**REPORT RESEARCH INTERNSHIP I**

**PARTI**

The purpose of this part was to detect any legitimate abnormal behavior from the testbed gateways

The first part of this work consisted on finding out the best location for the Snort sensor. In our case SNORT does not need to run in intrusion prevention mode, mirroring the network traffic from the testbed network to a server port was more than enough.

In our first attempt, the snort sensor was a server that had a port where all the testbed traffic was dumped. In our first attempt, traffic from the gateways to the INTERNET was not captured because all traffic from the testbed was tunneled directly from the university router to the firewall.

In our second attempt, the snort sensor was one of the university routers. To be able to test the rules, we set a dynamic port forwarding connection from a remote computer to the testbed gateway (node71). From the remote computer we generated traffic in order to fire alerts in case abnormal behaviors were detected on the gateway.

**1. Applications' detection**

Below are the three rules that were tested:

**alert tcp $HOME\_NET any -> any any (msg:"Facebook"; content:"facebook"; nocase;sid:100001; rev:001;)**

**alert tcp $HOME\_NET any -> any any (msg:"youtubeConnect";content:"youtube";nocase;sid:100004; rev:1;)**

The first rule fires an alert every time a TCP packet from the router home network (143.129.85.0/23) contains the word “facebook”. During a TCP handshake, when a client connects to Facebook. The request is forwarded to the Facebook servers. In Wireshark any request to facebook.com generates a payload witch contains the string “facebook” which is the name of one of Facebook servers.

The second rule is just like the first one, but in the second one Youtube is used instead of Facebook.

Snort provides a plug-in to detect various types of applications like Skype, yahoo, yahoo messenger etc.. As typing rules for all those applications would be counter productive, the following week will be dedicated to configuring that plug-in (openappid) into SNORT.

**2. Heavy payload detection**

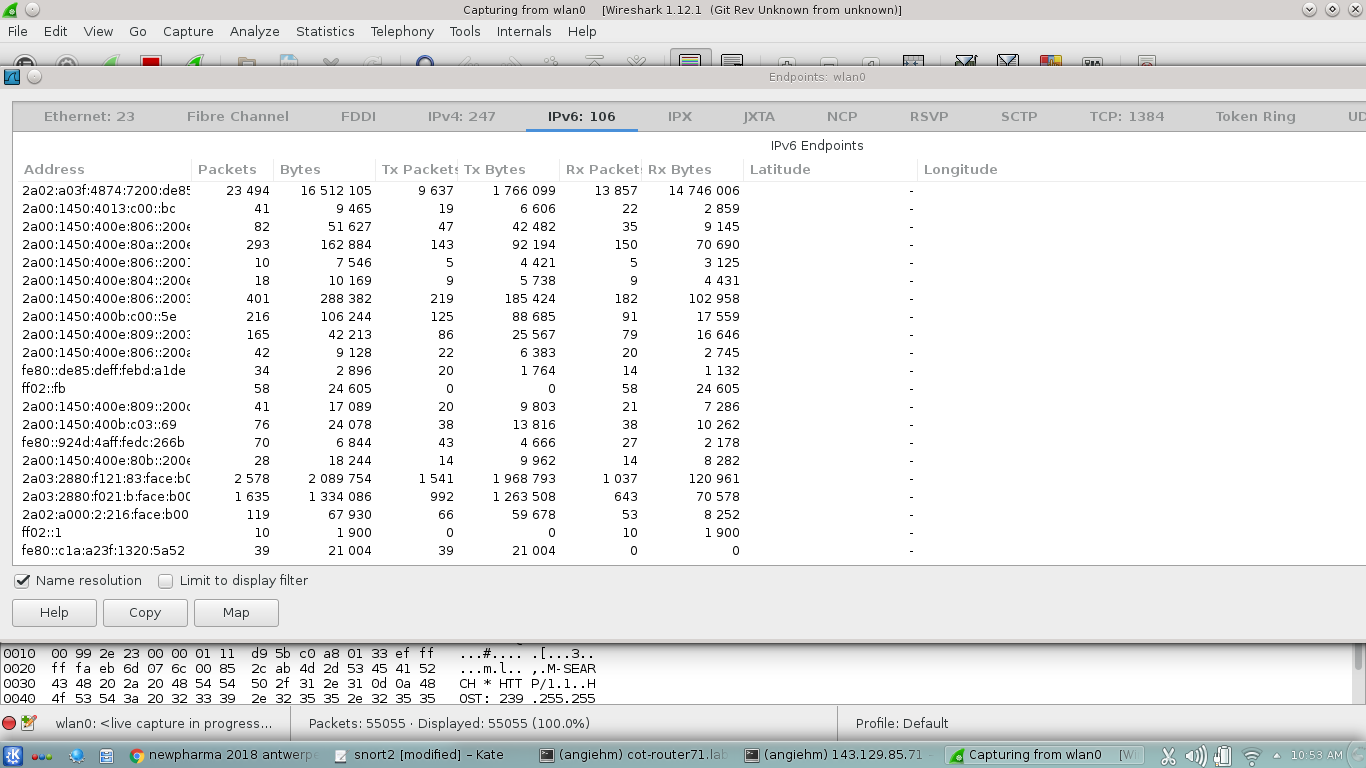


Fig1. Traffic after 5 minutes

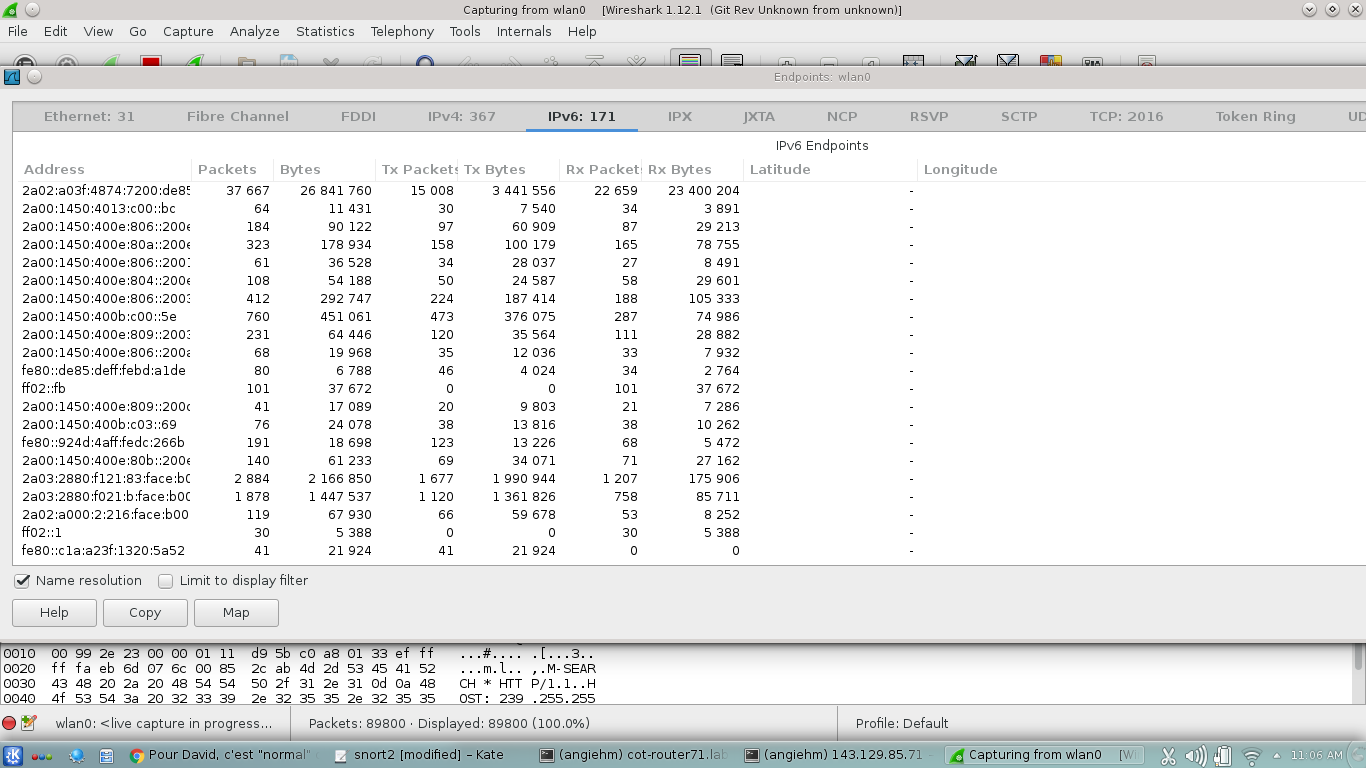


Fig2. Traffic after 15 minutes

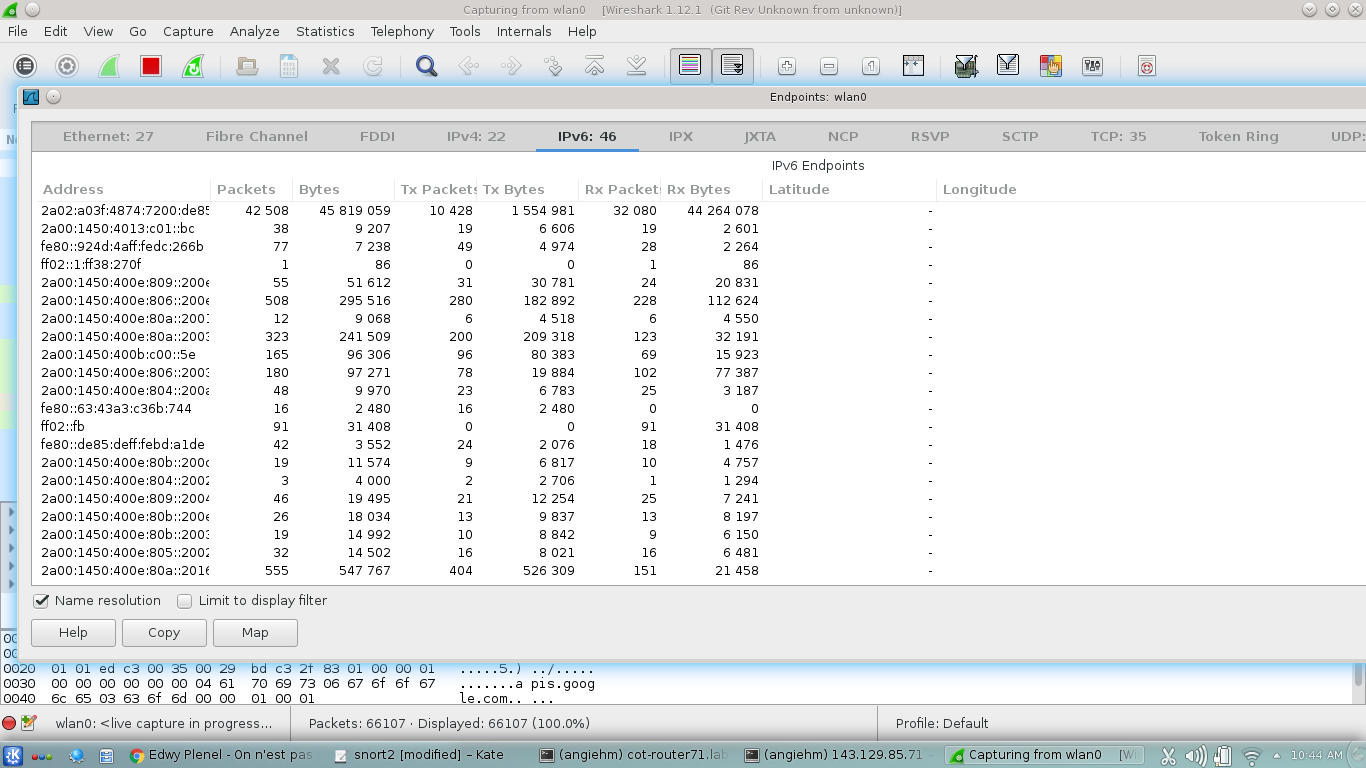


Fig3. Youtube traffic

The traffic from figure1 was generated when visiting numerous websites and the RX bytes seen at the top of the figure correspond to the bytes received after 5 minutes of browsing websites such as yahoo, Google and other sites.

The traffic from figure2 corresponds to the traffic after 15 minutes of browsing.

The last figure corresponds the bytes received after some few minutes of watching online Youtube videos.

From the traffic generated in the three cases, as shown in the above figures, we wrote the following rules.

**alert tcp $HOME\_NET any -> any any (msg:"Talkative host";flow:to\_client,established;stream\_size:server,>,22000000;threshold:type limit, track by\_src, count 1 , seconds 840;sid:100005; rev:1;)**

**alert tcp $HOME\_NET any -> any any (msg:"Super talkative host";flow:to\_client,established;stream\_size:server,>,40000000;threshold:type limit, track by\_src, count 1 , seconds 120;sid:100005; rev:1;)**

**alert tcp $HOME\_NET any -> any any (msg:"Huge payload possible download";flow:to\_client,established;stream\_size:server,>,80000000;threshold:type limit, track by\_src, count 1 , seconds 120;sid:100005; rev:1;)**

The first rule is fired if a particular user is receiving bytes above 22 M in an interval of 15 minutes (840s).

The second rule is received if a user received more than 40 M of bytes in an interval of 2 minutes.

The last rule is for a user who is probably watching on line videos or downloading huge files.

The stream\_size is used in snort to detect the number of bytes observed as determined by the sequence numbers. The Snort sensor keeps listening on the interface for the duration of a particular TCP session and every time it detects a stream that is greater than a specified number of bytes it fires an alert.

All the alerts generated are saved in snort log files.

**PART II**

For the analysis of sensor data, we have chosen Hadoop. Hadoop is a batch data processing framework that stores and processes big data. Thanks to its distributed storage system, Hadoop takes the storage overhead off the shoulder of one system. Hadoop is very fast in processing data because it uses the Map reduce algorithm in order to process and sort sensors data. It also uses parallel processing of data which consists on dividing the processing of data among multiple works.

For the Experiment we are going to work with the Hortonworks sandbox. The Hortonworks sandbox is a platform that allows the use and integration of all apache big data analysis frameworks like apache Stark apache Hive apache Storm etc.. Besides it runs in a virtual box allowing the user to run the program without making permanent changes to the host operating system thus preventing damages in case something goes wrong.

After querying the city of things back end for the sensors data. The data will be fed into the Hadoop framework using apache Kafka which is also included in the Hortonworks sandbox. The data will be analyzed by the Hadoop YARN-mapReduce and the results will be stored in the Hadoop HDFS.