COMS 4180 Network Security Programming Assignment 2

Due Friday, April 14th, 2017, 10:00pm Eastern time.
NOTE:
This due date is 2 days later than what was in the schedule in lecture 1.
This assignment is worth more points than the first programming assignment.

200 points
The homework is to be done individually.
NO LATE HOMEWORK WILL BE ACCEPTED.

What to submit: A zip file or tar file file containing:
a text or pdf file with your answers for Problem 1, your code for the ngram analysis for Problem 1, and the code, input and output as described in Problem 2
Submit via courseworks.

Problem 1 (100 points total for your code and answers to questions a,b,c)

This problem is a combination of using existing tools and writing a program to do ngram analysis. The ngram program must run your Google Ubuntu VM. You may use any programming language. The other tools can be run on any machine (strings requires Linux).

Perform static analysis using strings, ssdeep and ngram analysis on the six binaries prog1,prog2, prog3, prog4, prog5 and prog6 to determine if any of the executables are similar and information about the executables.

ssdeep is available from http://ssdeep.sourceforge.net/#download. Downloads are available for Windows and Linux under Stable Version from April 2015. The source code is for Linux. Read the Install file for installation instructions.

Write your own program that can count ngrams for n <= 3 with a sliding window s of 1 to n, i.e.

n = 1 byte (s = 1). This is just computing the byte distribution.

n=2 bytes with a sliding window s of 1 and 2.

n = 3 bytes with a sliding window s of 1,2,3.

The program must take exactly 4 inputs:

- an integer n that is the length of the ngrams
- an integer s that is the length of the slide
- the name of the file to analyze
- the name of the output file

There are no spaces in the filenames. Your program must not try to read the entire binary into memory then compute the counts but instead it should process bytes as they are read in.

When processing the binary to determine ngram counts, you need to work with bytes. Don't convert individual bytes into some form that takes up more space than a byte (such a string of length 8 containing the characters 0 and 1, or using an entire integer to represent 1 byte).

The program should output any information you decide is relevant to the extent that you have enough information to answer the questions. For example, you may decide not to output ngrams that occur only once or account for some negligible percent of the overall number of non-distinct ngrams, but instead summarize such ngrams by saying x distinct ngrams occur one time or account for < p% of all ngrams. For obvious reasons, do not try to store and print ngrams that do not occur.

The upper bound of 3 on n for this assignment is to limit the need for developing a program that is efficient enough to handle larger values of n.

For the case of n = 1, you may have a data structure (such as an array) with 256 entries containing counts of each byte value (the ith entry in the array is the count for the ith byte value). For n = 2 and 3, you will need to determine an efficient way of counting. Do not try to store counts for all possible 3-byte values. While it is feasible to store counts for all 2 byte values, it is not efficient.

You will turn in your code along with a README file saying how to compile and run it, and that describes what data the program outputs. If applicable, include a Makefile. Also include in the README approximately how long it takes your program to run on the largest of the 6 files. Your program will not be graded on the time it takes to run.

Do not include any sample output files from the program. Any output you use to answer the written questions below should be included in the file with your written answers.

a. (10 points) For each executable, in what programming language do you think it is written in and why?

b. (30 points) For each possible pairing of programs, state whether or not they are similar in any way and explain how you know from your static analysis. You may include output from the analysis but do not include the entire output of strings on each executable – only include specific strings and a general description/categorization of any subsets of strings that are relevant in explaining if two executables are similar or not similar. Also do not include the entire output of your ngram program. Only list specific ngrams as needed in your discussion. It may be useful to calculate the percent of specific ngrams out of all ngrams (this can be done external to your program or you may have your program compute it) when comparing the ngram distribution between executables.

Any Other?

c. (60 points) This includes the points for the code and having the correct output.) For each program,

- list the 20 bytes (1-grams) in hex that occur the most along with a count of each
- list the top 20 2-grams in hex along with a count for each for a slide of 1 and a slide of 2 (there are 2 top 20 lists of 2-grams for each program)
- list the 20 3-grams in hex along with the count of each for a slide of 1 and a slide of 3 (there are 2 top 20 lists of 3-grams for each program).

If there is a tie for the 20th most common ngram, include the one with the smallest hex value (0x00 is smallest, 0xff is largest when n = 1). If there are not 20 distinct ngrams that have a non-zero count, just list the ones that occur.

You may extract the list of the 20 most common ngrams from your program's output in whatever way is easiest for you. For example, if your program outputs any ngram occurring more than t times, for some t, along with its count, you could run a shell command from the command line to sort your output file on the count column or you could have your program identify the top 20.

Problem 2 (100 points total: 20 points for part a, 50 points for part b, 30 points for part c)

Use scapy for the following. If you are not familiar with python, you will need to look up some basic python syntax for reading input from a file and from a command line. You need to be root to send packets. Scapy 2.X was described in class, it works with Python versions 2.X. Use your Google VMs and/or laptop for these problems.

a. Create and send an ICMP (ping) message between two machines for which you know the IP address and which will respond to pings. Also have scapy show the response (i.e. use answer = sr1(<packet> in scapy to get the response and then show the answer. Monitor the packet being sent and the response by running tcpdump with the –vv –XX options (tcmpdump –i eth0 –vv –XX). You may either run scapy interactively or write a python script that calls the scapy functions to send the ICMP and receive the response. Include your scapy script (either the python file or a transcript of what you typed in scapy's interactive mode. Copy the text output from tcpdump for the ICMP message and the response (do not include any other messages that were captured and do not include a pcap file) and include it in a text file in your submission.

b. Write a python script that uses scapy to send an TCP/IP packet containing an HTTP GET message to port 80 on your laptop or port 80 on a VM (in order to avoid sending possibly malformed traffic to a machine you don't control, don't send the packet to an IP address belonging to a system you do not own).

raw 输入?

First, visit a website from your laptop and capture the traffic using Wireshark. Select a site that uses HTTP (not HTTPs). The ssdeep web page uses HTTP. Find the HTTP GET message in the capture. Your script will send the same get message, except the source and destination IP addresses will change to correspond to the machines you are using. Your program should read source and destination IP addresses, and source and destination port numbers and message payload (the HTTP GET message) from a file called inpartb.txt to populate the parameters used by scapy when forming the packet. The program should display the packet using "show" and also print the string representation using str() before sending the packet. Use Raw(load=<http get message contents>) to include the HTTP GET message in the packet.

Run tcpdump with the –vv –XX options to monitor that the packet is sent. Copy the text output from tcpdump from the terminal window for the single packet. Manually compare it to the original packet observed in Wireshark to verify the HTTP portion is correct.

Include your input file, python script and tcpdump text output in.txt in your submission. Do not include a pcap file.

c. Write a python script that uses scapy to do the following. The program will take the source and destination port numbers as command line input (the destination port is needed in part 2, the source port is used for both parts). The loop back address will be hard-coded 127.0.0.1.

(1) First send TCP/IP packets to the loopback address in which the destination port ranges from port 3000 to 3020 (i.e. use a loop that increments the dport # in the TCP portion of the packet). The source port is the port specified on the command line. The payload is empty.

(2) Second, holding the port numbers constant to those specified on the command line, send 5 packets with a random 10 character string as the payload (use Raw(load=<string>) to include the string in the packet. If you are not familiar with creating a random string in python you may hard-code a list of strings in a list and iterate through the list.

Use "send" in scapy to send the packet (as opposed to "sr" or "sr1"). You can use tcpdump to monitor the loopback interface. You do not need to include the tcpdump output in the submission. You should see the packets go out on the loopback interface. There will be no response to any of the packets.