**Lab 03 - CSCI 5742 - Cybersecurity Programming - Packet Sniffing**

Name \_\_\_\_\_\_\_\_\_\_\_**Lucas Fulmer**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Goals:

* Understand how individual bits form structs that make up TCP/IP and TCP ICMP protocols
* Understand how to send raw traffic to create an ICMP scanner
* Use scapy library to make your packet sending and receiving easy
* Understand concepts in packaging your python code

Due to Canvas: Lab Write-up, , sniffer\_ip\_header\_decode.py, sniffer.py, 1 screen shots, sniffer.txt

Work in teams of 2 (or 3 max). Please work at the pace of your lab partner. Each of you is responsible for doing every step and handing in your own deliverables. Print your responses (at > ) in **BOLD BLUE** text. Put corrections in **BOLD RED**

**Part I - IP Packets as structs**

Open Kali in NAT. Set the memory to 60% of your overall memory.

Go to the terminal and switch users to root (su)

First we want to see what version and architecture we are running with Kali Linux.

* Note the Debian Version, and whether you are using a 32 or 64 bit architecture (Remember Kali is just a Debian Linux Distribution)…ask google if you don’t know the command.

**64 bit**

* Write a quick python script (ctypesizes.py) to determine on Kali Linux the size in bytes of (**sizeof**) the following ctypes: **c\_ubyte**, **c\_ushort**, **c\_uint**, **c\_ulong**
* List the sizes here:
* c\_ubyte ---- **1**
* c\_ushort --- **2**
* c\_uint --- **4**
* c\_ulong --- **8**
* Based on Figure 3-1, how many (8 bit) bytes are in an Internet Protocol header?

**20**

* How many bytes would your ip (Structure) on page 40 have in it as written?

**28**

* Make sure to change the "src" and "dst" to the 32 bit c\_uint vs the 64 bit c\_ulong (we are using the 64 bit Kali Linux vs the 32-bit used in the book). If you don't then your packet will have too many bits.
* Add your name at the top of the output, run it and take a screen shot. Upload that screen shot and your sniffer\_ip\_header\_decode.py to Canvas
* What are the 7 layers of the OSI Model?

**(7) Application**

**(6) Presentation**

**(5) Session**

**(4) Transport**

**(3) Network**

**(2) Data Link**

**(1) Physical**

* In your own words describe why there are multiple layers rather than a single protocol for connection to another computer

1. **This allows for faster communication and ensures packets remain small. If there was a single layer, packets would need to be created which are far more robust in order to ensure that they are transmitted to the correct destination.**
2. **Security. Packets sent within the same subnet most likely do not need all of the security protocols which would be required if sending to different subnets/networks. Separating the layers means that packets can be tailored for the sender and destination. There is no unnecessary information being sent via local connections.**

* Which layer of the OSI Model is the IPV4 protocol located in?

**(3) Network**

* ICMP messages are contained within IPV4 packets. What statement do we use to create ICMP Messages?

**Ping or Echo**

**Part II - TCP/UDP Sniffer (simple version of Wireshark tool)**

* From Kali linux in NAT. Go to the terminal. As in previous labs, **su** , then **ifdown eth0**
* **ifup eth0**
* verify with ifconfig (you should have an ip address different than 192.168.10.2)
* Implement and test sniffer\_ip\_header\_decode.py as noted on page 41 and 42.
* Again, as always, you could get the code from the website, but instead you are asked to type in the code. Note that this is set to use Python 2.7, not Python 3.6. Create these items using Kali Linux
* From the kali terminal python will get you to 2.7, python3 will give you 3.6
* Add your name at the top of the code/output, run it and take a **screen shot**. Upload that screen shot and your **sniffer\_ip\_header\_decode.py** to Canvas
* Describe in your own words what sniffer\_ip\_header\_decode.py does (You will have to start reading from the beginning of the chapter)

**It is discovering hosts by sending a UDP datagram and waiting for ICMP responses. If it receives a response, then there is an active host. It then interprets the ICMP messages.**

* Read the rest of the chapter describing scanner.py and describe in your own words how /why Chapter 3 progressed from sniffer.py to sniffer\_ip\_header\_decode.py to sniffer with icmp.py to scanner.py

**Sniffer.y simply reads-in a single packet from a host (does not interpret)**

**Sniffer\_ip\_header\_decode reads-in multiple packets and decodes the ip header**

**With\_icmp.py decodes icmp packets which allows us to know the type and code which are being sent**

**Scanner.py is a host discovery tool for the entire subnet.**

**At a high level, we began by seeing if we could capture packets coming from a host that we are assessing. Then we began to interpret the packets coming from the host. Finally, we transitioned finding all of the hosts that are active on a subnet.**

* In Lab 02 you build a port scanner by opening up a raw socket, if it was listening on the other end, we assumed it was up. Now we progressed to make our own ICMP protocol. Look up ICMP protocol, and note three types of information statuses you can get from sending and receiving ICMP protocols?

**0 – echo reply**

**3 – destination unreachable**

**14 – timestamp reply**

**Part III - Using and setting up third party libaries**

* This section will show you how to load scapy; give a brief demonstration of usage and show you how you might package a third party library
* Scapy is a package that makes it very easy to send and receive all types of packet information, especially TCP/IP traffic. It also allows you to visualize the packets in a nice set of graphical libraries. It has functions built in for sending and receiving all of the major types of traffic, tracing packet hops on routers as well as creating "raw data" packets of your own design.
* pip is python package input. It is a script that gets packages from the internet, unzips them, and places them in your distribution directory. Packages are validated then added to the PIP library. These packages require a specially formatted setup.py file which defines dependencies. This pytsetup.py can also be used to package 3rd party dependencies without uploading them to the big PIP library in the sky.
* **su**
* **apt-get update**
* This updates the index of packages
* **apt-get install python3-pip (then answer y)**
* Do not just use pip as that will run either 2 or 3 depending on the last one installed. Remember that python 2.7 and 3.6+ require different code, and therefore different packages, and package managers. You can find packages at <https://pypi.python.org/pypi> They have a poor search engine so you have to kind of know what you are looking for
* **pip2 install scapy**
* **pip3 install scapy-python3**
* Find where it installed the package with the show parameter
* **pip2 show scapy**
* **pip3 show scapy-python3**
* Let's test out scapy. We will use a very small file, so you shouldn't need to use Pycharm, but can if you wish
* You should be in root to run these….most of these libraries will work in part with a non-privileged account, but why take the chance for lab!
* **Comment and implement this code using python3 as sniffer.py**

from scapy.all import \*

import sys

numpackets=int(sys.argv[1])

print ('Your Name, packet sniffer')

print('TCP Header')

ls(TCP)

print('ICMP Header')

ls(ICMP)

print('\nGo to a site in your browser')

packets=sniff(count=numpackets)

print('\nHere are the first {} packets'.format(numpackets))

packets.show()

print('\nhexdump')

for packet in range(0,numpackets-1):

hexdump(packets[packet])

ans,unans=sr(IP(dst="www.python.org", ttl=5)/ICMP())

print(ans)

print(unans)

* **python3 sniffer.py 50**
* in your firefox browser go to <ftp://ftp.gnu.org>, then click on the README
* Notice how you can see the packet information that defines the struct for most protocols, then you can sniff packets, and print out a summary and even a hexdump. You can also read to and from .pcap files, so you can make your own wireshark with ease. These are just a few of the things that scapy can do for you. If you are working with sending or receiving packets of any type for your project, you may want to look more into scapy.
* Once you see it working send the information to a file
* **python3 sniffer.py 50 > sniffer.txt**

**Part IV - Packages with \_\_init\_\_.py and setup.py**

* So you love scapy and want to include it into your final project. But we, like most users or employers expect a fully setup environment meeting all dependencies built into a single package
* For this portion, instead of just copying what you see on the page, understand what we are trying to do….the python version will change and therefore the directories will change. Instead of blindly copying, try to figure out what we are trying to do.
* When you import a package it looks in the local directory, then (using 3.5) it is looking in your /usr/local/lib/python3.5/dist-packages. This location is likely in some configuration file. If, however you are trying to build a package with all of the dependencies (like you have to for your final project)
* **cd /usr/local/lib/python3.5/dist-packages** (NOTE: You will likely have a different version of python ….change to that directory)
* So now we are hoping that the scapy-python3 folder has all of the things we need to use the library. It should have everything it needs in the folder if it is a properly formatted pip package. (There are some graphical functions that require additional packages, but most of the major functions are complete) If you cd into the scapy folder you will see a \_\_.init\_\_.py. Additionally, you will see an \_\_init\_\_.py in each subdirectory. When python sees the \_\_init\_\_.py it recognizes the directory (or subdirectory) as a "package". Per the documentation "In the simplest case, \_\_init\_\_.py can just be an empty file, but it can also execute initialization code for the package or set the \_\_all\_\_ variable, described later."
* Now we want to copy that entire package (again, copy from the right python directory)
* **cp -R /usr/local/lib/python3.5/dist-packages/scapy /root/Documents/scapytest/**
* **cd /root/Documents/scapytest**
* **ls** (you should see the scapy file folder…cd into that folder to verify that it has the \_\_init\_\_.py file as in the other directory)
* Now, we want to remove the originally installed scapy package to show you how the import works without the scapy package in that directory
* **cd /usr/local/lib/python3.5/dist-packages**
* **rm -R scapy\***
* **python3**
* >>>**from scapy.all import \***
* You should get a package not found error
* >>>**quit()**
* **cd /root/Documents/scapytest**
* **pwd** (you should be in root/Documents/scapytest) When you ls there should be a scapy folder
* **nano setup.py** (or vi setup.py)

**from distutils.core import setup**

**setup(name='scapy',**

**version='yourlastname',**

**packages=['scapy'],**

**)**

* python3 setup.py install
* **cd /usr/local/lib/python3.5/dist-packages**
* **Note that scapy and some scapy configuration files are now installed into dist-packages**
* **python3**
* >>>**from scapy.all import \***
* You should get a scapy message and all is good.
* There are tons of things that you would want to consider with multi-directory packages., including adding \_\_init\_\_.py files and parameters to use for the setup. These can be found at:
* [**https://docs.python.org/3/distutils/examples.html**](https://docs.python.org/3/distutils/examples.html)
* You don't need to hand in any deliverables for this section (Part IV), but you should understand how to make the pakages.