10	fec0::8:225:90ff:fe8e:b4a0	fec0::8:92e2:baff:fe5a:7ded	59307	22	6

pro	packets	bytes	flags	sTime	duration	eTime	sen
6	6	450		2015/04/10T13:44:31.326	0.100	2015/04/10T13:44:31.426	0
6	9	1130		2015/04/10T13:44:45.120	0.200	2015/04/10T13:44:45.320	0
6	5	959		2015/04/10T13:44:45.170	0.150	2015/04/10T13:44:45.320	0
6	9	2563		2015/04/10T13:44:44.280	0.200	2015/04/10T13:44:44.480	0
6	5	488		2015/04/10T13:44:44.330	0.150	2015/04/10T13:44:44.480	0
6	5	959		2015/04/10T13:44:59.330	0.000	2015/04/10T13:44:59.330	0
6	9	1130		2015/04/10T13:45:03.330	0.100	2015/04/10T13:45:03.430	0
6	9	2563		2015/04/10T13:45:03.330	0.100	2015/04/10T13:45:03.430	0
6	6	450		2015/04/10T13:45:08.330	0.250	2015/04/10T13:45:08.580	0

figure 4.1 and 4.2 Screenshot of ten flows from the ipv6Flow.txt

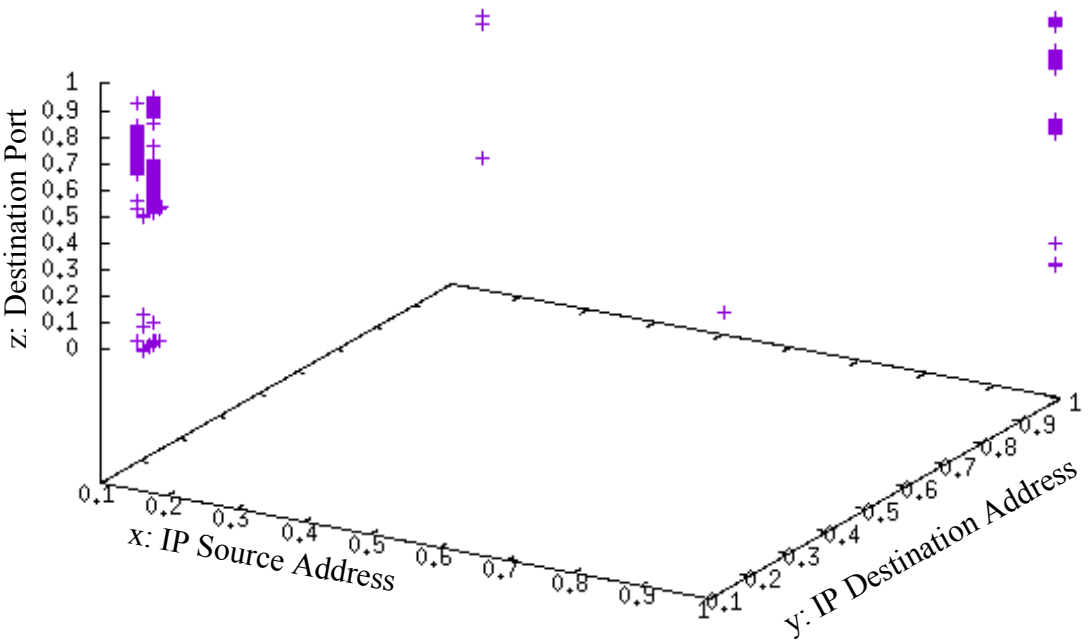
After reading and getting the data from this file with flowReader_2.py, we were able to convert the IP source and destination addresses and the dPort into x, y and z coordinates, respectively.

```
0.148546219  0.148546219  0.000335693
0.148546219  0.148546219  0.006759644
0.148546219  0.148546219  0.866836548
0.148546219  0.148546219  0.854385376
0.148546219  0.148546219  0.001220703
0.125038658  0.148546219  0.696350098
0.148546219  0.125038658  0.006759644
0.125038658  0.148546219  0.632080078
0.995117188  0.995117188  0.000335693
0.995117188  0.995117188  0.001220703
0.995117188  0.995117188  0.744033813
```

Figure 4.3: Screenshot of the coordinates for the first 10 flows shown in figures 4.1 and 4.2.

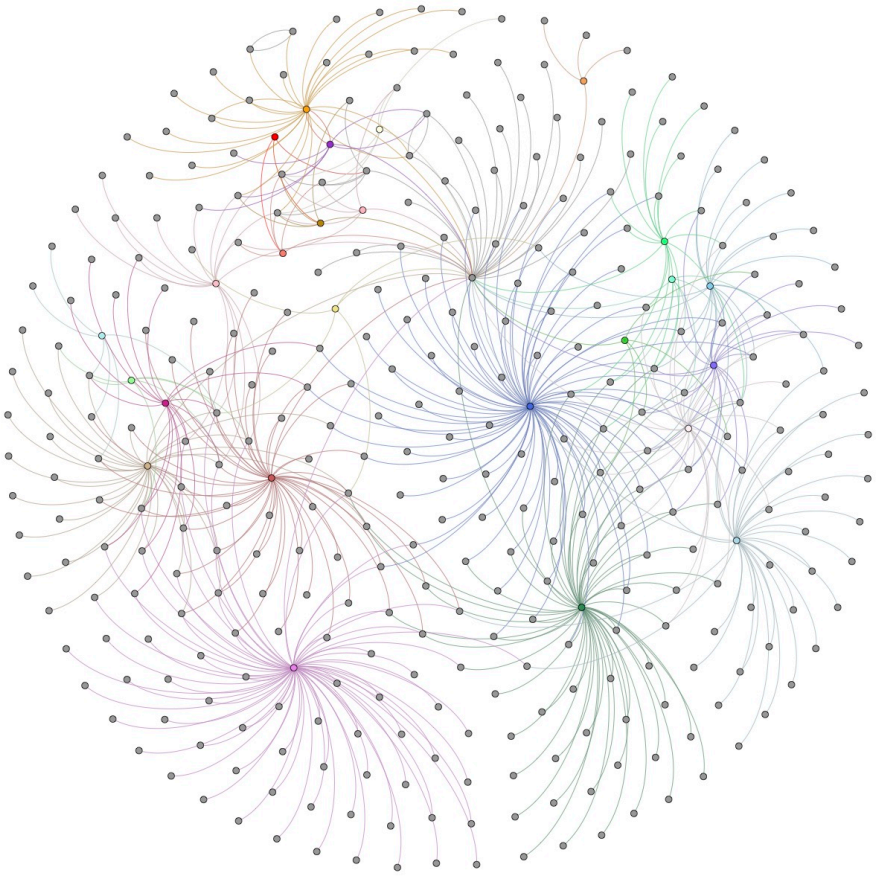
As seen, the coordinates for the first five flows are the same because these flows have the same first 64 bits. With this coordinates, we were able to make a 3d graph using *gnuplot*.

Figure 4.4: IPv6 Flow Data of the UPR-RP from April 10, 2015



In the graph, we can see that the most concentrations of points are in 0.1,0.1,0 (x, y, z) and 0.2,0.2,1. This is because these points are mostly the coordinates for the UPR's addresses. The other concentration of points in 0.9,1,0 to 1,1,1 is the coordinates for external addresses. The other few points in the other ranges are also external addresses. Afterwards, we used IPv6graph.py to have better visualization on the addresses connectivity. After making the .dot file, we opened it in the program *Gephi.*

Figure 4.5: Force Directed Layout of IPv6 Connectivity Graph



Apart from looking pretty, figure 4.5 shows with more ease the IP Source Addresses that connect the most to other IP addresses. All of the circles with colors are IPv6 addresses from the UPR-RP. The rest of the circles, which are in gray, are IPv6 addresses outside the UPR. This graph doesn't show the amount of times each address made a connection to another address, it just shows all the different connections there were in that day. With this graph, we can find which external addresses did different networks in the URR connected to the most. We can later study why they were many different addresses connecting to those particular addresses, and check if those external addresses aren't malicious. In figure 4.6 and 4.5 we can see how the UPR-RP networks connect to many networks outside the UPR.

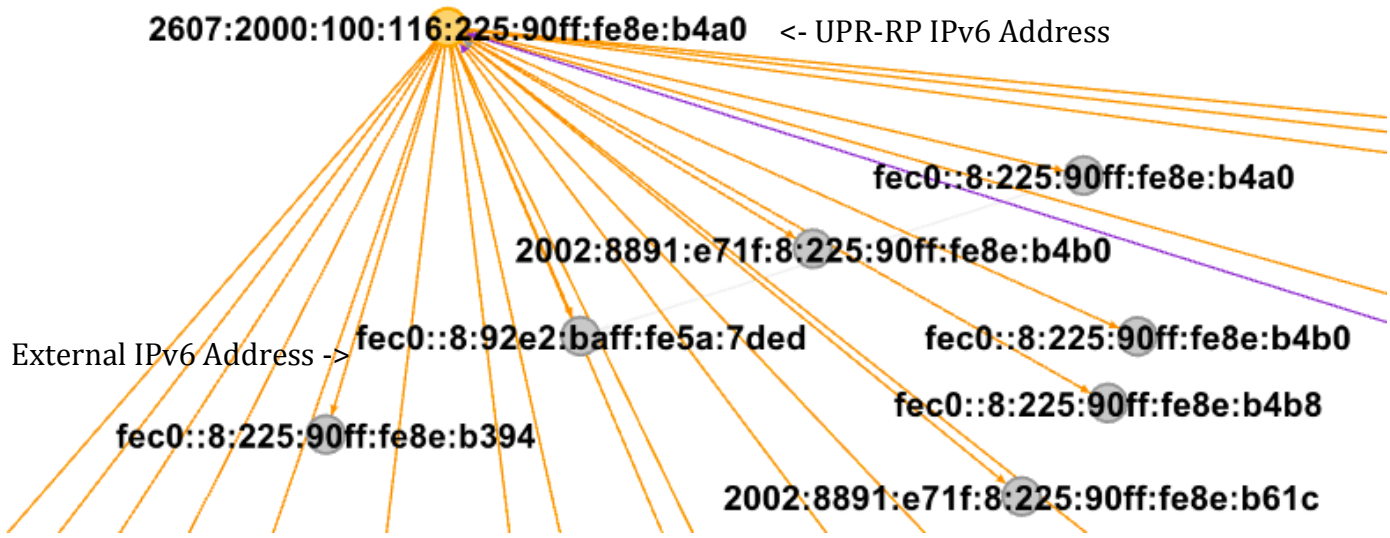


Figure 4.6: Close up of Force Directed Layout of IPv6 Connectivity Graph

There was one special case that we found in the Ipv6 addresses were we have to modified the flowReader_2.py script. There is one particular address outside the UPR named 2607:fc18:bad:dead:beef:: Since it has two colons at the end, we have to specify the program to replace them with a cero every time it found them at the end of the address. Upon searching it in Google, we can assume it's an address from the USA.

5. Future Plans:

For the future, we would like to keep exploring other ways to analyze flow data and detect anomalies in IPv6 flows. It would be beneficial to continue analyzing IPv6 flows because there isn't much research that concentrates in IPv6. It would be a good contribution for network analysis and security to learn more about using and analyzing IPv6 data.

We would also like to make more code implementations that separates the flow data according to the amount of each field. In this case, we concentrated on displaying the flow data in a graph according to the IP addresses. Another implementation that can be done to help us analyze the flow data better would be with separating the data depending of its amount of packets, bytes, date, etc. It would also help us compare the difference between IPv4 and IPv6 data.

As we learned, the data is easier to understand if its being displayed in visualization tools such as graphs. Another work we would like to do in the future is to make an implementation that feeds the graph the desired data dynamically. That way, we can see with more detailed how the graph grows or lowers according to its coordinates.

6. Conclusion:

SiLK is a very useful tool to help detect anomalies, but since it's a very big collection, it has a lot of methods that still need to be learned. We learned how to use the command rwfilter to helps us gather IPv4 and IPv6 data. With this we learned that IPv6 is still being used a lot less than IPv4 in the UPR's network. Therefore, the networks that use IPv4 are more exposed to malicious attacks than IPv6.

After implementing the program with the IPv6 data, we realized there can be a big amount of flows for only one day. Therefor, it would be almost impossible to try and analyze each flow one by one. It is then more practical to write a code that reads the data and organizes it depending on its data type, like we did with the Python scripts in section 3.6. It is also easier to understand when the data is displayed in a graph because we can see all of the data in more detail.

7. Acknowledgements:

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.

We also want to thank our research advisor, Dr. Humberto Ortiz-Zuazaga, for pointing us in the direction when we were lost with the topic, especially when learning SiLK tools and helping us answer our doubts in the implementations.

We also want thank Dr. José Ortiz Ubarri for bringing the ATACK-PR project to the UPR-RP with Dr. Humberto.

Finally, we want to thank the other members in the lab that also help us with the programs and codes, especially Omar Rosado, Ricardo Augusto López, Edwardo Rivera and Felipe Torres.

8. References:

[1] Bandes, Ron, Timothy Shimeall, Matt Heckathorn, and Sidney Faber. *Using SiLK for Network Traffic Analysis*. Vol. Analyst's Handbook for SiLK Versions 3.8.3 and Later. Pittsburgh: Carnegie Mellon U, 2014. Web. <http://tools.netsa.cert.org/silk/analysis-handbook.pdf>.

[2] Colon, Bianca, and Humberto Ortiz-Zuazaga. "Techniques for Anomaly Detection in Network Flows." (2014). 17 May 2015.

[3] Fullmer, Mark. "Flow-tools." *Flow-tools(1) - Linux Man Page*. Web. 17 May 2015. <http://linux.die.net/man/1/flow-tools>.

[4] Garcia, Ivan O., and Humberto Ortiz-Zuazaga. "Techniques for Anomaly Detection in Network Flows." (2013). 17 May 2015 <http://ccom.uprrp.edu/~humberto/research/anomaly-detection.pdf>.

[5] Fullmer, Mark, and Steve Romig. 'The OSU Flow-Tools Package And Cisco Netflow Logs'. *Usenix.org*. N.p., 2015. Web. 17 May 2015.

<https://www.usenix.org/legacy/publications/library/proceedings/lisa2000/full_papers/fullmer/fullmer_html/index.html>.

[6] "IPv6 Tutorial." *IPv6 Tutorial*. Tutorials Point (I) Pvt. Ltd., 2014. Web. 17 May 2015. <http://www.tutorialspoint.com/ipv6/>.

[7] Li, Bingdong, Jeff Springer, George Bebis, and Mehmet Hadi Gunes. "A Survey of Network Flow Applications." *Journal of Network and Computer Applications* (2013): 15. *Journal of Network and Computer Applications*. Elsevier. Web. 17 May 2015. <http://www.journals.elsevier.com/journal-of-network-and-computer-applications>.

[8] Patcha, Animesh, and Jung-Min Park. "An Overview of Anomaly Detection Techniques: Existing Solutions and Latest Technological Trends." *Computer Networks* (2007): 3448-470. *Science Direct*. Elsevier. Web. 17 May 2015. <http://www.sciencedirect.com/science/article/pii/S138912860700062X>.

[9] Quittek, Juergen, Tanja Zseby, Benoit Claise, and Sebastian Zander. "RFC 3917 - Requirements for IP Flow Information Export (IPFIX)." *RFC 3917 - Requirements for IP Flow Information Export (IPFIX)*. The Internet Society, 2004. Web. 17 May 2015. <https://tools.ietf.org/html/rfc3917>.

[10] "SiLK." *SiLK*. CERT Network Situational Awareness (CERT NetSA). Web. 17 May 2015. <https://tools.netsa.cert.org/silk/credits.html>.