OSU – Defense Against the Dark Arts

Week 6, First Lab (aka “Lab2”)

# Header Analysis

We often wish to examine network traffic to analyze whether a given network is running correctly, is under attack, has secrets in it we would like to reveal or protect, and so on. There is an engineering tradeoff here. We can’t store and analyze all the data that flies by in the network. So we would like to use a portion of the total data to decide where to look deeper.

Often the initial security analysis uses protocol headers or security and application logs.

* This information is small, so it can be stored for more traffic over a longer period of time
* This information can be determined from packets flying by with a minimum of processing, so it is possible to get the info without either spending a fortune in monitoring hardware or slowing down the network traffic

There is a perception that the content data is “sensitive” and that header data is “anonymized” or information that people don’t care as much about. If you had time to read the Kieran Healy’s amusing article on “Using Metadata to Find Paul Revere”, you can see that this is not always a valid perception.

Let’s see what we can find from the most basic packet header data.

The trick here is to rearrange the data in various ways and look for ‘spikes’ in frequency. Then look at what all the frequent items have in common, such as the same protocol, similar IP addresses, etc. To do this well, you need to know about common TCP and UDP services. Fortunately, we have Google to help us with that!

In this lab, you are provided with two samples of packet data, entitled ‘R’ and ‘O’. The packet data is in order, and covers a brief period of time. For each packet, you get:

* Length, IP type, IP Source Address, IP Destination Address
* TCP: Flags, Source port, Destination Port
* UDP: Source port, Destination port
* ICMP: type, code

The data is stored in a textual CSV format (you can even look at it using a terminal or read it into a spreadsheet). In this assignment, you will write scripts to analyze the data. We have provided starter scripts in PERL and PYTHON, entitled scancsv.{pl,py}. You can either language, or another, if you want to rewrite the starter script.

Hand in your script and the script output, as requested in the numbered assignments, below.

For the record, the field names (line 1 of the csv) are: *len, proto, ipsrc, ipdst, tcpflags, tcpsport, tcpdport, udpsport, udpdport, icmpcode, icmptype*

The R data (100k packets) is a lot smaller than the O data (1M packets). Prototype your scripts using R data, then switch to O data to apply the concepts on a larger data set.

The ultimate goal of this assignment is to understand the function of the networks monitored in the R and O data sets. What is their function? We will choose among: work, home, data center, ISP. On the way, we’ll learn as much as we can about these networks from the scanty data that we have.

The starter scripts collect statistics based on IP protocol numbers:

user@kali:~/Lab2$ python scancsv.py R.csv

Num packets: 99142, Num bytes: 71683046

IP Protocols:

1: 7

2: 2

6: 39138

17: 59995

You can look the IP numbers up in **/etc/protocols** to find out that IP protocol 1=icmp; 2=igmp; 6=tcp; 17=udp. For example, ICMP (IP Protocol 1) occurs in 7 packets. You will have learned a little of these protocols from the advance reading. If not, google them now!

Each protocol has its own uses. For example, IGMP (proto=2) is used for router-to-router communications. So we could find the addresses of routers by looking at flows that use this protocol. Without writing a script to do this, we could just use the grep command:

user@kali:~/Lab2$ grep -e len -e '^[0-9][0-9]\*,2,' R.csv

len,proto,ipsrc,ipdst,tcpflags,tcpsport,tcpdport,udpsport,udpdport,icmpcode,icmptype

28,2,**10.5.63.36**,234.42.42.42,,,,,,,

28,2,**10.5.63.36**,234.142.142.142,,,,,,,

*(note: the first -e argument to grep preserves the CSV label line)*

We can now infer that 10.5.63.36 is a router IP address. The other addresses that start with 234.\*.\*.\* are multicast addresses (put them into google to see this).

## Find Statistics on TCP and UDP Services

Most interesting user protocols run on TCP although a few run on UDP. This usually works as follows:

* A server process listens on a given TCP or UDP port number. By convention, these “well known ports” are allocated from 1-1024 (some are outside that range by now). A well-known port is often called a *service.*
* Clients connect in to the server using this as the TCP or UDP *destination port*.

The file */etc/services* contains a listing of well-known port numbers for TCP and UDP.

1. Extend your script’s statistics gathering to count the use of all well-known destination port numbers for TCP and UDP (ports 1-1024). For example, you should be able to look up in your output how many TCP packets have destination port 80 and how many UDP packets have destination port 53. Run your new script on R and O data. Enable this function using a ‘-stats’ flag (i.e., the script should have no output unless there is a -stats flag in the command line).
2. Based on this information, characterize the main functions on each network. What kind of a network is it? (e.g., work, home, data center, ISP)

# Investigate IP Addresses

1. Add to your script an option called “-countip” which creates list of distinct IP addresses with their usage counts. Sort the list by the usage count, not by the IP address.
2. Run your countip script on R and O data. Does this inform your answer in [2]?
3. Attempt to determine the network number (network prefix) that seems to dominate the traffic.

There are some IP protocols that are typically used between routers or other special networking devices. Traffic from these protocols can identify the infrastructure of the network under observation.

1. Generate sorted output from ‘-countip’ for the IP protocols to identify all the IP addresses that use:
   1. GRE (Generic Routing Encapsulation) – this is used to create tunnels between networks with overlapping address spaces. It is also the base protocol for PPTP, a remote access mechanism.
   2. IPSEC – this is the protocol that creates virtual private networks, creating an overlay network structure on top of the Internet. Most IPSEC is router-router these days.
   3. OSPF – Open Shortest Path First routing protocol. This is the ‘standard’ routing protocol for Internet routers, allowing them to discover the topology and choose the best routing paths as connections between routers appear and disappear.
   * Hint: create a new protocol argument to filter the data to ‘-countip’ to just include lines for these protocols. Alternately, use the GREP pipeline in the example above, for IGMP traffic, and change ‘2’ to the right protocol number.
2. Find another network prefix that also seems to be associated with this traffic.
3. Does the OSPF information inform your answer to question 2?

# Find the servers

The server machines are the main assets of each site. Can we find them?

Imagine drawing a diagram for each service, with a line from each client machine that uses the service to the server that provides it. Then the servers look like stars, with clients around each server. Important servers will stick out as stars with lots of spokes.

Client 1

Client 2

Client 3

Client 4

Client 5

Client 6

tcp/80

tcp/80

tcp/80

tcp/80

tcp/80

tcp/80

To find the ‘stars’ in the diagram, we want to find IP addresses that are the *destination* of many transactions.

1. Add an option to your script ‘-connto’, which counts the number of packets sent to each service (ports 1-1024) on the network. For example, a dictionary maps each *ipdst* to the tuple <*proto*, *dport*>, where *proto* is **tcp** or **udp**, based on the IP protocol (6 or 17) and *dport* is the value of tcpdport or udpdport.
   * Sort the output by the number of distinct source IP address – source port combinations, so that servers which serve a lot of different connections all cluster at one end of the output.
   * For output, generate a summary line that shows, for each destination IP address, how many distinct source IP addresses accessed it, and what ports were referenced:

ipdst 1.2.3.4 has 334 distinct ipsrc on ports: udp/53, tcp/80, tcp/443

ipdst 5.6.7.8 has 335 distinct ipsrc on ports: tcp/22, tcp/25

…

⇨ Since lab time is short, here are some programming hints you may wish to use:

* 1. To create the ports output, create a set for each “ipdst” that contains the string “udp/” or “tcp/” appended to the port number (e.g., udp/53, tcp/360). (In languages without an explicit set class, use a dictionary or hash where each entry maps to TRUE or 1)
  2. You can use the same trick to compute the distinct ipsrc for the summary line. In this case, put the ipsrc in the string from [a], as *ipsrc-proto/port*. For example, dict['1.2.3.4'] is a set containing 205.9.3.55-udp/53, means that 205.9.3.55 connects to 1.2.3.4 on UDP port 53.
  3. You can use leading zeros to make these formats sort correctly without fuss, such as: "tcp/00033" or "udp/00721". *However, you must still arrange for the program output not to have leading zeros.*

1. Run your -connto option on R and O data (ignore anything that ends in .255 – this is a broadcast address). Does this suggest a set of servers to you?
   1. Return the top 20 servers from your ‘connto’ output.
   2. For the R data, identify the web servers, the printers, the mail servers, the DNS servers
   3. For the O data, identify the mail servers, the pop/imap servers, the DNS servers
2. Update your answer from [5] based on this information.

# For More Fun, if you finish early:

In the O data, some of the GRE lines actually have the following form:

len,greproto,ipsrc,ipdst,iproto,iipsrc,iipdst

where greproto is 47, and iproto, iipsrc and iipdst describe the packet being transmitted inside the GRE tunnel.

Extend your program to identify these lines and see what you can learn from this additional information.