

CSC 432-Z1

Lab – Week 5

Professor: Ronny Bull

April 10, 2022

Joseph (JT) Cavallaro

Abstract

Scapy is a library for Python that allows programmers to create, send, sniff, and dissect network packets. It is able to create tools with the power to scan, analyze, probe, and/or attack networks.

“In other words, Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. Scapy can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks, or network discovery” (Scapy, 2022).

We will use the Interactive Tutorial in the docs, first within the Ubuntu VM terminal, then within the CORE network topology emulation we will continue the tutorial and review the results for both. Then we will create two Python scripts using commands from the tutorial and again review the results.

Introduction

In this lab, we will be using this Python library, Scapy, to customize the terminal, build a packet, stack the layers, read PCAP files, create a graphical dump, generate sets of packets, send packets, fuzz packets, inject bytes, send and receive packets, perform syn scans, and perform a tcp traceroute. To do this, we will run through the Scapy interactive tutorial on the docs and review the results. Then we will create a network topology using CORE with three nodes (PCs) and a Hub. Once this is created and configured, we will continue the tutorial from the Scapy docs, and review the results. Once we are done, we will create two Python scripts using commands from the Scapy tutorial. We will run these within the topology and use TCPDump to capture the data so we can review the results.

Python Module Scapy, Scapy, & Dependencies Install

In this section, we will be installing the Python module Scapy, a library that allows programmers to create, send, sniff, and dissect network packets. Along with this, we will also be installing Scapy, the source code itself, then installing all of the dependencies needed for Scapy to work properly.

Python Scapy

First, we need to install the Python Scapy module. Scapy comes in three different bundles, only Scapy, Scapy Basic, and Scapy Complete. Scapy Basic installs Scapy and IPython while the complete bundle installs Scapy and all of the main dependencies. To install Python Scapy, I used the command “sudo pip install --pre scapy[complete]” which was already installed on my VM (Figure 1).

```
j@i:~$ sudo pip install --pre scapy[complete]
Requirement already satisfied: scapy[complete] in /usr/local/lib/python3.8/dist-packages (2.4.5)
Requirement already satisfied: ipython; extra == "complete" in /usr/local/lib/python3.8/dist-packages (from scapy[complete]) (7.29.0)
Requirement already satisfied: pyx; extra == "complete" in /usr/local/lib/python3.8/dist-packages (from scapy[complete]) (0.15)
Requirement already satisfied: cryptography>=2.0; extra == "complete" in /usr/lib/python3/dist-packages (from scapy[complete]) (2.8)
```

Figure 1 - Python Scapy Library Install

Scapy

Now that Python Scapy is installed, we can install the Scapy source code. We can access this at the Github page and the command “sudo git clone https://github.com/secdev/scapy.git” (Figure 2). Then, after changing directories to the newly created scapy directory and built and installed Scapy (Figure 2 & 3). To do this, I used the commands “cd scapy” and “sudo python3 setup.py build” and “sudo python3 setup.py build install” (Figure 2).

```
j@i:~$ sudo git clone https://github.com/secdev/scapy.git
Cloning into 'scapy'...
remote: Enumerating objects: 35790, done.
remote: Counting objects: 100% (2346/2346), done.
remote: Compressing objects: 100% (1147/1147), done.
remote: Total 35790 (delta 1553), reused 1774 (delta 1192), pack-reused 33444
Receiving objects: 100% (35790/35790), 85.28 MiB / 9.94 MiB/s, done.
Resolving deltas: 100% (23831/23831), done.
j@i:~$ cd scapy
j@i:~/scapy$ sudo python3 setup.py build
running build
running build_py
creating build
creating build/lib
creating build/lib/test
copying test/__init__.py -> build/lib/test
copying test/testsocket.py -> build/lib/test
creating build/lib/scapy
copying scapy/sessions.py -> build/lib/scapy
copying scapy/config.py -> build/lib/scapy
```

Figure 2 - Scapy Install - Steps 1 - 3

```
j@i:~/scapy$ sudo python3 setup.py build install
running build
running build_py
copying scapy/VERSION -> build/lib/scapy
running install
running bdist_egg
running egg_info
Installed /usr/local/lib/python3.8/dist-packages/scapy-2.4.5rc1.dev228-py3.8.egg
Processing dependencies for scapy==2.4.5rc1.dev228
Finished processing dependencies for scapy==2.4.5rc1.dev228
j@i:~/scapy$
```

Figure 3 - Scapy Install - Step 4

Matplotlib

Now that the Scapy source code is built and installed, we can install the needed dependencies for Scapy to run properly. The first is Matplotlib, installed using the command “sudo pip install matplotlib” (Figure 4). This was already installed on my Ubuntu VM.

```
j@i:~/scapy$ sudo pip install matplotlib
Requirement already satisfied: matplotlib in /usr/local/lib/python3.8/dist-packages (3.5.0rc1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib)
(1.3.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/dist-packages (from matplotlib) (2
.1.2)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.8/dist-packages (from matplotlib)
(4.27.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/lib/python3/dist-packages (from matplotlib) (2.7.
3)
Requirement already satisfied: setuptools-scm>=4 in /usr/local/lib/python3.8/dist-packages (from matplotlib)
(6.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages (from matplotlib) (0.11
.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.8/dist-packages (from matplotlib) (1.21.
3)
Requirement already satisfied: pyparsing>=2.2.1 in /usr/lib/python3/dist-packages (from matplotlib) (2.4.6)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.8/dist-packages (from matplotlib) (8.3
.2)
Requirement already satisfied: setuptools in /usr/lib/python3/dist-packages (from setuptools-scm>=4->matplot
lib) (45.2.0)
Requirement already satisfied: tomli>=1.0.0 in /usr/local/lib/python3.8/dist-packages (from setuptools-scm>=4
->matplotlib) (1.2.2)
```

Figure 4 - Matplotlib Install

PyX, Graphviz & Python Graphviz

Now that Matplotlib is installed, we can install PyX, Graphviz and Python Graphviz.

First is PyX, which was installed by using the command “sudo pip install PyX” (Figure 5). Then I installed Graphviz by using the command “sudo apt-get install -y graphviz” (Figure 5). Finally, I installed the Python Graphviz module by using the command “sudo apt-get install python3-graphviz” (Figure 6).

```
j@i:~/scapy$ sudo pip install PyX
Requirement already satisfied: PyX in /usr/local/lib/python3.8/dist-packages (0.15)
j@i:~/scapy$ sudo apt-get install -y graphviz
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libnetplan0 python3-netifaces
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libann0 libcdt5 libcgraph6 libgts-0.7-5 libgts-bin libgv6 libgvpr2 liblab-gamut1 libpathplan4
Suggested packages:
  graphviz-doc
The following NEW packages will be installed:
  graphviz libann0 libcdt5 libcgraph6 libgts-0.7-5 libgts-bin libgv6 libgvpr2 liblab-gamut1 libpathplan4
0 upgraded, 10 newly installed, 0 to remove and 0 not upgraded.
Need to get 1,880 kB of archives.
```

Figure 5 - PyX & Graphviz Install

```
j@i:~/scapy$ sudo apt-get install -y python3-graphviz
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libnetplan0 python3-netifaces
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  python3-graphviz
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 23.1 kB of archives.
After this operation, 92.2 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu focal/universe amd64 python3-graphviz all 0.8.4-2 [23.1 kB]
Fetched 23.1 kB in 0s (110 kB/s)
Selecting previously unselected package python3-graphviz.
(Reading database ... 461565 files and directories currently installed.)
Preparing to unpack .../python3-graphviz_0.8.4-2_all.deb ...
Unpacking python3-graphviz (0.8.4-2) ...
Setting up python3-graphviz (0.8.4-2) ...
j@i:~/scapy$
```

Figure 6 - Python Graphviz Install

ImageMagick

Moving on, we can now install ImageMagick. The first step is to install the Ubuntu package version of ImageMagick. To do this, I used the “sudo apt install imagemagick” which was already installed (Figure 7).

```
j@i:~/scapy$ sudo apt install imagemagick
Reading package lists... Done
Building dependency tree
Reading state information... Done
imagemagick is already the newest version (8:6.9.10.23+dfsg-2.lubuntu11.4).
The following packages were automatically installed and are no longer required:
  libnetplan0 python3-netifaces
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
j@i:~/scapy$
```

Figure 7 - ImageMagick Install - Step 1

Now that the Ubuntu package of ImageMagick, we can install the source code of ImageMagick by using the command “sudo wget https://download.imagemagick.org/ImageMagick/download/ImageMagick.tar.gz” (Figure 8).

```
j@i:~$ sudo wget https://download.imagemagick.org/ImageMagick/download/ImageMagick.tar.gz
--2022-04-11 19:39:09-- https://download.imagemagick.org/ImageMagick/download/ImageMagick.tar.gz
Resolving download.imagemagick.org (download.imagemagick.org) ... 50.251.58.13
Connecting to download.imagemagick.org (download.imagemagick.org) |50.251.58.13|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 15092144 (14M) [application/x-gzip]
Saving to: 'ImageMagick.tar.gz'

ImageMagick.tar.gz          100%[=====] 14.39M  653KB/s   in 47s
2022-04-11 19:39:57 (313 KB/s) - 'ImageMagick.tar.gz' saved [15092144/15092144]

j@i:~$
```

Figure 8 - ImageMagick Install - Step 2

Now that we download the tar.gz file, we can use the ‘tar’ command to extract it. Then change directories to the newly created ImageMagick folder and configured the source code. After the configuration completed, I needed to make and install the ImageMagick source code. This allows us to set ImageMagick to be used within the home directory. Then, to verify that ImageMagick is installed, we can use the identify command from withing the ImageMagick directory as well from the home directory (Figure 10). Below are the command, listed in order, used to complete the steps above:

- “sudo tar -axvf ImageMagick.tar.gz”
- “cd ImageMagick-7.1.0-29/”
- “sudo ./configure”
- “sudo make install”
- “sudo ldconfig /usr/local/lib”
- “identify”
- “cd”
- “identify”

```
j@i:~$ sudo tar -axvf ImageMagick.tar.gz
j@i:~$ cd ImageMagick-7.1.0-29/
ImageMagick-7.1.0-29/
ImageMagick-7.1.0-29/coders/
ImageMagick-7.1.0-29/coders/aai.h
ImageMagick-7.1.0-29/coders/art.h
ImageMagick-7.1.0-29/coders/ashlar.h
ImageMagick-7.1.0-29/coders/avs.h
ImageMagick-7.1.0-29/coders/bgr.h
ImageMagick-7.1.0-29/coders/bmp.h
ImageMagick-7.1.0-29/coders/braille.h
j@i:~/ImageMagick-7.1.0-29$ sudo ./configure
j@i:~/ImageMagick-7.1.0-29$ checking build system type... x86_64-pc-linux-gnu
j@i:~/ImageMagick-7.1.0-29$ checking host system type... x86_64-pc-linux-gnu
j@i:~/ImageMagick-7.1.0-29$ checking target system type... x86_64-pc-linux-gnu
j@i:~/ImageMagick-7.1.0-29$ checking for a BSD-compatible install... /usr/bin/install -c
j@i:~/ImageMagick-7.1.0-29$ checking whether build environment is sane... yes
j@i:~/ImageMagick-7.1.0-29$ checking for a thread-safe mkdir -p... /bin/mkdir -p
j@i:~/ImageMagick-7.1.0-29$ checking for gawk... no
j@i:~/ImageMagick-7.1.0-29$ checking for mawk... mawk
j@i:~/ImageMagick-7.1.0-29$ make install
make install-am
make[1]: Entering directory '/home/j/ImageMagick-7.1.0-29'
  CC      utilities/magick.o
  CC      MagickCore/libMagickCore_7_Q16HDRI_la-accelerate.lo
  CC      MagickCore/libMagickCore_7_Q16HDRI_la-animate.lo
  CC      MagickCore/libMagickCore_7_Q16HDRI_la-annotate.lo
  CC      MagickCore/libMagickCore_7_Q16HDRI_la-artifact.lo
  CC      MagickCore/libMagickCore_7_Q16HDRI_la-attribute.lo
  CC      MagickCore/libMagickCore_7_Q16HDRI_la-blob.lo
  CC      MagickCore/libMagickCore_7_Q16HDRI_la-cache.lo
/usr/bin/install -c -m 644 MagickCore/ImageMagick.pc MagickCore/ImageMagick-7.Q16HDRI.pc MagickCore/MagickCore.pc MagickCore/MagickCore-7.Q16HDRI.pc MagickWand/MagickWand.pc MagickWand/MagickWand-7.Q16HDRI.pc Magick++/lib/Magick++.pc Magick++/lib/Magick++-7.Q16HDRI.pc '/usr/local/lib/pkgconfig'
make[2]: Leaving directory '/home/j/ImageMagick-7.1.0-29'
make[1]: Leaving directory '/home/j/ImageMagick-7.1.0-29'
j@i:~/ImageMagick-7.1.0-29$
```

Figure 9 - ImageMagick Install - Steps 3 - 6

```
j@i:~/ImageMagick-7.1.0-29$ sudo ldconfig /usr/local/lib
j@i:~/ImageMagick-7.1.0-29$ identify
Version: ImageMagick 7.1.0-29 Q16-HDRI x86_64 19841 https://imagemagick.org
Copyright: (C) 1999 ImageMagick Studio LLC
License: https://imagemagick.org/script/license.php
Features: Cipher DPC HDRI OpenMP(4.5)
Delegates (built-in): xml zlib
Compiler: gcc (9.4)
Usage: identify [options ...] file [*options ZIP file ...]
Image Settings:
  -alpha option      on, activate, off, deactivate, set, opaque, copy
  -transparent       transparent, extract, background, or shape
  -antialias        remove pixel-aliasing
  -authenticate password
```



```
j@i:~/ImageMagick-7.1.0-29$ cd
j@i:~$ identify
Version: ImageMagick 7.1.0-29 Q16-HDRI x86_64 19841 https://imagemagick.org
Copyright: (C) 1999 ImageMagick Studio LLC
License: https://imagemagick.org/script/license.php
Features: Cipher DPC HDRI OpenMP(4.5)
Delegates (built-in): xml zlib
Compiler: gcc (9.4)
Usage: identify [options ...] file [*options ZIP file ...]
```

Figure 10 - ImageMagick Install - Steps 7 - 10

vPython, pyzmq, libzmq3-dev & Python-jinja2

Now that ImageMagick is installed, we can install vPython, pyzmq, libzmq3-dev and Python-jinja2. First we will install vPython by using the command “`pip install vpython`” (Figure 11). I noticed that `pyzmq` and `jinja2` were out of date (red text in Figure 11), so I updated them.

```
j@i:~$ pip install vpython
Collecting vpython
  Downloading vpython-7.6.3-cp38-cp38-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2
010_x86_64.whl (3.7 MB)
|██████████| 3.7 MB 2.0 MB/s
Collecting jupyter-server-proxy
  Downloading jupyter_server_proxy-3.2.1-py3-none-any.whl (35 kB)
Collecting ipykernel
  Downloading ipykernel-6.13.0-py3-none-any.whl (131 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from vpython) (1.21.3)
Collecting autobahn>=18.8.2
  Downloading autobahn-22.3.2.tar.gz (376 kB)
Collecting jupyter
  Downloading jupyter-1.0.0-py2.py3-none-any.whl (2.7 kB)
Collecting jupyter-server>=1.0
  Downloading jupyter_server-1.16.0-py3-none-any.whl (343 kB)
Collecting importlib-resources>=1.4.0; python_version < "3.9"
  Downloading importlib_resources-5.6.0-py3-none-any.whl (28 kB)
Collecting pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0
  Downloading pyrsistent-0.18.1-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (119 kB)
Collecting zipp>=3.1.0; python_version <="3.10"
  Downloading zipp-3.8.0-py3-none-any.whl (5.4 kB)
Building wheels for collected packages: autobahn
  Building wheel for autobahn (setup.py): ... done
    Created wheel for autobahn: filename=autobahn-22.3.2-py2.py3-none-any.whl size=521515 sha256=2fa30d15cdc02c
ab6c8d3b1b6cf30989a8b563ec1819478465e2e0d4c5def674
    Stored in directory: /home/j/.cache/pip/wheels/1b/5e/16/6dbc0bff2674dae009933d1efc4b3487001eaacd05811f5513
Successfully built autobahn
ERROR: jupyter-client 7.2.2 has requirement pyzmq>=22.3, but you'll have pyzmq 18.1.1 which is incompatible.
ERROR: nbconvert 6.5.0 has requirement jinja2>=3.0, but you'll have jinja2 2.10.1 which is incompatible.
```

Figure 11 - vPython Install

To first install pyzmq, I used the following three commands: “sudo pip install pyzmq” and “pip install --no-binary=:all: pyzmq” and “sudo apt-get install libzmq3-dev” (Figure 12).

```
j@i:~$ sudo pip install pyzmq
Requirement already satisfied: pyzmq in /usr/lib/python3/dist-packages (18.1.1)
j@i:~$ pip install --no-binary=:all: pyzmq
/usr/lib/python3/dist-packages/secretstorage/dhcrypto.py:15: CryptographyDeprecationWarning: int_from_bytes is deprecated, use int.from_bytes instead
  from cryptography.utils import int_from_bytes
/usr/lib/python3/dist-packages/secretstorage/util.py:19: CryptographyDeprecationWarning: int_from_bytes is deprecated, use int.from_bytes instead
  from cryptography.utils import int_from_bytes
Requirement already satisfied: pyzmq in /usr/lib/python3/dist-packages (18.1.1)
j@i:~$ sudo apt-get install libzmq3-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
libzmq3-dev is already the newest version (4.3.2-2ubuntu1).
libzmq3-dev set to manually installed.
The following packages were automatically installed and are no longer required:
  libnetplan0 python3-netifaces
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

Figure 12 - pyzmq & libzmq3-dev Install

Now we can install the jinja Python module. To do this, I used the command “sudo apt-get install -y python-jinja2” (Figure 13).

```
j@i:~$ sudo apt-get install -y python-jinja2
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libnetplan0 python3-netifaces
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  python-markupsafe
Suggested packages:
  python-jinja2-doc
The following NEW packages will be installed:
  python-jinja2 python-markupsafe
```

Figure 13 - Python-jinja2 Install

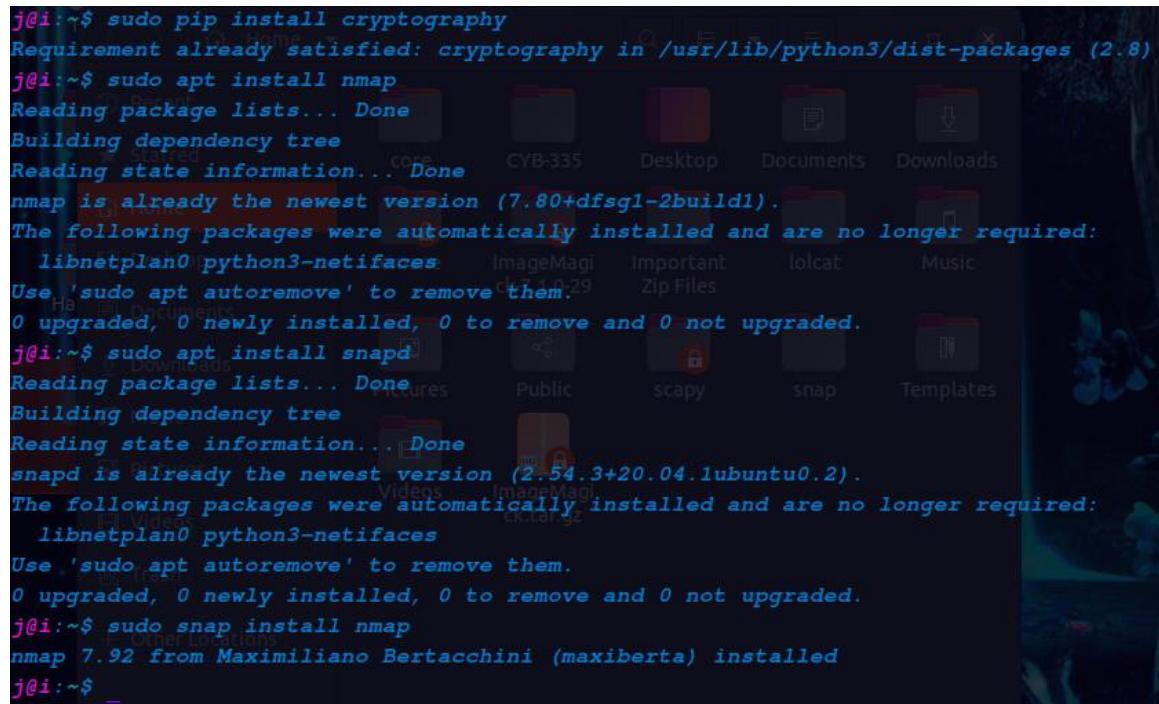
Now that these other dependencies for vPython are installed, I re-installed this by running the same command “pip install vpython” (Figure 14). This installed pyzmq and allowed vPython to successfully install (Figure 14).

```
j@i:~$ pip install vpython
/usr/lib/python3/dist-packages/secretstorage/dhcrypto.py:15: CryptographyDeprecationWarning: int_from_bytes is deprecated, use int.from_bytes instead
  from cryptography.utils import int_from_bytes
/usr/lib/python3/dist-packages/secretstorage/util.py:19: CryptographyDeprecationWarning: int_from_bytes is deprecated, use int.from_bytes instead
  from cryptography.utils import int_from_bytes
Requirement already satisfied: vpython in ./local/lib/python3.8/site-packages (7.6.3)
Requirement already satisfied: ipykernel in ./local/lib/python3.8/site-packages (from vpython) (6.13.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from vpython) (1.21.3)
Requirement already satisfied: autobahn>=18.8.2 in ./local/lib/python3.8/site-packages (from vpython) (22.3)
Collecting pyzmq==22.3
  Downloading pyzmq-22.3.0-cp38-cp38-manylinux_2_12_x86_64_manylinux2010_x86_64.whl (1.1 MB)
    |████████| 1.1 MB 2.6 MB/s
Requirement already satisfied: entrypoints in /usr/lib/python3/dist-packages (from jupyter-client>=6.1.12->ipykernel->vpython) (0.3)
Installing collected packages: pyzmq
Successfully installed pyzmq-22.3.0
j@i:~$
```

Figure 14 - Successful vPython Install

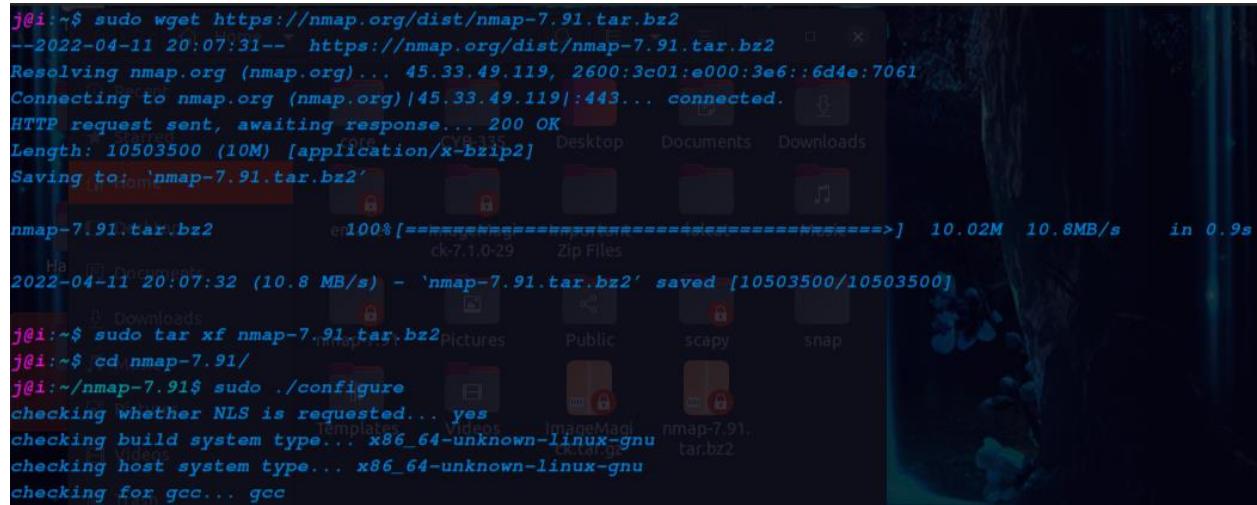
Cryptography, Nmap, OpenSSL, LibSSH2, Linssh2-1-dev & SoX

Now we can install Cryptography, Nmap, OpenSSL, LibSSH2, Linssh2-1-dev & SoX. These are more dependencies of Scapy. First I installed Cryptography using the command “sudo pip install cryptography” (Figure 15). I then needed to install snap and Nmap. To do this, I used the following commands: “sudo apt install nmap” & “sudo apt install snapd” & sudo snap install nmap” (Figure 15). Next I needed to install the source code for Nmap. To do this, I used the following commands: “sudo wget https://nmap.org/dist/nmap-7.91.tar.bz2” & “sudo tar xf nmap-7.91.tar.bz2” & “cd nmap-7.91/” & “sudo ./configure” (Figure 16).



```
j@i:~$ sudo pip install cryptography
Requirement already satisfied: cryptography in /usr/lib/python3/dist-packages (2.8)
j@i:~$ sudo apt install nmap
Reading package lists... Done
Building dependency tree
Reading state information... Done
nmap is already the newest version (7.80+dfsg1-2build1).
The following packages were automatically installed and are no longer required:
  libnetplan0 python3-netifaces
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
j@i:~$ sudo apt install snapd
Reading package lists... Done
Building dependency tree
Reading state information... Done
snapd is already the newest version (2.54.3+20.04.lubuntu0.2).
The following packages were automatically installed and are no longer required:
  libnetplan0 python3-netifaces
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
j@i:~$ sudo snap install nmap
nmap 7.92 from Maximiliano Bertacchini (maxiberta) installed
j@i:~$
```

Figure 15 - Cryptography & Basic NMap Install



```
j@i:~$ sudo wget https://nmap.org/dist/nmap-7.91.tar.bz2
--2022-04-11 20:07:31-- https://nmap.org/dist/nmap-7.91.tar.bz2
Resolving nmap.org (nmap.org)... 45.33.49.119, 2600:3c01:e000:3e6::6d4e:7061
Connecting to nmap.org (nmap.org)|45.33.49.119|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10503500 (10M) [application/x-bzip2]
Saving to: 'nmap-7.91.tar.bz2'

nmap-7.91.tar.bz2      100%[=====] 10.02M 10.8MB/s    in 0.9s
2022-04-11 20:07:32 (10.8 MB/s) - 'nmap-7.91.tar.bz2' saved [10503500/10503500]

j@i:~$ sudo tar xf nmap-7.91.tar.bz2
j@i:~$ cd nmap-7.91/
j@i:~/nmap-7.91$ sudo ./configure
checking whether NLS is requested... yes
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking for gcc... gcc
```

Figure 16 - NMap Install - Steps 1 - 4

After this was done, I noticed that two warnings appeared. They stated, “You are compiling without OpenSSL and LibSSH2” (Figure 17). So, I decided to install both of these.

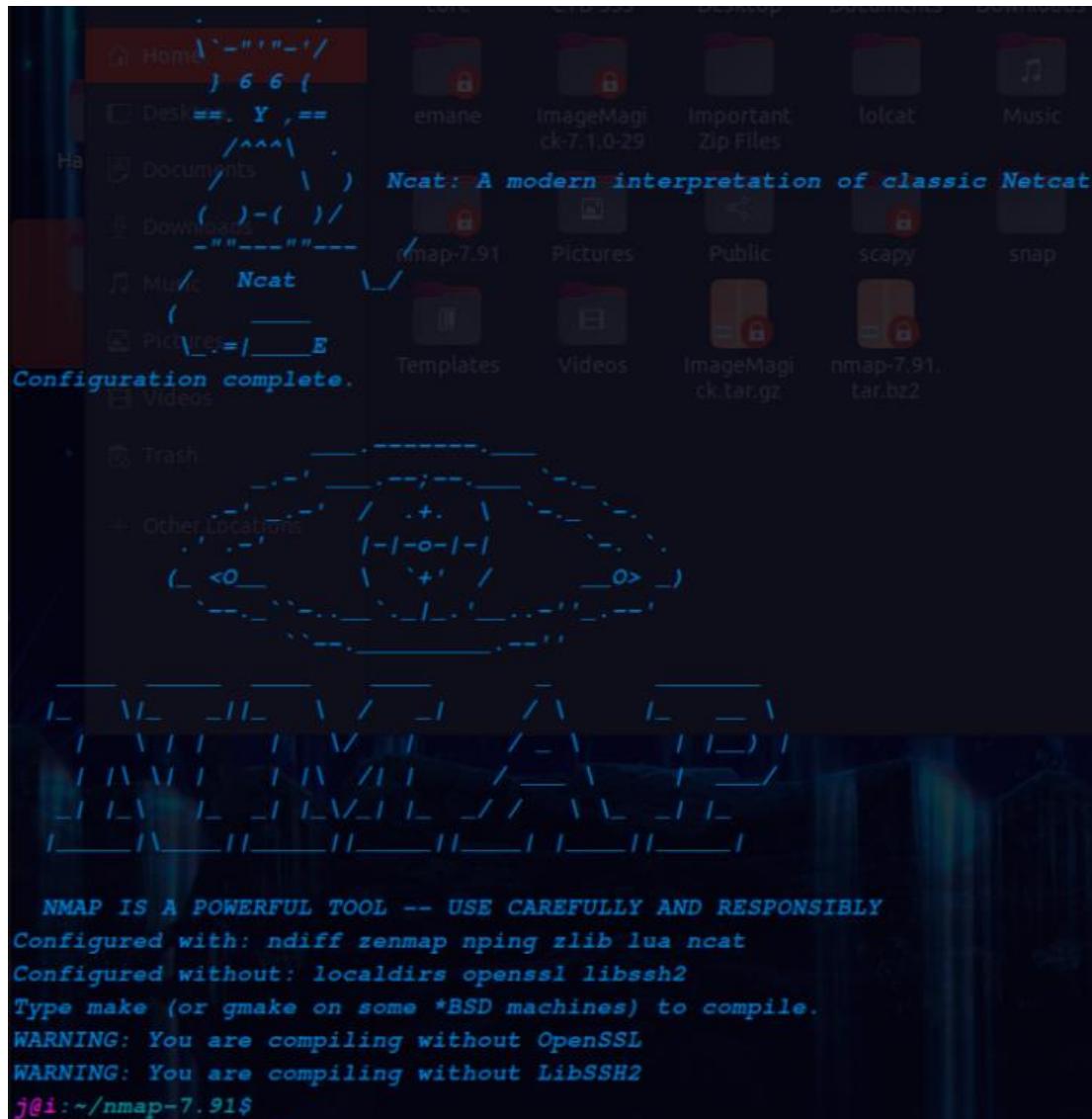
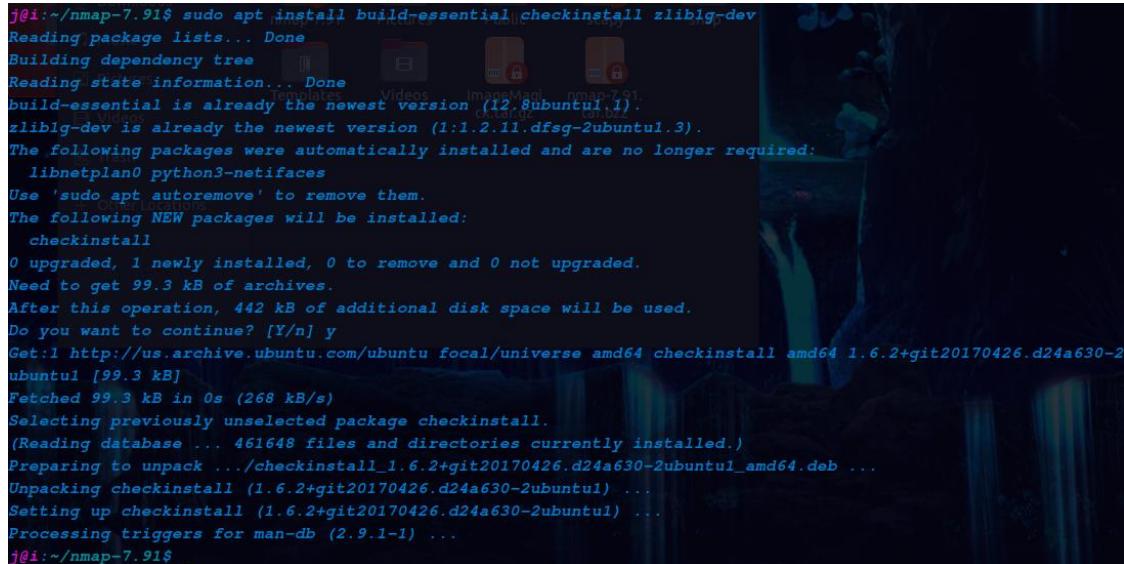


Figure 17 - NMap Successful Configuration

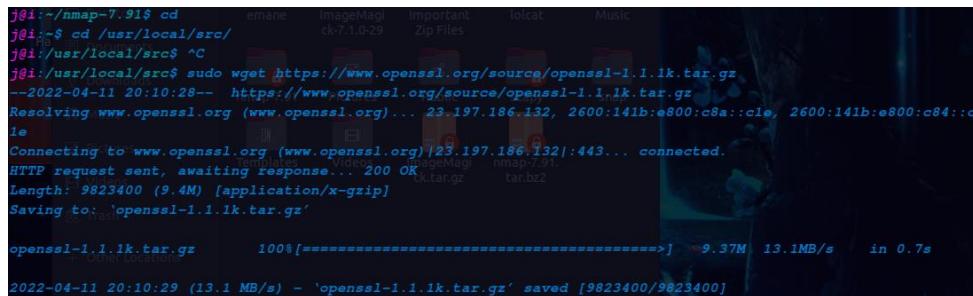
I first installed the Ubuntu packages “build-essential, checkinstall, and zlib1g-dev” (Figure 18). To do this, I used the following command: “sudo apt install build-essential checkinstall zlib1g-dev” (Figure 18). I then changed to the directory “/usr/local/src/” (Figure 19) and installed the source code for OpenSSL. To do this, I used the following commands:

- “sudo apt install build-essential checkinstall zlib1g-dev”
- “cd”
- “cd /usr/local/src”
- “sudo wget https://www.openssl.org/source/openssl-1.1.1k.tar.gz”



```
j@i:~/nmap-7.91$ sudo apt install build-essential checkinstall zlib1g-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libnetplan0 python3-netifaces
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  checkinstall
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 99.3 kB of archives.
After this operation, 442 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us.archive.ubuntu.com/ubuntu focal/universe amd64 checkinstall amd64 1.6.2+git20170426.d24a630-2
[99.3 kB]
Fetched 99.3 kB in 0s (268 kB/s)
Selecting previously unselected package checkinstall.
(Reading database ... 461648 files and directories currently installed.)
Preparing to unpack .../checkinstall_1.6.2+git20170426.d24a630-2ubuntul_amd64.deb ...
Unpacking checkinstall (1.6.2+git20170426.d24a630-2ubuntul) ...
Setting up checkinstall (1.6.2+git20170426.d24a630-2ubuntul) ...
Processing triggers for man-db (2.9.1-1) ...
j@i:~/nmap-7.91$
```

Figure 18 - NMap Install - Step 5



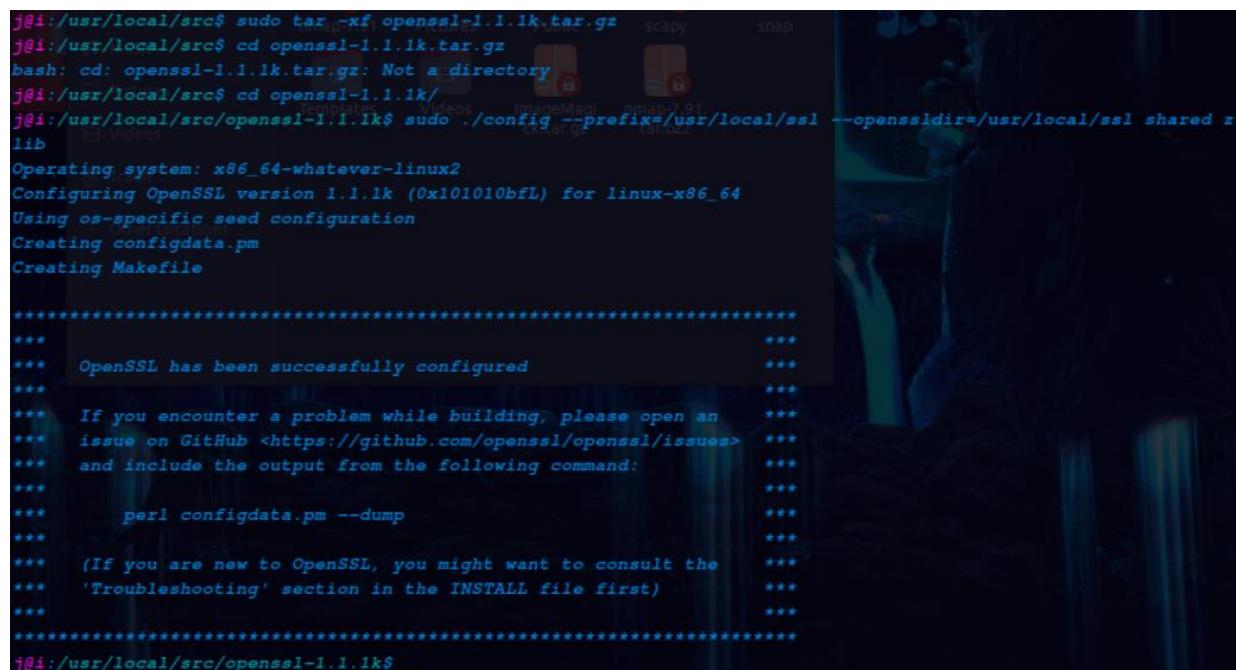
```
j@i:~/nmap-7.91$ cd
j@i:~$ cd /usr/local/src/
j@i:~/usr/local/src$ ^C
j@i:~/usr/local/src$ sudo wget https://www.openssl.org/source/openssl-1.1.1k.tar.gz
--2022-04-11 20:10:28-- https://www.openssl.org/source/openssl-1.1.1k.tar.gz
Resolving www.openssl.org (www.openssl.org)... 23.197.186.132, 2600:141b:e800:c8a::c1e, 2600:141b:e800:c84::c1e
Connecting to www.openssl.org (www.openssl.org)|23.197.186.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 9823400 (9.4M) [application/x-gzip]
Saving to: 'openssl-1.1.1k.tar.gz'

openssl-1.1.1k.tar.gz      100%[=====]  9.37M  13.1MB/s   in 0.7s

2022-04-11 20:10:29 (13.1 MB/s) - 'openssl-1.1.1k.tar.gz' saved [9823400/9823400]
```

Figure 19 - OpenSSL Install - Steps 1 – 3

- “sudo tar -xf openssl-1.1.1k.tar.gz”
- “cd openssl-1.1.1k.tar.gz”
- “cd openssl-1.1.1k/”
- “sudo ./config --prefix=/usr/local/ssl --openssldir=/usr/local/ssl shared zlib”
- “sudo make && sudo make build && sudo make install”

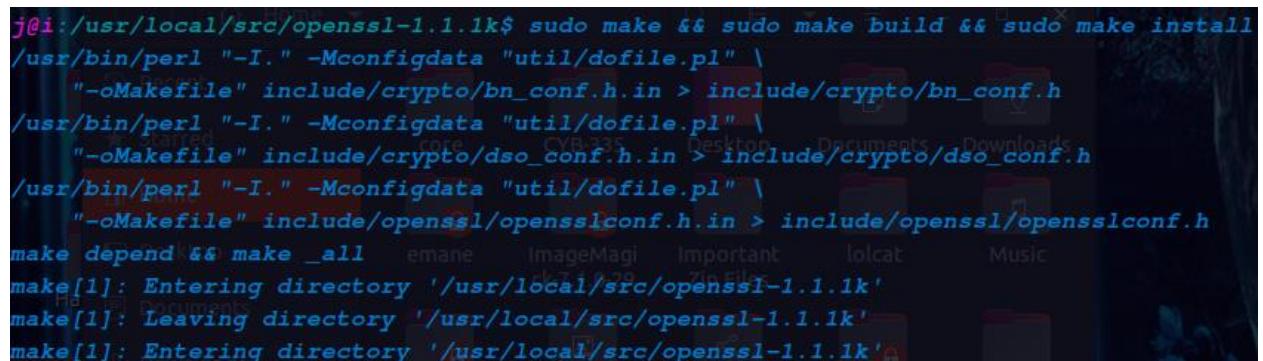


```
j@i:/usr/local/src$ sudo tar -xf openssl-1.1.1k.tar.gz      scapy      snap
j@i:/usr/local/src$ cd openssl-1.1.1k.tar.gz
bash: cd: openssl-1.1.1k.tar.gz: Not a directory
j@i:/usr/local/src$ cd openssl-1.1.1k/
j@i:/usr/local/src/openssl-1.1.1k$ sudo ./config --prefix=/usr/local/ssl --openssldir=/usr/local/ssl shared zlib
Operating system: x86_64-whatever-linux2
Configuring OpenSSL version 1.1.1k (0x101010bfL) for linux-x86_64
Using os-specific seed configuration
Creating configdata.pm
Creating Makefile

*****
*** OpenSSL has been successfully configured ***
*** If you encounter a problem while building, please open an issue on GitHub <https://github.com/openssl/openssl/issues>
*** and include the output from the following command:
*** perl configdata.pm --dump
*** (If you are new to OpenSSL, you might want to consult the 'Troubleshooting' section in the INSTALL file first)
*** *****

j@i:/usr/local/src/openssl-1.1.1k$
```

Figure 20 - OpenSSL Install - Steps 4 - 7



```
j@i:/usr/local/src/openssl-1.1.1k$ sudo make && sudo make build && sudo make install
/usr/bin/perl "-I." -Mconfigdata "util/dofile.pl" \
    "-oMakefile" include/crypto/bn_conf.h.in > include/crypto/bn_conf.h
/usr/bin/perl "-I." -Mconfigdata "util/dofile.pl" \
    "-oMakefile" include/crypto/dso_conf.h.in > include/crypto/dso_conf.h
/usr/bin/perl "-I." -Mconfigdata "util/dofile.pl" \
    "-oMakefile" include/openssl/opensslconf.h.in > include/openssl/opensslconf.h
make depend && make _all  emanem  ImageMagick  Important  lolcat  Music
make[1]: Entering directory '/usr/local/src/openssl-1.1.1k'
make[1]: Leaving directory '/usr/local/src/openssl-1.1.1k'
make[1]: Entering directory '/usr/local/src/openssl-1.1.1k'
```

Figure 21 - OpenSSL Install - Step 8

The next package to install was LibSSH2, which can be installed using the command “sudo apt install libssh2-1-dev” (Figure 22). After installing these two packages, I changed back to the directory ‘nmap-7.91’ and used the command “sudo make” (Figure 23).

```
j@i:~/nmap-7.91$ sudo apt install libssh2-1-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libnetplan0 python3-netifaces
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libgcrypt20-dev libgpg-error-dev libssh2-1
Suggested packages:
  libgcrypt20-doc
The following NEW packages will be installed:
  libgcrypt20-dev libgpg-error-dev libssh2-1 libssh2-1-dev
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
```

Figure 22 - Libssh2-1-dev Install

```
j@i:~/nmap-7.91$ sudo make
g++ -MM -I./liblinear -I./liblbus -I./libdnet-stripped/include -I./nbase -I./nsock/include -DHAVE_CONFIG_H -D
NMAP_PLATFORM=\"x86_64-unknown-linux-gnu\" -DNMAPDATAIR=\"/usr/local/share/nmap\" -D_FORTIFY_SOURCE=2 charpo
ol.cc FingerPrintResults.cc FFEngine.cc FPMModel.cc idle_scan.cc MACLookup.cc main.cc nmap.cc nmap_dns.cc nmap_
_error.cc nmap_ftp.cc NmapOps.cc NmapOutputTable.cc nmap_tty.cc osscan2.cc osscan.cc output.cc payload.cc por
tlist.cc portreasons.cc protocols.cc scan_engine.cc scan_engine_connect.cc scan_engine_raw.cc scan_lists.cc s
ervice_scan.cc services.cc string_pool.cc Target.cc NewTargets.cc TargetGroup.cc targets.cc tcpip.cc timing.c
c traceroute.cc utils.cc xml.cc nse_main.cc nse_utility.cc nse_nsock.cc nse_dnet.cc nse_fs.cc nse_nmaplib.cc
nse_debug.cc nse_pcrelib.cc nse_lpeg.cc nse_zlib.cc > makefile.dep
Compiling libnetutil
cd libnetutil && make
```

Figure 23 - NMap Install - Step 6

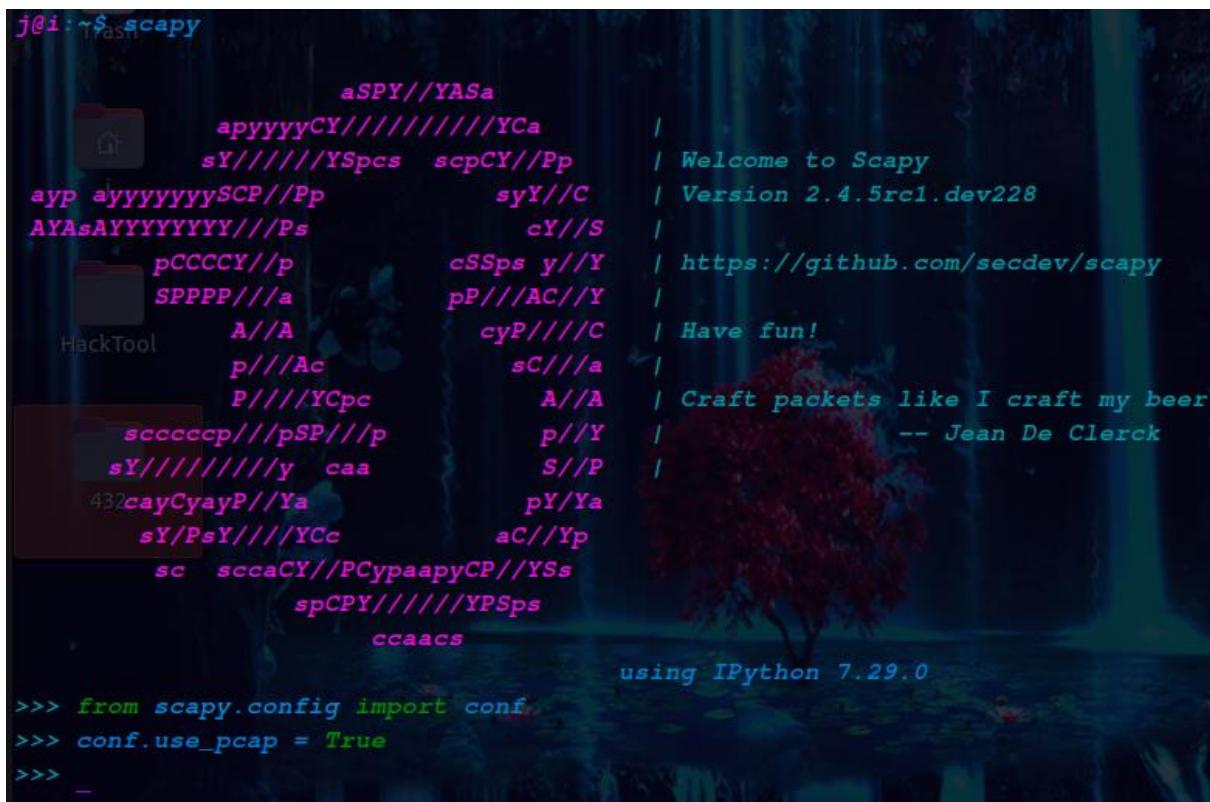
After this ran successfully, Nmap, OpenSSL, and LibSSH2 were installed successfully. So, the last dependency that is needed to install scapy is SoX. This can be installed using the command “sudo apt install sox” (Figure 24).

```
j@i:~/nmap-7.91$ sudo apt install sox
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libnetplan0 python3-netifaces
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libopencore-amrnb0 libopencore-amrwb0 libsox-fmt-alsa libsox-fmt-base libsox3
Suggested packages:
  libsox-fmt-all
The following NEW packages will be installed:
  libopencore-amrnb0 libopencore-amrwb0 libsox-fmt-alsa libsox-fmt-base libsox3 sox
0 upgraded, 6 newly installed, 0 to remove and 0 not upgraded.
Need to get 513 kB of archives.
After this operation, 1,564 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

Figure 24 - SoX Install

Scapy Install Verification

Now that Scapy, and all of its required dependencies are installed, as well successfully, we can now run the command “scapy” within the terminal to start the program. After starting scapy, I configured Scapy to read pcap files by running the two commands: “from scapy.config import conf” & “conf.use_pcap = True” (Figure 25). Now that we have verified that Scapy is installed, we can start testing scapy out.



```
j@i:~$ scapy
  aSPY//YAsa
  apyyyyCY//////////YCa
  sY////////YSpCs  scpCY//Pp  | Welcome to Scapy
  ayp  ayyyyyyyySCP//Pp      syY//C  | Version 2.4.5rc1.dev228
  AYAsAYYYYYYYYY//Ps          cY//S  |
  pCCCCY//P                cSSPs y//Y  | https://github.com/secdev/scapy
  SPPPP//a                  pP///AC//Y  |
  A//A                      cyP///C  | Have fun!
  p///Ac                   sC///a  |
  P///YCpc                 A//A  | Craft packets like I craft my beer.
  scccccp///pSP///p          p//Y  | -- Jean De Clerck
  sY/////////y   caa          S//P  |
  432cayCyayP//Ya           pY/Ya
  sY/PsY////YCc             aC//Yp
  sc  sccaCY//PCypaapyCP//YSs
  spCPY//////YPSPs
  ccaacs
                                         using IPython 7.29.0
>>> from scapy.config import conf
>>> conf.use_pcap = True
>>> _
```

Figure 25 - Scapy Install Verification & Configuring Super-Sockets

Scapy Test

In this section, we will be following the interactive tutorial on the Scapy Docs. This is the ‘Usage’ section of the Scapy Documentation. It teaches people how to get use to Scapy’s interactive shell, as well commands used by Scapy.

Customizing Terminal, Building Packet & Stacking Layers –

Before using the Scapy interface, they have different theme options that change the color of the interface. These consist of the “DefaultTheme, BrightTheme, RastaTheme, ColorOnBlackTheme, BlackAndWhite, HTMLTheme, LatexTheme” (Scapy, 2022). In this case, I choose the Rasta Theme as it was the easiest to view (Figure 26). After customizing the terminal theme, we can start the first steps of using scapy, which is to build a packet, and play around with it.

After completing the first steps, we can now start the next section, stacking layers (Scapy, 2022). Using the ‘/’ operator represents the “composition operator between two layers” (Scapy, 2022). When using this operator, “the lower layer can have one or more of its defaults fields overloaded according to the upper layer” (Scapy, 2022). Any packet dissected still contains all filled data (fields). This is because we took into consideration “that each field has its value imposed by the original string” (Scapy, 2022).

The commands used to do this are listed above the section they are in, as well their output (Figure 26 - 28). Some commands have a long output, so these will not be seen listed with the command, however, can be seen in the Figure that corresponds with it. Example: (“command” = output) or (“command” & “command” = output).

First steps:

- “a=IP(ttl=10)” & “a” = < IP ttl=10 |>
- “a.src” = ‘127.0.0.1’
- “a.dst=”192.168.1.1”” & “a” = < IP ttl=10 dst=192.168.1.1 |>
- “a.src” = ‘192.168.8.14’
- “del(a.ttl)” & “a” = < IP dst=192.168.1.1 |>
- “a.ttl” = 64

Stacking layers:

- “IP()” = <IP |>
- “IP()/TCP()” = <IP frag=0 proto=TCP |<TCP |>>
- “Ether()/IP()/TCP()” <Ether type=0x800 |<IP frag=0 proto=TCP |<TCP |>>>
- “IP()/TCP()/"GET / HTTP/1.0\r\n\r\n"” = <IP frag=0 proto=TCP |<TCP |<Raw
load='GET / HTTP/1.0\r\n\r\n' |>>>

```
>>> conf.color_theme = RastaTheme()
>>> a=IP(ttl=10)
>>> a
<IP ttl=10 />
>>> a.src
'127.0.0.1'
>>> a.dst="192.168.1.1"
>>> a
<IP ttl=10 dst=192.168.1.1 />
>>> a.src
'192.168.110.135'
>>> del(a.ttl)
>>> a
<IP dst=192.168.1.1 />
>>> a.ttl
64
>>> IP()
<IP />
>>> IP()/TCP()
<IP frag=0 proto=tcp /<TCP />>
>>> Ether()/IP()/TCP()
<Ether type=IPv4 /<IP frag=0 proto=tcp /<TCP />>>
>>> IP()/TCP()/"GET / HTTP/1.0\r\n\r\n"
<IP frag=0 proto=tcp /<TCP /<Raw load='GET / HTTP/1.0\r\n\r\n' />>>
```

Figure 26 - Ubuntu VM Terminal - Scapy Test - 1

- “Ether()/IP()/IP()/UDP()” = <Ether type=0x800 |<IP frag=0 proto=IP |<IP frag=0 proto=UDP |<UDP |>>>
- “IP(proto=55)/TCP()” = <IP frag=0 proto=55 |<TCP |>
- ‘raw(IP())’ =

```
'E\x00\x00\x14\x00\x01\x00\x00@|\x00|\xe7\x7f\x00\x00\x01\x7f\x00\x00\x01'
```
- “IP(“) = <IP version=4L ihl=5L tos=0x0 len=20 id=1 flags= frag=0L ttl=64 proto=IP
 chksum=0x7ce7 src=127.0.0.1 dst=127.0.0.1 |>
- “a=Ether()/IP(dst="www.slashdot.org")/TCP()/"GET /index.html HTTP/1.0 \n\n"” &
 “hexdump(a)” = (Figure 28).
- “b=raw(a)” & “b” = (Figure 28).
- “c=Ether(b)” & “c” = (Figure 28).
- “c.hide_defaults()” & “c” = (Figure 28).

```
>>> Ether() / IP() / IP() / UDP()
<Ether type=IPv4 |<IP frag=0 proto=ipencap |<IP frag=0 proto=udp |<UDP |>>>
>>> IP(proto=55) / TCP()
<IP frag=0 proto=55 |<TCP |>
>>> raw(IP())
b'E\x00\x00\x14\x00\x01\x00\x00@|\x00|\xe7\x7f\x00\x00\x01\x7f\x00\x00\x01'
>>> IP(_)
<IP version=4 ihl=5 tos=0x0 len=20 id=1 flags= frag=0 ttl=64 proto=hopopt chksum=0x7ce7 src=127.0.0.1 dst=127.0.0.1 |>
```

Figure 27 - Ubuntu VM Terminal - Scapy Test – 2

```
>>> a=Ether()/"IP(dst='www.slashdot.org')/TCP()/"GET /index.html HTTP/1.0 \n\n"
>>> hexdump(a)
0000  00 50 56 F9 14 49 00 0C 29 FC 11 14 08 00 45 00  .PV..I..).....E.
0010  00 43 00 01 00 00 40 06 0F D6 C0 A8 6E B7 CC 44  .C....@.....n..D
0020  6F 6A 00 14 00 50 00 00 00 00 00 00 00 50 02  oj...P.....P.
0030  20 00 52 D3 00 00 47 45 54 20 2F 69 6E 64 65 78  .R...GET /index
0040  2E 68 74 6D 6C 20 48 54 54 50 2F 31 2E 30 20 0A  .html HTTP/1.0 .
0050  0A

>>> b=raw(a)
>>> b
HackTool
b'\x00PV\xf9\x14I\x00\x0c )xfc\x11\x14\x08\x00E\x00\x00\x00\x00\x01\x00\x00@\'x06\x0f\xd6\xc0\xae\x8n\x87\xccDoj\x00
\x14\x00P\x00\x00\x00\x00\x00\x00\x00\x00\x00P\x02 )x00R\xd3\x00\x00GET /index.html HTTP/1.0 \n\n'
>>> c=Ether(b)
>>> c
<Ether dst=00:50:56:f9:14:49 src=00:0c:29:fc:11:14 type=IPv4 |<IP version=4 ihl=5 tos=0x0 len=67 id=1 flags
= frag0 ttl=64 proto=tcp cksum=0xfd6 src=192.168.110.135 dst=204.68.111.106 |<TCP sport=ftp_data dport=htt
p seq=0 ack=0 dataofs=5 reserved=0 flags=S window=8192 cksum=0x52d3 urgptr=0 |<Raw load='GET /index.html HT
TP/1.0 \n\n' >>>
>>> c.hide_defaults()
>>> c
<Ether dst=00:50:56:f9:14:49 src=00:0c:29:fc:11:14 type=IPv4 |<IP ihl=5 len=67 frag=0 proto=tcp cksum=0xfd
6 src=192.168.110.135 dst=204.68.111.106 |<TCP dataofs=5 cksum=0x52d3 |<Raw load='GET /index.html HT
TP/1.0 \n\n' >>>
>>>
```

Figure 28 - Ubuntu VM Terminal - Scapy Test - 3

Reading PCAP Files & Graphical Dumps –

Scapy holds the ability to read packets from .pcap files, as well create and write to a .pcap file (Scapy, 2022). Within the file manager, I located the Scapy directory, selected the ‘test’ folder and selected the ‘pcaps’ folder (Figure 29). This is where some default Scapy pcap files reside (Figure 30). After examining some of these, I decided to use the file named “pfcp.pcap” which sends 200 UDP packets (Figure 30). Then we will create a graphical dump of data. If PyX is installed, users are able to create “a graphical PostScript/PDF dump of a packet or a list of packets” (Scapy, 2020). This allows the user to view data in a picture form (Figure 32). Once again, just as the last section, the command as well its output will be listed above its respected Figure, as well if the output is too long, the Figure that it resides in will replace the output.

- “a=rdpcap("/home/j/scapy/tests/pfcp.pcap")” & “a” = <pfcp.pcap: TCP:0 UDP:200 ICMP:0 Other:0>
- “a=rdpcap("/home/j/scapy/tests/doip_ack.pcap")” & “a” = <doip_ack.pcap: TCP:1 UDP:0 ICMP:0 Other:0>
- “a=rdpcap("/home/j/scapy/tests/http_comppressed.pcap")” & “a” = <http_comppressed.pcap: TCP:3 UDP:0 ICMP:0 Other:0>
- “a=rdpcap("/home/j/scapy/tests/http2_h2c.pcap")” & “a” = <http2_h2c.pcap: TCP:10 UDP:0 ICMP:0 Other:0>
- “a=rdpcap("/home/j/scapy/tests/ecu_trace.pcap.gz")” & “a” = <ecu_trace.pcap.gz: TCP:0 UDP:0 ICMP:0 Other:141>

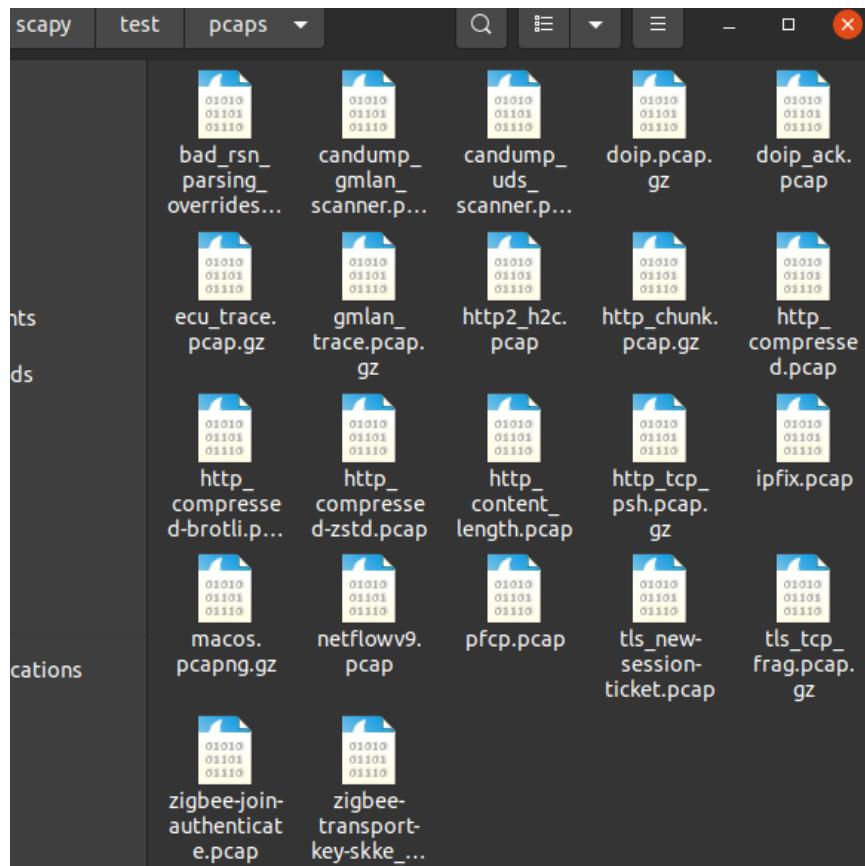


Figure 29 - Ubuntu VM Terminal - Scapy Test - 4

```
>>> a=rdpcap("/home/j/scapy/test/pcaps/pfcp.pcap")
>>> a
<pfcp.pcap: TCP:0 UDP:200 ICMP:0 Other:0>
>>> a=rdpcap("/home/j/scapy/test/pcaps/doip_ack.pcap")
>>> a
<doip_ack.pcap: TCP:1 UDP:0 ICMP:0 Other:0>
>>> a=rdpcap("/home/j/scapy/test/pcaps/http_compressed.pcap")
>>> a
<http_compressed.pcap: TCP:3 UDP:0 ICMP:0 Other:0>
>>> a=rdpcap("/home/j/scapy/test/pcaps/http2_h2c.pcap")
>>> a
<http2_h2c.pcap: TCP:10 UDP:0 ICMP:0 Other:0>
>>> a=rdpcap("/home/j/scapy/test/pcaps/ecu_trace.pcap.gz")
>>> a
<ecu_trace.pcap.gz: TCP:0 UDP:0 ICMP:0 Other:141>
>>>
```

Figure 30 - Ubuntu VM Terminal - Scapy Test - 6

- “a[423].pdfdump(layer_shift=1)” = (Figure 31).

```
>>> a[150].pdfdump(layer shift=1)
/usr/lib/python3.8/subprocess.py:946: ResourceWarning: subprocess 66389 is still running
  warn("subprocess %s is still running" % self.pid,
ResourceWarning: Enable tracemalloc to get the object allocation traceback
>>> No protocol specified
Unable to init server: Could not connect: Connection refused
Cannot parse arguments: Cannot open display:
No protocol specified
Unable to init server: Could not connect: Connection refused
Cannot parse arguments: Cannot open display:
Warning: program returned non-zero exit code #256
Opening "/tmp/scapy6_bjnm0f.pdf" with LibreOffice Draw (application/pdf)
javaldx: Could not find a Java Runtime Environment!
Please ensure that a JVM and the package libreoffice-java-common
is installed.
If it is already installed then try removing ~/.config/libreoffice/4/user/config/javasettings_Linux_*.xml
Warning: failed to read path from javaldx
```

Figure 31 - Ubuntu VM Terminal - Scapy Test - 7

- Graphical Dump Created from previous command

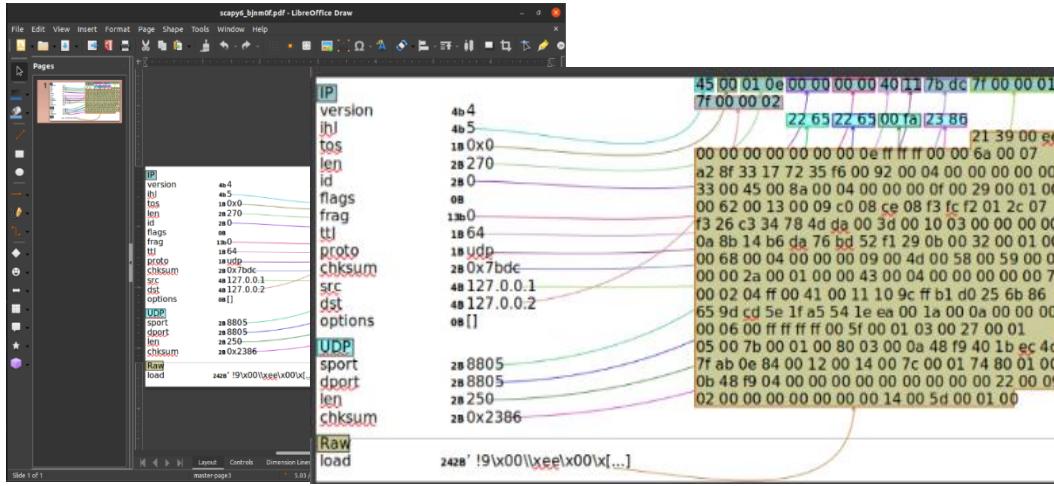


Figure 32 - Ubuntu VM Terminal - Scapy Test - 8

Generating Sets of Packets –

As of now, only one packet has been generated, however, in this section we will be generating sets of packets. Every field of an entire packet can be changed and configured (Scapy, 2022). “This implicitly defines a set of packets, generated using a kind of cartesian product between all the fields” (Scapy, 2022). A few operations will not “work on a set of packets” and this occurs when users forget to “unroll” the packet set (Scapy, 2022). The first element of the list, and only this, when not generated by the user, “will be used to assemble the packet” (Scapy, 2022).

- “a=IP(dst="www.slashdot.org/30)")” & “a” = <IP dst=Net('www.slashdot.org/30') |>
- “[p for p in a]” = [<IP dst=204.68.111.104 |>, <IP dst=204.68.111.105 |>, <IP dst=204.68.111.106 |>, <IP dst=204.68.111.107 |>]
- “b=IP(ttl=[1,2,(5,9)])” & “b” = <IP ttl=[1, 2, (5, 9)] |>
- “[p for p in b]” = [<IP ttl=1 |>, <IP ttl=2 |>, <IP ttl=5 |>, <IP ttl=6 |>, <IP ttl=7 |>, <IP ttl=8 |>, <IP ttl=9 |>]
- “c=TCP(dport=[80,443])” & “[p for p in a/c]” = (Figure 33).

```
>>> a=IP(dst="www.slashdot.org/30")
>>> a
<IP dst=Net('www.slashdot.org/30') |>
>>> [p for p in a]
[<IP dst=204.68.111.104 |>,
 <IP dst=204.68.111.105 |>,
 <IP dst=204.68.111.106 |>,
 <IP dst=204.68.111.107 |>]
>>> b=IP(ttl=[1, 2, (5, 9)])
>>> b
<IP ttl=[1, 2, (5, 9)] |>
>>> [p for p in b]
[<IP ttl=1 |>,
 <IP ttl=2 |>,
 <IP ttl=5 |>,
 <IP ttl=6 |>,
 <IP ttl=7 |>,
 <IP ttl=8 |>,
 <IP ttl=9 |>]
>>> c=TCP(dport=[80, 443])
>>> [p for p in a/c]
[<IP frag=0 proto=tcp dst=204.68.111.104 /<TCP dport=http />>,
 <IP frag=0 proto=tcp dst=204.68.111.104 /<TCP dport=https />>,
 <IP frag=0 proto=tcp dst=204.68.111.105 /<TCP dport=http />>,
 <IP frag=0 proto=tcp dst=204.68.111.105 /<TCP dport=https />>,
 <IP frag=0 proto=tcp dst=204.68.111.106 /<TCP dport=http />>,
 <IP frag=0 proto=tcp dst=204.68.111.106 /<TCP dport=https />>,
 <IP frag=0 proto=tcp dst=204.68.111.107 /<TCP dport=http />>,
 <IP frag=0 proto=tcp dst=204.68.111.107 /<TCP dport=https />>]
>>> _
```

Figure 33 - Ubuntu VM Terminal - Scapy Test - 9

CORE Switch Network Topology

After completing the previous section and moving on to the Sending Packets section in the practice run, I noticed that these commands worked on the Ubuntu VM terminal, but I wanted to see what commands could be sniffed by TCPDump and what ones couldn't. So, for the purpose of this, in this run I decided to create a basic switched environment, however for some reason without realizing it until after completing the entire lab that I used a Hub on accident, so for time sake we will be continuing with a CORE network topology using three nodes (PCs) and one Hub (Figure 34). After placing the nodes, I linked them to the hub. After doing this, I right-clicked each PC and renamed them: PC1, PC2, and PC3-L (Figure 34). PC1 and PC2 will be used to communicate with each other, and PC3-L will be listening to them communicate by sniffing the traffic with TCPDump (Figure 34). As well, I obtained the IP address for these nodes, which are listed below, as well can be seen in Figure 34. After obtaining these IPs, I started the CORE emulation by selecting the green play button (Figure 34).

- PC1: 10.0.0.20
- PC2: 10.0.0.21
- PC3-L: 10.0.0.22

After starting the emulation, to be sure that it is set up correctly, I double-clicked each node to open the terminal (Figure 35). This allows the user to run commands as if they were within the Ubuntu VM terminal, as well allows the user access to anything installed in their Ubuntu VM. After the terminal opened, I used the ping command to ping the other nodes (Figure 35). After this, I used ‘ifconfig’ to view network interface information to gain the default interface. In this case, the interface used was ‘eth0’ (Figure 36). This was successful, the commands used to do this will be listed above their respected Figure.

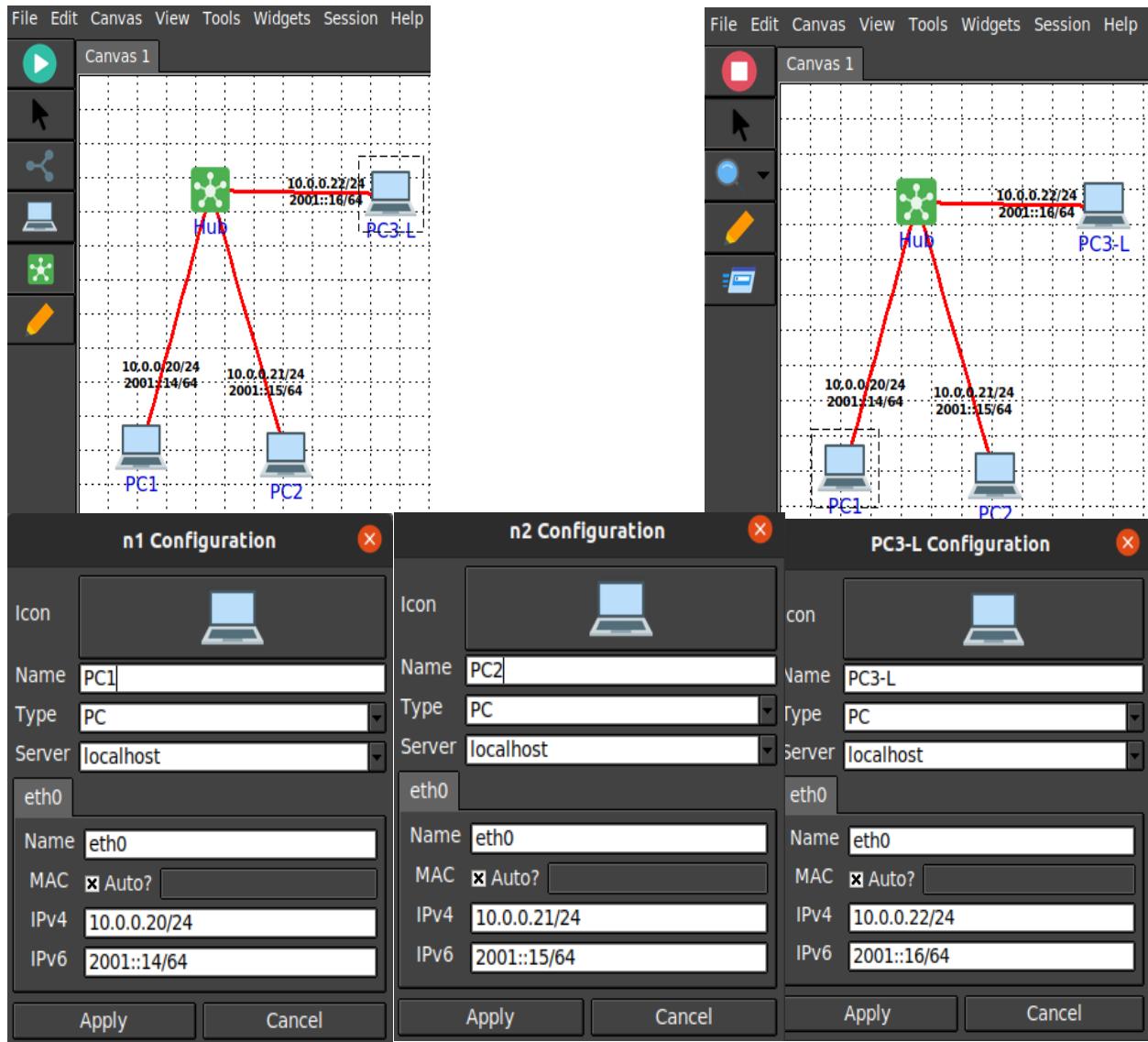


Figure 34 - CORE Network Topology - Steps 1 - 5

- PC1: “ping 10.0.0.21” & “ping 10.0.0.22” (Figure 35)
- PC2: “ping 10.0.0.20” & “ping 10.0.0.22” (Figure 35)
- PC3-L: “ping 10.0.0.20” & “ping 10.0.0.21” (Figure 36)
- “ifconfig” (Figure 36)

```

root@PC1:/tmp/pycore.1/PCI.conf# ping 10.0.0.21
PING 10.0.0.21 (10.0.0.21) 56(84) bytes of data.
64 bytes from 10.0.0.21: icmp_seq=1 ttl=64 time=0.081 ms
64 bytes from 10.0.0.21: icmp_seq=2 ttl=64 time=0.036 ms
^Z
PC1--> PC2
[1]+ Stopped ping 10.0.0.21
root@PC1:/tmp/pycore.1/PCI.conf# ping 10.0.0.22
PING 10.0.0.22 (10.0.0.22) 56(84) bytes of data.
64 bytes from 10.0.0.22: icmp_seq=1 ttl=64 time=0.026 ms
64 bytes from 10.0.0.22: icmp_seq=2 ttl=64 time=0.052 ms
^Z
[2]+ Stopped ping 10.0.0.22
root@PC1:/tmp/pycore.1/PCI.conf#
root@PC2:/tmp/pycore.1/PC2.conf# ping 10.0.0.20
PING 10.0.0.20 (10.0.0.20) 56(84) bytes of data.
64 bytes from 10.0.0.20: icmp_seq=1 ttl=64 time=0.022 ms
64 bytes from 10.0.0.20: icmp_seq=2 ttl=64 time=0.036 ms
^Z
PC1--> PC2
[1]+ Stopped ping 10.0.0.20
root@PC2:/tmp/pycore.1/PC2.conf# ping 10.0.0.22
PING 10.0.0.22 (10.0.0.22) 56(84) bytes of data.
64 bytes from 10.0.0.22: icmp_seq=1 ttl=64 time=0.025 ms
64 bytes from 10.0.0.22: icmp_seq=2 ttl=64 time=0.036 ms
^Z
[2]+ Stopped ping 10.0.0.22
root@PC2:/tmp/pycore.1/PC2.conf#

```

Figure 35 - CORE Network Topology - Steps 6 - 7

```

root@PC3-L:/tmp/pycore.1/PC3-L.conf# ping 10.0.0.20
PING 10.0.0.20 (10.0.0.20) 56(84) bytes of data.
64 bytes from 10.0.0.20: icmp_seq=1 ttl=64 time=0.081 ms
64 bytes from 10.0.0.20: icmp_seq=2 ttl=64 time=0.085 ms
^Z
PC1--> PC2
[1]+ Stopped ping 10.0.0.20
root@PC3-L:/tmp/pycore.1/PC3-L.conf# ping 10.0.0.21
PING 10.0.0.21 (10.0.0.21) 56(84) bytes of data.
64 bytes from 10.0.0.21: icmp_seq=1 ttl=64 time=0.024 ms
64 bytes from 10.0.0.21: icmp_seq=2 ttl=64 time=0.035 ms
^Z
[2]+ Stopped ping 10.0.0.21
root@PC3-L:/tmp/pycore.1/PC3-L.conf#
root@PC3-L:/tmp/pycore.1/PC3-L.conf# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 10.0.0.22 netmask 255.255.255.0 broadcast 0.0.0.0
          10.0.0.22 brd 0.0.0.0 scopeid 0x0<global>
          inet6 fe80::200:ff:fea:2 prefixlen 64 scopeid 0x20<link>
            ether 00:00:00:aa:00:02 txqueuelen 1000  (Ethernet)
              RX packets 94 bytes 10850 (10.8 KB)
              RX errors 0 dropped 0 overruns 0 frame 0
              TX packets 13 bytes 1102 (1.1 KB)
              TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
      inet 127.0.0.1 netmask 255.0.0.0
      inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000  (Local Loopback)
          RX packets 16 bytes 1040 (1.0 KB)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 16 bytes 1040 (1.0 KB)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@PC3-L:/tmp/pycore.1/PC3-L.conf#

```

Figure 36 - CORE Network Topology - Steps 8 - 9

After successfully verifying that the CORE network topology is set-up successfully, using the command “sudo scapy” within the terminal of node PC1, we can access to Scapy interface and continue with the interactive tutorial in the Scapy documentation (Figure 37).

```

root@PC1:/tmp/pynicore_1/PC1.conf# sudo scapy
[sudo: unable to resolve host PC1: Temporary failure in name resolution]
[10.0.0.21/24]
>>> aSPY//YASa
apyyyyCY//////////YCa
sY////////YSpcy  scpCY//Pp | Welcome to Scapy
aYP aYYYYYYYSCP//PP  syY//C | Version 2.4.5rc1.dev228
AYAsAYYYYYYYY///Ps  cY//S |
pCCCCCY//Pc  cSSPscy//Y | https://github.com/secdev/scapy
SPPPPP///a  pP///AC//Y |
A//A  cyP///C | Have fun!
p///Ac  sC///a |
P///YCpc  A//A | Craft packets like it is your last
scccecp///pSP///p  p//Y | day on earth.
sY/////////y_caa  S//P | -- Lao-Tze
cayCyayP//Ya  pY/Ya |
sY/PsY////YCce  aC//Yp |
sc  seesCY//PCypaipyCP//YSa
spCPY/////////YPSpcy
ccacs
using IPython 7.29.0
>>>

```

Figure 37 - CORE Network Topology - Steps 10

Scapy Examples & Results

In this section, we will be continuing from where we left off previously when working on the Scapy interactive tutorial, which was Generating Sets of Packets. The next section is Sending Packets. This section will also cover Fuzzing, Injecting Bytes, Send and Receive Packets, SYN Scans, and finally TCP Traceroute (Scapy, 2022). This section will be laid out the same as the other interactive tutorial section.

Sending Packets

After learning how to manipulate packets, we can now learn how to send them (Scapy, 2022). Using the `send()` function allows the user to “send packets at layer 3” (Scapy, 2022). It will also “handle routing and layer 2” (Scapy, 2022). Using the `sendp()` function also works at layer 2 however it’s up to the user to select the correct interface and layer protocol (Scapy, 2022). “`send()` and `sendp()` will also return sent packet list if `return_packets=True` is passed as parameter” (Scapy, 2022). The commands used are listed above their respected figure.

- “send(IP(dst="10.0.0.21")/ICMP())” = . Sent 1 packets.
 - “sendp(Ether()/IP(dst="10.0.0.21",ttl=(1,4)), iface="eth1")” = Sent 4 packets.
 - “sendp("I'm travelling on Ethernet", iface="eth0", loop=1, inter=0.2)” =^C
Sent 18 packets.
 - “sendp(rdpcap("/home/j/scapy/test/pcaps/pfcp.pcap"))” = Sent 200 packets.

```
>>> send(IP(dst="10.0.0.21")/ICMP())
Sent 1 packets.
>>> sendp(Ether()/IP(dst="10.0.0.21", ttl=(1,4)), iface="eth0")
Sent 4 packets.
>>> sendp("I'm travelling on Ethernet", iface="eth0", loop=1, inter=0.2))
^C
Sent 18 packets.
>>> sendp(rdpcap("/home/j/scapy/test/pcaps/pfcp.pcap")))
Sent 200 packets.
>>>
```

Figure 38 - Scapy Docs Usage Tutorial - Sending Packets – 1

After running the previous commands, I noticed that TCPDump, running in the terminal of PC3-L had sniffed some traffic. This contained a mass amount of data, which means the output takes up a few figures. Most of the data here is ICMP traffic, I did see some payload traffic as well (Figure 39). There was also some ‘who-has’ requests that I noticed (Figure 40). To my surprise, TCPDump captured traffic from an unknown source (Figures 41 & 42). The data in these can be viewed by zooming in this document.

- “tcpdump -v”

```

TERM environment variable not set.
sudo: unable to resolve host PC2: Temporary failure in name resolution
root@PC2:/tmp/pycore.1/PC2.conf# tcpdump -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
21:20:10.174877 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) fe80::1048:e5ff:fe4:4b9d > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16
    source link-address option (1), length 8 (1): 12:48:e5:f4:4b:9d
21:20:14.270149 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) fe80::200:ff:fea:0 > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16
    source link-address option (1), length 8 (1): 00:00:00:aa:00:00
21:20:20.413939 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) fe80::289a:dcff:fe04:6a7b > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16
    source link-address option (1), length 8 (1): 12:48:e5:f4:4b:9d
21:20:20.413962 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) fe80::200:ff:fea:2 > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16
    source link-address option (1), length 8 (1): 00:00:00:aa:00:02
21:20:28.868633 IP6 (flowlabel 0x11aa9, hlim 255, next-header UDP (17) payload length: 149) fe80::289a:dcff:f
e04:6a7b.mdns > ff02::fb.mdns: [bad udp cksum 0x6d3f -> 0xacf4!] 0 [9q] PTR (QM)? _nfs._tcp.local. PTR (QM)?
_ipp._tcp.local. PTR (QM)? _ipps._tcp.local. PTR (QM)? _ftp._tcp.local. PTR (QM)? _webdav._tcp.local. PTR (QM)?
_webdavs._tcp.local. PTR (QM)? _sftp-ssh._tcp.local. PTR (QM)? _smb._tcp.local. PTR (QM)? _afpovertcp._tcp.
.local. (141)
21:20:28.934844 IP6 (flowlabel 0xf2ad5, hlim 255, next-header UDP (17) payload length: 149) fe80::1048:e5ff:f
e4:4b9d.mdns > ff02::fb.mdns: [bad udp cksum 0x3fff -> 0xda34!] 0 [9q] PTR (QM)? _nfs._tcp.local. PTR (QM)?
_ipp._tcp.local. PTR (QM)? _ipps._tcp.local. PTR (QM)? _ftp._tcp.local. PTR (QM)? _webdav._tcp.local. PTR (QM)?
_webdavs._tcp.local. PTR (QM)? _sftp-ssh._tcp.local. PTR (QM)? _smb._tcp.local. PTR (QM)? _afpovertcp._tcp.
.local. (141)
21:21:13.661964 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) fe80::200:ff:fea:1 > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16
    source link-address option (1), length 8 (1): 00:00:00:aa:00:01

```

Figure 39 - Scapy Docs Usage Tutorial - Sending Packets - 2

```

21:22:48.847180 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.21 tell 10.0.0.20, length 28
21:22:48.847207 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.21 is-at 00:00:00:aa:00:01 (oui Ethernet),
  si///y...caa S/E                                         -- sebastien Chabal
length 28
21:22:48.867229 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto ICMP (1), length 28)
  10.0.0.20 > 10.0.0.21: ICMP echo request, id 0, seq 0, length 8
21:22:48.867271 IP (tos 0x0, ttl 64, id 39953, offset 0, flags [none], proto ICMP (1), length 28)
  10.0.0.21 > 10.0.0.20: ICMP echo reply, id 0, seq 0, length 8
21:22:54.014804 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.20 tell 10.0.0.21, length 28
21:22:54.014873 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.20 is-at 00:00:00:aa:00:00 (oui Ethernet),
length 28
21:23:37.522605 IP (tos 0x0, ttl 1, id 1, offset 0, flags [none], proto Options (0), length 20)
  10.0.0.20 > 10.0.0.21: hopopt 0
21:23:37.522639 IP (tos 0xc0, ttl 64, id 44279, offset 0, flags [none], proto ICMP (1), length 48)
  10.0.0.21 > 10.0.0.20: ICMP 10.0.0.21 protocol 0 unreachable, length 28
    IP (tos 0x0, ttl 1, id 1, offset 0, flags [none], proto Options (0), length 20)
      >>> send("IP travelling on Ethernet", iface='eth0', loop=1, inter=0)
  10.0.0.20 > 10.0.0.21: hopopt 0
21:23:37.523064 IP (tos 0x0, ttl 2, id 1, offset 0, flags [none], proto Options (0), length 20)
  10.0.0.20 > 10.0.0.21: hopopt 0

```

Figure 40 - Scapy Docs Usage Tutorial - Sending Packets - 3

```

21:25:07.979355 00:00:40:11:7c:80 (oui Unknown) > 45:00:00:6a:00:00 (oui Unknown), ethertype Unknown (0x7f00)
, length 106:
  0x0000: 0001 7f00 0002 2265 2265 0056 e792 2139 ....."e"e.V.!9
  0x0010: 004a 0000 0000 ffff ffff ff00 0085 .J.....
  0x0020: 0013 0d36 6856 656f 0b79 2fd7 30a6 a32e ...6hVeo.y/0...
  0x0030: 92d7 3b1c 4700 0f00 2300 8e00 0101 0013 ..;G..#....
  0x0040: 0001 4b00 7c00 0100 007c 0001 ff00 2600 ..K/....!....&
  0x0050: 0b03 0008 396f 1d03 8198 2306 .....9o....#.
21:25:07.979415 00:00:40:11:7c:b6 (oui Unknown) > 45:00:00:34:00:00 (oui Unknown), ethertype Unknown (0x7f00)
, length 52:
  0x0000: 0001 7f00 0002 2265 2265 0020 9b2f 2139 ....."e"e...!/9
  0x0010: 0014 0000 0000 0000 0000 0000 0060 .....`.....
  0x0020: 0004 0000 0000 .....using IPython 7.1.0
21:25:07.979450 00:00:40:11:7c:5e (oui Unknown) > 45:00:00:8c:00:00 (oui Unknown), ethertype Unknown (0x7f00)
, length 140:
  0x0000: 0001 7f00 0002 2265 2265 0078 f7e3 2103 ....."e"e.x..!.
  0x0010: 006c 0000 0000 0000 0006 ffff ff00 0092 .1.....
  0x0020: 0004 0000 0001 0072 0005 0100 0000 1400 .....r.....
  0x0030: 2600 0a00 0007 b652 9532 3cbd d200 6700 &....R.2<..g.
  0x0040: 0502 cb26 5caa 0017 0011 0500 0009 7377 ..&\.....sw
  0x0050: b587 7297 b0ac 2100 0000 0000 2b00 02bd ..r...!....+.
  0x0060: 0200 8d00 1903 0733 3938 3235 3530 0f32 .....3982550.2
  0x0070: 3838 3530 3337 3337 3533 3230 3538 88503737532058
21:25:07.979543 00:00:40:11:7c:97 (oui Unknown) > 45:00:00:53:00:00 (oui Unknown), ethertype Unknown (0x7f00)
, length 83:
  0x0000: 0001 7f00 0002 2265 2265 003f 0c2e 2139 ....."e"e.?..!9
  0x0010: 0033 0000 0000 ffff ffff ffff ff00 008a .3.....
  0x0020: 0004 ffff ffff 0017 000b 0e00 0000 0000 .....
  0x0030: 0002 0000 2481 f400 0c48 f983 50a3 f4fa ...$...H..P...
  0x0040: 9aa6 d5c1 ad ..... .

```

Figure 41 - Scapy Docs Usage Tutorial - Sending Packets - 4

```

21:25:07.986121 00:00:40:11:7c:08 (oui Unknown) > 45:00:00:a2:00:00 (oui Unknown), ethertype Unknown (0x7E00)
, length 226:
0x0000: 0001 7E00 0002 2265 2265 00ce f635 2103 .... "e"e..5!.
0x0010: 00c2 0000 0000 0000 0012 0000 0000 003a .....
0x0020: 000a 0070 0001 1E00 2700 0107 0058 0001 ..p....'...X..
0x0030: 3100 0800 1d00 5800 0100 002f 0001 ff00 1....X..../
0x0040: 4c00 0400 0000 0000 9000 0701 2b52 83a8 L.....+R..
0x0050: 3335 008a 0004 0000 0000 0039 000d 0200 35.....9...
0x0060: 0000 0000 0000 00ad 2f31 5b00 5c00 0a02 ...../1[.\...
0x0070: 0007 ea02 4892 bca3 e800 3600 2600 3900 ....H....6.6.9.
0x0080: 0d02 ffff ffff ffff 3eb0 dc8f 0070 .....?...P
0x0090: 0001 4000 8500 0100 0037 0001 4c00 2b00 ..@....7..L+.
0x00a0: 02c9 0300 7300 0501 ffff ffff 005a 0001 ...s.....Z..
0x00b0: 0c00 8d00 1E03 0c30 3639 3739 3437 3932 .....069794792
0x00c0: 3236 3410 3437 3838 3032 3738 3232 3130 264.478802782210
0x00d0: 3034 3230 .....0420

```



```

21:25:07.986252 00:00:40:11:7c:44 (oui Unknown) > 45:00:00:a6:00:00 (oui Unknown), ethertype Unknown (0x7E00)
, length 166:
0x0000: 0001 7E00 0002 2265 2265 0092 5029 2102 .... "e"e..P)!.
0x0010: 0086 0000 0000 0000 ffff ffff 0080 .....
0x0020: 0018 0018 0008 69dc ae40 76b4 a786 0087 ....i..@v...
0x0030: 0003 0450 df00 1E00 0100 0048 0009 01ff ..P.....H..
0x0040: ffff ffff ffff ffff 4300 0400 0000 0000 .....C...
0x0050: 8f00 1d00 2200 1907 ffff ffff ffff ....."
0x0060: 0000 0000 0000 0007 0000 0000 0000 0017 .....
0x0070: 0027 0001 0600 4500 0400 0000 0000 2b00 .....K+.
0x0080: 02b8 0b00 4200 1103 ffff ffff ffff .....B...
0x0090: 0000 0000 0000 0004 ..... .

```

Figure 42 - Scapy Docs Usage Tutorial - Sending Packets - 5

The last command from the Sending Packets section:

- “send(IP(dst='127.0.0.1'), return_packets=True)” = . Sent 1 packets. <PacketList: TCP:0 UDP:0 ICMP:0 Other:1>

```

>>> send(IP(dst='127.0.0.1'), return_packets=True)
.
Sent 1 packets.
<PacketList: TCP:0 UDP:0 ICMP:0 Other:1>

```

Figure 43 - Scapy Docs Usage Tutorial - Sending Packets - 6

Fuzzing

Using the fuzz() function allows the user the ability “to change any default value that is not to be calculated by an object whose value is random and whose type is adapted to the field” (Scapy, 2022). This works by enabling quick “building fuzzing templates and sending them in a loop” (Scapy, 2022).

- “send(IP(dst="10.0.0.21")/fuzz(UDP()/NTP(version=4)),loop=1) = ...^C Send 129

Packets

```
>>> send(IP(dst="10.0.0.21")/fuzz(UDP()/NTP(version=4)),loop=1)
...>>> receive timestamp: 3409069056.411350715 (2008/01/11 14:37:36)
...>>> ^C
Sent 129 packets.
>>>
```

Figure 44 - Scapy Docs Usage Tutorial - Fuzzing - 1

This command also had traffic sniffed by TCPDump. The output contains who-has requests, as well replays (Figure 45 & 46). As well, it contains timestamp outputs (Figure 45 & 46). The command used to capture this was “tcpdump -v” (Figure 45). Again, to view this Figures it may be best to zoom this document in.

```
root@PC0: /tmp/pycore.1/PC2.conf# tcpdump -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
21:30:39.623547 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.21 tell 10.0.0.20, length 28
21:30:39.623603 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.21 is-at 00:00:00:aa:00:01 (oui Ethernet), length 28
21:30:39.639266 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 76)
 10.0.0.20.ntp > 10.0.0.21.ntp: NTPv4, length 48
    Server, Leap indicator: -ls (128), Stratum 141 (reserved), poll 193 (2s), precision 70
    Root Delay: 700.433425, Root dispersion: 6651.736328, Reference-ID: 223.113.218.118
    Reference Timestamp: 3409069056.411350715 (2008/01/11 14:37:36)
    Originator Timestamp: 3858715839.639125347 (2022/04/11 21:30:39)
    Receive Timestamp: 1556204212.270442444 (2085/05/31 17:25:08)
    Transmit Timestamp: 3858715839.639137268 (2022/04/11 21:30:39)
    Originator - Receive Timestamp: +1992455668.631317097
    Originator - Transmit Timestamp: +0.000011920
21:30:39.639287 IP (tos 0x0, ttl 64, id 5861, offset 0, flags [none], proto ICMP (1), length 104)
  10.0.0.21 > 10.0.0.20: ICMP 10.0.0.21 udp port ntp unreachable, length 84
    IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 76)
      10.0.0.20.ntp > 10.0.0.21.ntp: NTPv4, length 48
        Server, Leap indicator: -ls (128), Stratum 141 (reserved), poll 193 (2s), precision 70
        Root Delay: 700.433425, Root dispersion: 6651.736328, Reference-ID: 223.113.218.118
        Reference Timestamp: 3409069056.411350715 (2008/01/11 14:37:36)
        Originator Timestamp: 3858715839.639125347 (2022/04/11 21:30:39)
        Receive Timestamp: 1556204212.270442444 (2085/05/31 17:25:08)
        Transmit Timestamp: 3858715839.639137268 (2022/04/11 21:30:39)
        Originator - Receive Timestamp: +1992455668.631317097
        Originator - Transmit Timestamp: +0.000011920
```

Figure 45 - Scapy Docs Usage Tutorial - Fuzzing - 2

```

21:30:39.640653 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 76)
  10.0.0.20.ntp > 10.0.0.21.ntp: NTPv4, length 48
    symmetric passive, Leap indicator: +1s (64), Stratum 123 (reserved), poll 5 (32s), precision -105
    Root Delay: 42338.480041, Root dispersion: 42427.381271, Reference-ID: 98.220.154.100
    Reference Timestamp: 38702508135.966751331 (2058/05/12 23:37:11)
    Originator Timestamp: 3858715839.640588760 (2022/04/11 21:30:39)
    Receive Timestamp: 3977233048.627023092 (2026/01/12 13:57:28)
    Transmit Timestamp: 3858715839.640599250 (2022/04/11 21:30:39) dev/scapy
    Originator - Receive Timestamp: +118517208.986434332
    Originator - Transmit Timestamp: +0.000010490 un/
21:30:39.640663 IP (tos 0xc0, ttl 64, id 5862, offset 0, flags [none], proto ICMP (1), length 104)
  10.0.0.21 > 10.0.0.20: ICMP echo request, length 84
  IP (tos 0x0, ttl 64, pid 1, offset 0, flags [none], proto UDP (17), length 76)
  10.0.0.20.ntp > 10.0.0.21.ntp: NTPv4, length 48 the wires and in the waves.
    symmetric passive/Leap indicator: +1s (64), Stratum 123 (reserved), poll 5 (32s), precision -105
    Root Delay: 42338.480041, Root dispersion: 42427.381271, Reference-ID: 98.220.154.100
    Reference Timestamp: 38702508135.966751331 (2058/05/12 23:37:11)
    Originator Timestamp: 3858715839.640588760 (2022/04/11 21:30:39)
    Receive Timestamp: 3977233048.627023092 (2026/01/12 13:57:28)
    Transmit Timestamp: 3858715839.640599250 (2022/04/11 21:30:39)
    >>> Originator - Receive Timestamp: +118517208.986434332 (loop=1)
    ... Originator - Transmit Timestamp: +0.000010490

```

Figure 46 - Scapy Docs Usage Tutorial - Fuzzing - 3

Injecting Bytes

Now we can inject bytes into packets. Each packets have fields have a specific type (Scapy, 2022). An example is that “the length field of the IP packet ‘len’ expects an integer. More on that later. If you’re developing a PoC, there are times where you’ll want to inject some value that doesn’t fit that type. This is possible using RawVal” (Scapy, 2022).

- “`pkt = IP(len=RawVal(b"NotAnInteger"), src="127.0.0.1")`” & “`bytes(pkt)`” =
`b'H\x00NotAnInt\x0f\xb3er\x00\x01\x00\x00@\x00\x00\x00\x7f\x00\x00\x01\x7f\x00\x00\x00\x01\x00\x00\x00\x01\x00\x00\x00'`

```

>>> pkt = IP(len=RawVal(b"NotAnInteger"), src="127.0.0.1")
>>> bytes(pkt)
b'H\x00NotAnInt\x0f\xb3er\x00\x01\x00\x00@\x00\x00\x00\x7f\x00\x00\x01\x7f\x00\x00\x00\x01\x00\x00\x00\x01\x00\x00\x00'
>>>

```

Figure 47 - Scapy Docs Usage Tutorial - Injecting Bytes – 1

Send & Receive Packets

Now we can try to send and receive packets using Scapy. Using the sr() function allows the user to send packets as well receive the answers (Scapy, 2022). This “The function returns a couple of packet and answers, and the unanswered packets” (Scapy, 2022). Using the sr1(), a version of sr(), which “only returns one packet that answered the packet (or the packet set) sent” as well these “packets must be layer 3 packets” (Scapy, 2022). Using the srp() function does this same thing except “for layer 2 packets (Ethernet, 802.3, etc.). If there is no response, a None-value will be assigned instead when the timeout is reached” (Scapy, 2022).

Send and Receive functions are the main part of Scapy, and they “return a couple of two lists. The first element is a list of couples (packet sent, answer), and the second element is the list of unanswered packets. These two elements are lists, but they are wrapped by an object to present them better, and to provide them with some methods that do most frequently needed actions:” (Figure 49) (Scapy, 2022).

When a “limited rate of answers” is returned, the user “can specify a time interval (in seconds) to wait between two packets with the inter parameter” (Scapy, 2022). Some packets may be lost if the interval is not enough , however users “can resend all the unanswered packets, either by calling the function again, directly with the unanswered list, or by specifying a retry parameter” (Scapy, 2022). As well, “he timeout parameter specify the time to wait after the last packet has been sent:” (Figure 50) (Scapy, 2022).

- “p = sr1(IP(dst="10.0.0.21")/ICMP()/"XXXXXXXXXXXX")” = Begin emission: ...

Finished to send 1 packets. .* Received 2 packets, got 1 answers, remaining 0 packets

- “p” & “p.show()” = (Figure 48).

```

>>> p = sr1(IP(dst="10.0.0.21")/ICMP()/"XXXXXXXXXX"))
Begin emission:
Received 2 packets, get 1 answers, remaining 0 packets
>>> pp
<IP version=4 ihl=5 tos=0x0 len=39 id=59660 flags= frag=0 ttl=64 proto=icmp checksum=0x7dal src=10.0.0.21 dst=10.0.0.20 |<ICMP type=echo-reply code=0 checksum=0xee45 id=0x0 seq=0x0 unused='' |<Raw load='XXXXXXXXXX' />>>
>>> p.show()
<bound method Packet.show of <IP version=4 ihl=5 tos=0x0 len=39 id=59660 flags= frag=0 ttl=64 proto=icmp checksum=0x7dal src=10.0.0.21 dst=10.0.0.20 |<ICMP type=echo-reply code=0 checksum=0xee45 id=0x0 seq=0x0 unused='' |<Raw load='XXXXXXXXXX' />>>

```

Figure 48 - Scapy Docs Usage Tutorial - Send & Receive Packets - 1

“A DNS query (rd = recursion desired)” (Scapy, 2022).

- “sr1(IP(dst="10.0.0.21")/UDP()/DNS(rd=1,qd=DNSQR(qname="www.slashdot.org")))” = (Figure 49).
- “sr(IP(dst="10.0.0.21")/TCP(dport=[21,22,23]))” = (Figure 49).
- “ans, unans = _” = (Figure 49).
- “ans.summary()” = (Figure 49).

```

>>> sr1(IP(dst="10.0.0.21")/UDP()/DNS(rd=1,qd=DNSQR(qname="10.0.0.20"))))
Begin emission:
=====
Finished sending 1 packets.
=====
Received 2 packets, got 1 answers, remaining 0 packets
<IP version=4 ihl=5 tos=0xc0 len=83 id=54654 flags= frag=0 ttl=64 proto=icmp cksum=0x9043 src=10.0.0.21 dst=10.0.0.20 |<ICMP type=dest-unreach code=port-unreachable cksum=0x115a reserved=0 length=0 nexthopmtu=0 unused='' /<IPerror version=4 ihl=5 tos=0x0 len=55 id=1 flags= frag=0 ttl=64 proto=udp cksum=0x668d src=10.0.0.20 dst=10.0.0.21 |<UDPerrror sport=domain dport=domain len=35 cksum=0x23af |<DNS id=0 qr=0 opcode=QUERY aa=0 tc=0 rd=1 ra=0 z=0 ad=0 rcode=ok qdcount=1 ancount=0 nscount=0 arcount=0 qd=<DNSQR qname='10.0.0.20.' qtype=A qclass=IN /> an=None ns=None ar=None />>>>
>>> sr(IP(dst="10.0.0.21")/TCP(dport=[21,22,23]))
Begin emission:
=====
Finished sending 3 packets.
=====
*** Received 3 packets, got 3 answers, remaining 0 packets
(<Results: TCP:3 UDP:0 ICMP:0 Other:0>,
 <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
>>> ans, unans = __
>>> ans.summary()
IP / TCP 10.0.0.20:ftp_data > 10.0.0.21:ftp S ==> IP / TCP 10.0.0.21:ftp > 10.0.0.20:ftp_data RA
IP / TCP 10.0.0.20:ftp_data > 10.0.0.21:ssh S ==> IP / TCP 10.0.0.21:ssh > 10.0.0.20:ftp_data RA
IP / TCP 10.0.0.20:ftp_data > 10.0.0.21:telnet S ==> IP / TCP 10.0.0.21:telnet > 10.0.0.20:ftp_data RA
>>>                                         Start ran for 0.605 seconds
                                                ZOOM 100%    CPU 1.00%    Alerts

```

Figure 49 - Scapy Docs Usage Tutorial - Send & Receive Packets - 2

- “sr(IP(dst="172.20.29.5/30")/TCP(dport=[21,22,23]),inter=0.5,retry=-2,timeout=1)” = (Figure 50).

```

>>> sr(IP(dst="10.0.0.21")/TCP(dport=[21,22,23]),inter=0.5,retry=-2,timeout=1)
Begin emission:
=====
Finished sending 3 packets.
=====
*** Received 4 packets, got 3 answers, remaining 0 packets
(<Results: TCP:3 UDP:0 ICMP:0 Other:0>,
 <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
>>>                                         Start ran for 0.605 seconds
                                                ZOOM 100%

```

Figure 50 - Scapy Docs Usage Tutorial - Send & Receive Packets - 3

SYN Scans

The classic Scapy “SYN Scan can be initialized by executing” the command seen in Figure 51. This “will send a single SYN packet to” PC2 on “port 80 and will quit after receiving a single response” (Scapy, 2022). The output from this shows that PC2 “returned “RA” flag indicating an open port” and scans “ports 400 through 443 on the system” (Figure 51) (Scapy, 2022). We can view “responses simply request a summary of collected packets:” (Figure 54) (Scapy, 2022). This can be shown as “stimulus/response pairs for answered probes” and “can display only the information we are interested in by using a simple loop:” (Figure 54). We can even build a table “using the make_table() function to display information about multiple targets” (Figure 57). The next two examples after this did not work, however it “will even print the ICMP error type if the ICMP packet was received as a response instead of expected TCP” (Figure 57) (Scapy, 2022). When performing large scans, we can choose to only display “certain responses” (Figure 57) (Scapy, 2022). There is also an option to perform an analysis on the responses, like indicating “which ports are open: Again, for larger scans we can build a table of open ports:” (Figure 57) (Scapy, 2022). If these methods don’t work or are “not enough, Scapy includes a report_ports() function which not only automates the SYN scan, but also produces a LaTeX output with collected results:” (Figure 57) (Scapy, 2022). The commands used as well their outputs will be listed above their repeated Figures, and the same goes for traffic sniffed by TCPDump running in PC3-L.

- “sr1(IP(dst="10.0.0.21")/TCP(dport=80,flags="S")) = (Figure 51)."
- “sr(IP(dst="10.0.0.21")/TCP(sport=666,dport=(440,443),flags="S"))” = (Figure 51).

```
>>> sr1(IP(dst="10.0.0.21")/TCP(dport=80,flags="S"))
Begin emission:
[00:07:11:19:06:54] ARP, ethernet (len 6), IPv4 (len 4)
Finished sending 1 packets.
* [00:07:11:19:06:54] ARP, ethernet (len 6), IPv4 (len 4)
Received 2 packets, got 1 answers, remaining 0 packets
<IP version=4 ihl=5 tos=0x0 len=40 id=0 flags=DF frag=0 ttl=64 proto=tcp cksum=0x26a8 src=10.0.0.21 dst=10.0.0.20 |<TCP sport=http dport=ftp_data seq=0 ack=1 dataofs=5 reserved=0 flags=RA window=0 cksum=0xb43 urgptr=0 />
>>> sr(IP(dst="10.0.0.21")/TCP(sport=666,dport=(440,443),flags="S"))
Begin emission:
[00:07:11:19:06:54] TCP, ethernet (len 6), IPv4 (len 4)
Finished sending 4 packets.
*** [00:07:11:19:06:54] TCP, ethernet (len 6), IPv4 (len 4)
Received 4 packets, got 4 answers, remaining 0 packets
(<Results: TCP:4 UDP:0 ICMP:0 Other:0>, <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
```

Figure 51 - Scapy Docs Usage Tutorial - SYN Scans - 1

On PC3-L using TCPDump, I was able to capture traffic after running the two previous commands. This consisted of ICMP packets as well payloads, router solicitation packets, and IPv6 payload packets (Figure 52). Then I saw who-has request/replay packets and messages stating “correct, none, DF” (Figure 53). To view the Figures this document should be zoomed in.

- “tcpdump -v”

```
root@PC3-L:/tmp/pycore.1/PC3-L.conf# tcpdump -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:05:32.382217 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) fe80::200:ff:feaa:1 > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16
    Begin source link-address option (1), length 8 (1): 00:00:00:aa:00:01
12:05:33.406467 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) fe80::bc72:30ff:fe6b:98d6 > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16
    Recv source link-address option (1), length 8 (1): 5a:cb:6f:11:e2:65
12:05:34.430287 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) fe80::a4e9:a5ff:feb1:85e3 > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16
    http sport=ftp_data seq=0 ack=1
    source link-address option (1), length 8 (1): a6:e9:a5:b1:85:e3
12:05:34.686619 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) fe80::200:ff:feaa:2 > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16
    Finish source link-address option (1), length 8 (1): 00:00:00:aa:00:02
12:05:34.686715 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) fe80::200:ff:feaa:0 > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16
    (<Result> source link-address option (1), length 8 (1): 00:00:00:aa:00:00
12:05:35.807296 IP6 (flowlabel 0xc53a2, hlim 255, next-header UDP (17) payload length: 53) fe80::bc72:30ff:fe6b:98d6.mdn
s > ff02::fb.mdns: [bad udp cksum 0x8379 -> 0x5104!] 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
12:05:37.312233 IP6 (flowlabel 0xa0171, hlim 255, next-header UDP (17) payload length: 53) fe80::a4e9:a5ff:feb1:85e3.mdn
s > ff02::fb.mdns: [bad udp cksum 0xce43 -> 0x063a!] 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
12:05:47.230047 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) fe80::200:ff:feaa:1 > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16
    (<Result> source link-address option (1), length 8 (1): 00:00:00:aa:00:01
```

Figure 52 - Scapy Docs Usage Tutorial - SYN Scans - 2

```

12:06:26.907943 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) fe80::bc72:30ff:fe6b:98d6 > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16
    source link-address option (1), length 8 (1): 5a:cb:6f:11:e2:65
12:06:26.907954 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) fe80::200:ff:fea:0 > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16 sport=http dport=ftp_data seq=0 ack=1
    source link-address option (1), length 8 (1): 00:00:00:aa:00:00
12:06:30.000107 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.21 tell 10.0.0.20, length 28
12:06:30.000129 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.21 is-at 00:00:00:aa:00:01 (oui Ethernet), length 28
12:06:30.015601 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.http: Flags [S], cksum 0x7b56 (correct), seq 0, win 8192, length 0
12:06:30.015625 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.http > 10.0.0.20.ftp-data: Flags [R.], cksum 0x9b43 (correct), seq 0, ack 1, win 0, length 0
12:06:35.099999 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.20 tell 10.0.0.21, length 28
12:06:35.100005 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.20 is-at 00:00:00:aa:00:00 (oui Ethernet), length 28
12:07:05.955857 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.666 > 10.0.0.21.440: Flags [S], cksum 0x7768 (correct), seq 0, win 8192, length 0
12:07:05.955877 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.440 > 10.0.0.20.666: Flags [R.], cksum 0x9755 (correct), seq 0, ack 1, win 0, length 0
12:07:05.956506 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.666 > 10.0.0.21.441: Flags [S], cksum 0x7767 (correct), seq 0, win 8192, length 0
12:07:05.956515 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.441 > 10.0.0.20.666: Flags [R.], cksum 0x9754 (correct), seq 0, ack 1, win 0, length 0
12:07:05.957091 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.666 > 10.0.0.21.442: Flags [S], cksum 0x7766 (correct), seq 0, win 8192, length 0
12:07:05.957099 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.442 > 10.0.0.20.666: Flags [R.], cksum 0x9753 (correct), seq 0, ack 1, win 0, length 0
12:07:05.957697 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.666 > 10.0.0.21.https: Flags [S], cksum 0x7765 (correct), seq 0, win 8192, length 0
12:07:05.957706 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.https > 10.0.0.20.666: Flags [R.], cksum 0x9752 (correct), seq 0, ack 1, win 0, length 0
12:07:11.190632 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.20 tell 10.0.0.21, length 28
12:07:11.190651 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.20 is-at 00:00:00:aa:00:00 (oui Ethernet), length 28

```

Figure 53 - Scapy Docs Usage Tutorial - SYN Scans - 3

- “ans, unans = _” & “ans.summary()” = (Figure 54)
- “ans.summary(lambda s,r: r.printf("%TCP.sport% \t %TCP.flags%"))” = (Figure 54).

```

>>> ans, unans =
10.0.0.20/24-10.0.0.21/24:
>>> ans.summary()
IP / TCP 10.0.0.20:666 > 10.0.0.21:440 S ==> IP / TCP 10.0.0.21:440 > 10.0.0.20:
666 RA
IP / TCP 10.0.0.20:666 > 10.0.0.21:441 S ==> IP / TCP 10.0.0.21:441 > 10.0.0.20:
666 RA
IP / TCP 10.0.0.20:666 > 10.0.0.21:442 S ==> IP / TCP 10.0.0.21:442 > 10.0.0.20:
666 RA
IP / TCP 10.0.0.20:666 > 10.0.0.21:https S ==> IP / TCP 10.0.0.21:https > 10.0.0.
.20:666 RA
>>> ans.summary( lambda s,r: r.printf("%TCP.sport% \t %TCP.flags%") )
440     RA
441     RA
442     RA
https   RA

```

[1]+ Stopped topdump -v
root@PC3-L:/tmp/pycore.1/PC3-L.conf# tcpdump -v
listening on eth0 link-type EN10MB (IP)

Figure 54 - Scapy Docs Usage Tutorial - SYN Scans - 4

TCPDump again was able to sniff traffic after the commands above executed. This captured again ICMP packets, ARP Ethernet who-has request/reply packets (Figure 55 & 56).

- “tcpdump -v”

```
root@PC3-L:/tmp/pycore.1/PC3-L.conf# tcpdump -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:24:28.169791 IP6 (slim 255, next-header ICMPv6 (58) payload length: 16) fe80::bc72:30ff:feab:98d6 > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16
    source link-address option (1), length 8 (1): 5a:cb:6f:11:e2:65
12:25:21.411221 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto ICMP (1), length 39)
    10.0.0.20 > 10.0.0.21: ICMP echo request, id 0, seq 0, length 19
12:25:21.411253 IP (tos 0x0, ttl 64, id 61938, offset 0, flags [none], proto ICMP (1), length 39)
    10.0.0.21 > 10.0.0.20: ICMP echo reply, id 0, seq 0, length 19
12:25:26.537803 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.20 tell 10.0.0.21, length 28
12:25:26.537825 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.20 is-at 00:00:00:aa:00:00 (oui Ethernet), length 28
12:26:40.330417 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.21 tell 10.0.0.20, length 28
12:26:40.330434 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.21 is-at 00:00:00:aa:00:01 (oui Ethernet), length 28
12:26:40.347076 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 55)
    10.0.0.20.domain > 10.0.0.21.domain: 0 A? 10.0.0.20. (27)
12:26:40.347100 IP (tos 0xc0, ttl 64, id 9506, offset 0, flags [none], proto ICMP (1), length 83)
    10.0.0.21 > 10.0.0.20: ICMP 10.0.0.21 udp port domain unreachable, length 63
    IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 55)
    10.0.0.20.domain > 10.0.0.21.domain: 0 A? 10.0.0.20. (27)
12:26:45.385637 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.20 tell 10.0.0.21, length 28
12:26:45.385644 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.20 is-at 00:00:00:aa:00:00 (oui Ethernet), length 28
12:27:36.831569 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.ftp: Flags [S], cksum 0x7b91 (correct), seq 0, win 8192, length 0
12:27:36.831600 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.ftp > 10.0.0.20.ftp-data: Flags [R.], cksum 0xb7e (correct), seq 0, ack 1, win 0, length 0
12:27:36.832156 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.ssh: Flags [S], cksum 0x7b90 (correct), seq 0, win 8192, length 0
```

Figure 55 - Scapy Docs Usage Tutorial - SYN Scans - 5

```
12:28:43.402226 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.20 tell 10.0.0.21, length 28
12:28:43.402234 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.20 is-at 00:00:00:aa:00:00 (oui Ethernet), length 28
12:29:06.990489 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.21 tell 10.0.0.20, length 28
12:29:06.990506 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.21 is-at 00:00:00:aa:00:01 (oui Ethernet), length 28
12:29:07.006275 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.http: Flags [S], cksum 0x7b56 (correct), seq 0, win 8192, length 0
12:29:07.006296 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.http > 10.0.0.20.ftp-data: Flags [R.], cksum 0xb43 (correct), seq 0, ack 1, win 0, length 0
12:29:48.123582 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.666 > 10.0.0.21.440: Flags [S], cksum 0x7768 (correct), seq 0, win 8192, length 0
12:29:48.123600 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.440 > 10.0.0.20.666: Flags [R.], cksum 0x9755 (correct), seq 0, ack 1, win 0, length 0
12:29:48.124214 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.666 > 10.0.0.21.441: Flags [S], cksum 0x7767 (correct), seq 0, win 8192, length 0
12:29:48.124222 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.441 > 10.0.0.20.666: Flags [R.], cksum 0x9754 (correct), seq 0, ack 1, win 0, length 0
12:29:48.124794 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.666 > 10.0.0.21.442: Flags [S], cksum 0x7766 (correct), seq 0, win 8192, length 0
12:29:48.124803 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.442 > 10.0.0.20.666: Flags [R.], cksum 0x9753 (correct), seq 0, ack 1, win 0, length 0
12:29:48.125363 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.666 > 10.0.0.21.https: Flags [S], cksum 0x7765 (correct), seq 0, win 8192, length 0
12:29:48.125371 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.https > 10.0.0.20.666: Flags [R.], cksum 0x9752 (correct), seq 0, ack 1, win 0, length 0
12:29:53.290548 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.20 tell 10.0.0.21, length 28
12:29:53.290556 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.20 is-at 00:00:00:aa:00:00 (oui Ethernet), length 28
12:32:19.551172 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.21 tell 10.0.0.20, length 28
12:32:19.551188 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.21 is-at 00:00:00:aa:00:01 (oui Ethernet), length 28
12:32:19.566457 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.ssh: Flags [S], cksum 0x7b90 (correct), seq 0, win 8192, length 0
```

Figure 56 - Scapy Docs Usage Tutorial - SYN Scans - 6

- “ans, unans = sr(IP(dst=["10.0.0.21","10.0.0.22"])/TCP(dport=[22,80,443],flags="S"))” = (Figure 57).
- “ans.make_table(lambda s,r: (s.dst, s.dport, r.sprintf("{TCP:%TCP.flags%}{ICMP:%IP.src% - %ICMP.type%}")))” = (Figure 57).
- “ans.nsummary(lfilter = lambda s,r: r.sprintf("%TCP.flags%") == "SA”) = No Output (Figure 57).
- “ans.summary(lfilter = lambda s,r: r.sprintf("%TCP.flags%") == "SA”,prn=lambda s,r: r.sprintf("%TCP.sport% is open”)) = No Output (Figure 57).
- “ans.filter(lambda s,r: TCP in r and r[TCP].flags&2).make_table(lambda s,r: (s.dst, s.dport, "X”)) = No Output (Figure 57).
- “report_ports("192.168.1.1”,(440,443)) = (Figure 57).

```

>>> ans.make_table(lambda s,r: (s.dst, s.dport, r.sprintf("{TCP:%TCP.flags%}{ICMP:%IP.src% - %ICMP.type%}")))
10.0.0.21 10.0.0.22
22 RA RA
80.0.0.21 RA 10.0.0.21 RA
2001:14:64:2001:15:RA 2001:14:64:2001:15:RA
443 RA RA

>>> ans.nsummary(lfilter = lambda s,r: r.sprintf("%TCP.flags%") == "SA")
>>> ans.summary(lfilter = lambda s,r: r.sprintf("%TCP.flags%") == "SA",prn=lambda s,r: r.sprintf("%TCP.sport% is open"))
>>> ans.filter(lambda s,r: TCP in r and r[TCP].flags&2).make_table(lambda s,r: (s.dst, s.dport, "X"))

>>> report_ports("10.0.0.21", (440, 443))
Begin emission:
Finished sending 4 packets.
*****
Received 5 packets, got 4 answers, remaining 0 packets
'\\begin{tabular}{|r|l|l|}\n\\hline\n\\hline\nn440 & closed & TCP RA \\\\\\n441 & closed & TCP RA \\\\\\n442 & closed & TCP RA \\\\\\nhttps & closed & TCP RA \\\\\\n\\hline\n\\hline\n\\end{tabular}\\n'
>>

```

Figure 57 - Scapy Docs Usage Tutorial - SYN Scans - 10

TCP Traceroute

A classic Scapy TCP Traceroute already has “some other high-level functions are already coded” into it (Figure 58, 61 & 62). Scapy can “also use the GeoIP2 module, in combination with matplotlib and cartopy to generate fancy graphics” (Scapy, 2022). Figure 62 uses ‘the traceroute_map() function to print the graphic’ however this did not print anything (Scapy, 2022). This requires that the “geoip2 module, its database” and “also the cartopy module” (Scapy, 2022).

- “ans, unans = sr(IP(dst=target, ttl=(4,25),id=RandShort())/TCP(flags=0x2))” = (Figure 58).

```
>>> ans, unans = sr(IP(dst="10.0.0.21", ttl=(4,25), id=RandShort())/TCP(flags=0x2))
10.0.0.20.ftp-data > 10.0.0.21.http: Flags [S], ...
Begin emission:
Finished sending 22 packets.
*****
Received 22 packets, got 22 answers, remaining 0 packets
>>>
```

Figure 58 - Scapy Docs Usage Tutorial - TCP Traceroute - 1

TCPDump running on PC3-L was able to sniff some traffic from the previous commands. This is again very similar to previous results as there is ICMP packets, payloads, who-has/is-at replays and requests (Figure 59 & 60).

- “tcpdump -v”

```

root@PC3-L:/tmp/pycore.1/PC3-L.conf# tcpdump -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:37:34.602642 IP6 (hlen 255, next-header ICMPv6 (58) payload length: 16) fe80::200:ff:feaa:1 > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16
    source link-address option (1), length 8 (1): 00:00:00:aa:00:01
12:37:50.966424 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.21 tell 10.0.0.20, length 28
12:37:50.966457 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.21 is-at 00:00:00:aa:00:01 (oui Ethernet), length 28
12:37:50.985686 IP6 (hlen 255, next-header ICMPv6 (58) payload length: 16) fe80::a4e9:a5ff:feb1:85e3 > ip6-allrouters: [icmp6 sum ok] ICMP6, router solicitation, length 16
    source link-address option (1), length 8 (1): a6:e9:a5:b1:85:e3
12:37:50.986187 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.440: Flags [S], cksm 0x79ee (correct), seq 0, win 8192, length 0
12:37:50.986218 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.440 > 10.0.0.20.ftp-data: Flags [R.], cksm 0x99db (correct), seq 0, ack 1, win 0, length 0
12:37:50.987020 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.441: Flags [S], cksm 0x79ed (correct), seq 0, win 8192, length 0
12:37:50.987046 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.441 > 10.0.0.20.ftp-data: Flags [R.], cksm 0x99da (correct), seq 0, ack 1, win 0, length 0
12:37:50.987694 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.442: Flags [S], cksm 0x79ec (correct), seq 0, win 8192, length 0
12:37:50.987712 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.442 > 10.0.0.20.ftp-data: Flags [R.], cksm 0x99d9 (correct), seq 0, ack 1, win 0, length 0
12:37:50.988244 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.https: Flags [S], cksm 0x79eb (correct), seq 0, win 8192, length 0
12:37:50.988263 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.https > 10.0.0.20.ftp-data: Flags [R.], cksm 0x99d8 (correct), seq 0, ack 1, win 0, length 0
12:37:56.106477 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.20 tell 10.0.0.21, length 28
12:37:56.106498 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.20 is-at 00:00:00:aa:00:00 (oui Ethernet), length 28
12:39:04.878966 IP (tos 0x0, ttl 4, id 21089, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.http: Flags [S], cksm 0x7b56 (correct), seq 0, win 8192, length 0

```

Figure 59 - Scapy Docs Usage Tutorial - TCP Traceroute - 2

```

12:39:04.882042 IP (tos 0x0, ttl 9, id 13698, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.http: Flags [S], cksm 0x7b56 (correct), seq 0, win 8192, length 0
12:39:04.882060 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.http > 10.0.0.20.ftp-data: Flags [R.], cksm 0x9b43 (correct), seq 0, ack 1, win 0, length 0
12:39:04.882633 IP (tos 0x0, ttl 10, id 5858, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.http: Flags [S], cksm 0x7b56 (correct), seq 0, win 8192, length 0
12:39:04.882660 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.http > 10.0.0.20.ftp-data: Flags [R.], cksm 0x9b43 (correct), seq 0, ack 1, win 0, length 0
12:39:04.883214 IP (tos 0x0, ttl 11, id 46467, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.http: Flags [S], cksm 0x7b56 (correct), seq 0, win 8192, length 0
12:39:04.883233 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.http > 10.0.0.20.ftp-data: Flags [R.], cksm 0x9b43 (correct), seq 0, ack 1, win 0, length 0
12:39:04.883838 IP (tos 0x0, ttl 12, id 6152, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.http: Flags [S], cksm 0x7b56 (correct), seq 0, win 8192, length 0
12:39:04.883857 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.http > 10.0.0.20.ftp-data: Flags [R.], cksm 0x9b43 (correct), seq 0, ack 1, win 0, length 0
12:39:04.884434 IP (tos 0x0, ttl 13, id 51915, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.http: Flags [S], cksm 0x7b56 (correct), seq 0, win 8192, length 0
12:39:04.884452 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.http > 10.0.0.20.ftp-data: Flags [R.], cksm 0x9b43 (correct), seq 0, ack 1, win 0, length 0
12:39:04.885075 IP (tos 0x0, ttl 14, id 9437, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.http: Flags [S], cksm 0x7b56 (correct), seq 0, win 8192, length 0
12:39:04.885097 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.http > 10.0.0.20.ftp-data: Flags [R.], cksm 0x9b43 (correct), seq 0, ack 1, win 0, length 0
12:39:04.886581 IP (tos 0x0, ttl 15, id 47148, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.ftp-data > 10.0.0.21.http: Flags [S], cksm 0x7b56 (correct), seq 0, win 8192, length 0

```

Figure 60 - Scapy Docs Usage Tutorial - TCP Traceroute - 3

- “for snd,rcv in ans: print snd.ttl, rcv.src, isinstance(rcv.payload, TCP)”

```
>>> for snd,rcv in ans: print(snd.ttl, rcv.src, isinstance(rcv.payload, TCP))
4 10.0.0.21 True
5 10.0.0.21 True
6 10.0.0.21 True
7 10.0.0.21 True
8 10.0.0.21 True
PC1 PC2
9 10.0.0.21 True
root@PC3-L:/tmp/pycore.1/PC3-L.conf# tcpdump -v
10 10.0.0.21 True
tcpdump: listening on eth0, link-type EN10MB (Ethernet)
11 10.0.0.21 True
es
12 10.0.0.21 True
-
13 10.0.0.21 True
14 10.0.0.21 True
15 10.0.0.21 True
16 10.0.0.21 True
17 10.0.0.21 True
18 10.0.0.21 True
19 10.0.0.21 True
20 10.0.0.21 True
21 10.0.0.21 True
22 10.0.0.21 True
23 10.0.0.21 True
24 10.0.0.21 True
25 10.0.0.21 True
>>>
```

Figure 61 - Scapy Docs Usage Tutorial - TCP Traceroute – 4

- “lsc()” = (Figure 62).
- “conf.geoip_city = "path/to/GeoLite2-City.mmdb"" ← Why command didn't work, no specified path... = (Figure 62).
- “a = traceroute(["www.google.co.uk", "www.secdev.org"], verbose=0)” = (Figure 62).
- “a.world_trace()” = (Figure 62).

```
>>> lsc()
IPID_count 10.0.0.22/24 : Identify IP id values classes in a list of packets
arpcahepoison 10.0.0.22/24 : Poison target's cache with (your MAC,victim's IP) couple
arping_up PC3 : Send ARP who-has requests to determine which hosts are up
arpleak : Exploit ARP leak flaws, like NetBSD-SA2017-002.
bind_layers : Bind 2 layers on some specific fields' values.
bridge_and_sniff 10.0.0.22/24 : Forward traffic between interfaces if1 and if2, sniff and return
hexdump 10.0.0.22/24 2001::15/64 : Build a per byte hexadecimal representation
computeNIGroupAddr : Compute the NI group Address. Can take a FQDN as input parameter
corrupt_bits :
corrupt_bytes :
defrag PC2 : defrag(plist) -> ([not fragmented], [defragmented], ...
defragment : defragment(plist) -> plist defragmented as much as possible), capture
dhcp_request : Send a DHCP discover request and return the answer
dyndns_add : Send a DNS add message to a nameserver for "name" to have a new "rdat
a"
dyndns_del : Send a DNS delete message to a nameserver for "name"
etherleak : Exploit Etherleak flaw
explore : Function used to discover the Scapy layers and protocols.
fletcher16_checkbytes: Calculates the Fletcher-16 checkbytes returned as 2 byte binary-stri
ng.
>>> conf.geoip_city = "path/to/GeoLite2-City.mmdb"
>>> a = traceroute(["10.0.0.21", "10.0.0.22"], verbose=0)
>>> a.world_trace()
```

Figure 62 - Scapy Docs Usage Tutorial - TCP Traceroute – 5

Even though no graphic was generated because I forgot to specify a path and didn't notice for some reason, TCPDump actually picked up traffic. These again were the same as previous sections.

- “tcpdump -v”

```
root@PC3-L:/tmp/pycore.1/PC3-L.conf# tcpdump -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:41:23.978757 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) fe80::200:ff:fea:2 > ip6-allrouters: [icmp6
sum ok] ICMP6, router solicitation, length 16
    source link-address option (1), length 8 (1): 00:00:00:aa:00:02
12:43:35.050205 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) fe80::200:ff:fea:0 > ip6-allrouters: [icmp6
sum ok] ICMP6, router solicitation, length 16
    source link-address option (1), length 8 (1): 00:00:00:aa:00:00
12:43:47.198548 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.21 tell 10.0.0.20, length 28
12:43:47.198566 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.21 is-at 00:00:00:aa:00:01 (oui Ethernet), length 28
12:43:47.214961 IP (tos 0x0, ttl 1, id 55724, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.28579 > 10.0.0.21.http: Flags [S], cksum 0xcdcb4 (correct), seq 1667160755, win 8192, length 0
12:43:47.214981 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.http > 10.0.0.20.28579: Flags [R.], cksum 0xed1 (correct), seq 0, ack 1667160756, win 0, length 0
12:43:47.215797 IP (tos 0x0, ttl 2, id 41360, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.46025 > 10.0.0.21.http: Flags [S], cksum 0x336f (correct), seq 4156202102, win 8192, length 0
12:43:47.215808 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.21.http > 10.0.0.20.46025: Flags [R.], cksum 0x535c (correct), seq 0, ack 4156202103, win 0, length 0
```

Figure 63 - Scapy Docs Usage Tutorial - TCP Traceroute - 6

```
12:43:47.255440 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.22 tell 10.0.0.20, length 28
12:43:47.255458 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.22 is-at 00:00:00:aa:00:02 (oui Ethernet), length 28
12:43:47.270717 IP (tos 0x0, ttl 1, id 5537, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.51977 > 10.0.0.22.http: Flags [S], cksum 0x98fc (correct), seq 2513142167, win 8192, length 0
12:43:47.270735 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.22.http > 10.0.0.20.51977: Flags [R.], cksum 0xb8e9 (correct), seq 0, ack 2513142168, win 0, length 0
12:43:47.271456 IP (tos 0x0, ttl 2, id 32753, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.1950 > 10.0.0.22.http: Flags [S], cksum 0x7583 (correct), seq 1517462485, win 8192, length 0
12:43:47.271465 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.22.http > 10.0.0.20.1950: Flags [R.], cksum 0x9570 (correct), seq 0, ack 1517462486, win 0, length 0
12:43:47.272070 IP (tos 0x0, ttl 3, id 15468, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.41285 > 10.0.0.22.http: Flags [S], cksum 0x581f (correct), seq 1569072254, win 8192, length 0
12:43:47.272077 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.22.http > 10.0.0.20.41285: Flags [R.], cksum 0x780c (correct), seq 0, ack 1569072255, win 0, length 0
12:43:47.272694 IP (tos 0x0, ttl 4, id 62746, offset 0, flags [none], proto TCP (6), length 40)
    10.0.0.20.41708 > 10.0.0.22.http: Flags [S], cksum 0xb07f (correct), seq 1013640082, win 8192, length 0
12:43:47.272702 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
    10.0.0.22.http > 10.0.0.20.41708: Flags [R.], cksum 0xd06c (correct), seq 0, ack 1013640083, win 0, length 0
```

Figure 64 - Scapy Docs Usage Tutorial - TCP Traceroute - 7

Scapy Python Scripts

In this section, we will be taking scripts from the previous sections, and using them to create two different Python scripts that we can use to run on PC1 and target PC2, and then running TCPDump on PC3-L to attempt to sniff the traffic.

Setting-up .txt for Python Script

The first step is to create two .py files using the text editor. The first step is to open the text editor, select the little down arrow next to the words ‘Plain Text’, search for Python3 in the search bar and double click Python3 when it appears (Figure 65). This sets the language of the document to Python3. Then we can select the save option and name the document. These names must end in .py which makes the text files turn into the Python executable scripts needed by this lab. These were saved in the directory ‘/Desktop/432/py’ (Figure 65). These are named “432py-1.py” & “432py-1.py” (Figure 65).

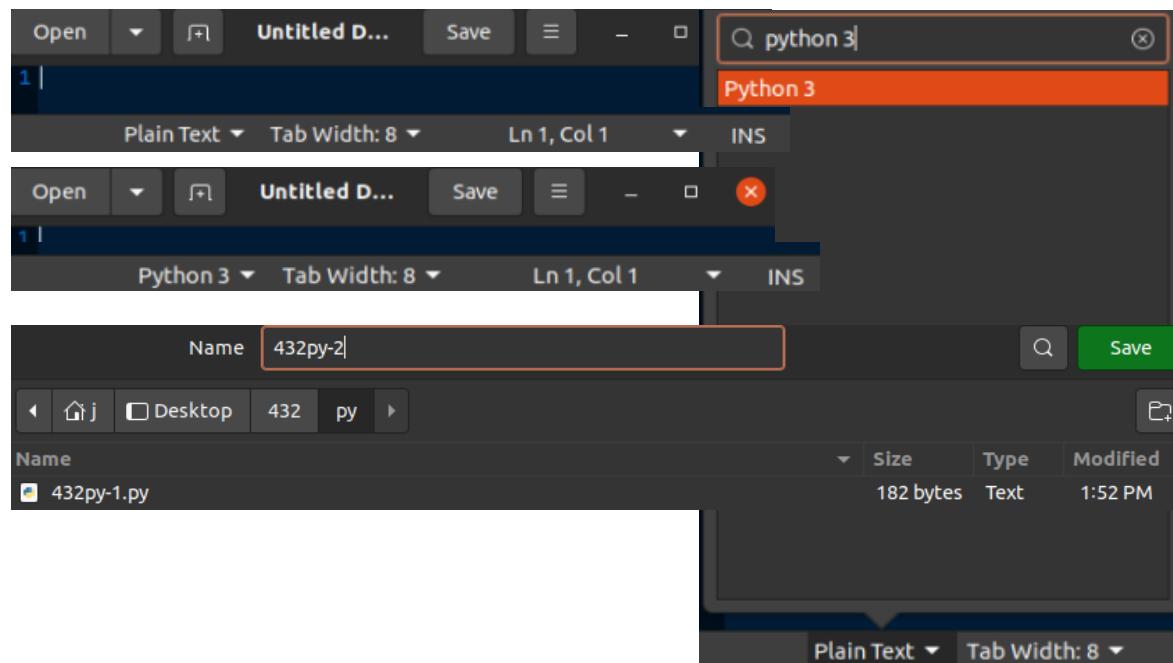


Figure 65 - Setting-up Text File

432py-1.py

The first script uses commands from the following sections, along with the sections, the command(s) used will be listed.

- Sending Packets:
 - o “send(IP(dst="10.0.0.21")/ICMP())”
 - o “sendp(Ether()/IP(dst="10.0.0.21",ttl=(1,4)), iface="eth0")”
 - o “sendp(rdpcap("/home/j/scapy/test/pcaps/pfcp.pcap"))”
- Fuzzing:
 - o “send(IP(dst="target")/fuzz(UDP()/NTP(version=4)),loop=1)”
- Injecting Bytes:
 - o “pkt = IP(len=RawVal(b"NotAnInteger"), src="127.0.0.1")”
 - o “print(bytes(pkt))”

```
432py-1.py ×

1 import scapy
2 from scapy.all import *
3
4 print("#####\nSending Packets\n")
5 time.sleep(1)
6 send(IP(dst="10.0.0.21")/ICMP())
7 time.sleep(0.5)
8 print("----\n")
9 sendp(Ether()/IP(dst="10.0.0.21", ttl=(1, 4)), iface="eth0")
10 print("----\n")
11 time.sleep(0.5)
12 sendp(rdpcap("/home/j/scapy/test/pcaps/pfcp.pcap"))
13 time.sleep(0.5)
14 print("\n----\n\nDone.\n")
15 time.sleep(0.5)
16
17 print("#####\nFuzzing - Use 'Ctrl + C' to Stop Script!\n")
18 time.sleep(1)
19 send(IP(dst="10.0.0.21")/fuzz(UDP() / NTP(version=4)), loop=1)
20 time.sleep(0.5)
21 print("\n----\n\nDone.\n")
22 time.sleep(1)
23
24 print("#####\nInjecting Bytes\n")
25 time.sleep(1)
26 pkt = IP(len=RawVal(b"NotAnInteger"), src="127.0.0.1")
27 time.sleep(0.5)
28 print("----\n")
29 print(bytes(pkt))
30 time.sleep(0.5)
31 print("\n----\n\nDone.\n")
32 time.sleep(1)
33
```

Figure 66 - Python Script - 432py-1.py

432py-2.py

The second script uses commands from the following sections, along with the sections, the command(s) used will be listed.

- Send & Receive Packets: Pts. 1
 - o “p = sr1(IP(dst="10.0.0.21")/ICMP()/"XXXXXXXXXXXX")”
 - o “print(p)”
 - o “s = p.show(dump=True)”
 - o “print(s)”
- Send & Receive Packets: Pts. 2
 - o “sr1(IP(dst="10.0.0.21")/UDP()/DNS(rd=1,qd=DNSQR(qname="10.0.0.20")))”
 - o “an, unan =

sr1(IP(dst="10.0.0.21")/UDP()/DNS(rd=1,qd=DNSQR(qname="10.0.0.20")))”
 - o “print(an, unan)”
 - o “print(an.summary())”
- SYN Scan
 - o “sr1(IP(dst="10.0.0.21")/TCP(dport=80,flags="S"))”
 - o “ans, unans =

sr(IP(dst="10.0.0.21")/TCP(sport=666,dport=(440,443),flags="S"))”
 - o “print(ans, unans)”
 - o “print(ans.summary())”
 - o “print(ans.summary(lambda s,r: r.sprintf("%TCP.sport% \t %TCP.flags%")))”
 - o “report_ports("10.0.0.21", (440,443))”

```

1 import scapy
2 from scapy.all import *
3
4 print("#####\nSend & Revice Packet(s) - Pt.1\n")
5 time.sleep(1)
6 p = sr1(IP(dst="10.0.0.21")/ICMP()/"XXXXXXXXXXXX")
7 time.sleep(0.5)
8 print("----\n")
9 print(p)
10 s = p.show(dump=True)
11 time.sleep(0.5)
12 print("\n----\n")
13 print(s)
14 time.sleep(1)
15 print("----\nDone.\n\n#####\nSend & Revice Packet(s) - Pt. 2\n")
16 time.sleep(0.5)
17 sr1(IP(dst="10.0.0.21")/UDP()/DNS(rd=1,qd=DNSQR(qname="10.0.0.20")))
18 an, unan = sr(IP(dst="10.0.0.21")/TCP(dport=[21,22,23]))
19 time.sleep(0.5)
20 print("----\n")
21 print(an, unan)
22 time.sleep(0.5)
23 print("----\n")
24 print(an.summary())
25 time.sleep(1)
26 print("----\n\nDone.\n\n#####\nSYN Scan\n")
27 time.sleep(0.5)
28 sr1(IP(dst="10.0.0.21")/TCP(dport=80,flags="S"))
29 ans, unans = sr(IP(dst="10.0.0.21")/TCP(sport=666,dport=(440,443),flags="S"))
30 time.sleep(0.5)
31 print("----\n")
32 print(ans, unans)
33 time.sleep(0.5)
34 print("----\n")
35 print(ans.summary())
36 time.sleep(0.5)
37 print("----\n")
38 print(ans.summary( lambda s,r: r.sprintf("%TCP.sport% \t %TCP.flags%") ))
39 time.sleep(0.5)
40 print("----\n")
41 report_ports("10.0.0.21", (440,443))
42 time.sleep(0.5)
43 print("----\n\nDone.\n")
44

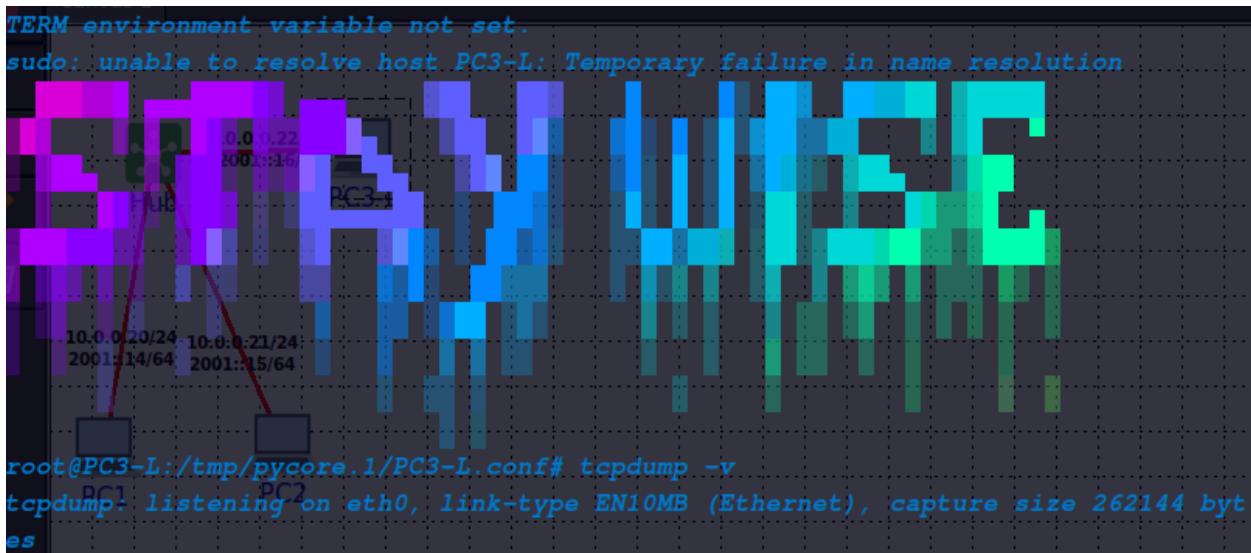
```

432py-2.py
~/Desktop/432/py

Figure 67 - Python Script - 432py-2.py

Running in CORE Network Topology

Now that we have created our two Python scripts using commands from the interactive tutorial on the Scapy Docs, we can now test these scripts out within the CORE network topology and attempt to sniff traffic using TCPDump. The set up to this is similar to other labs, as we will be communicating between nodes one and two (PC1 & PC2) and run TCPDump on node three (PC3-L). The first step we will take is starting TCPDump on PC3-L using the command “tcpdump -v” (Figure 68).



```
TERM environment variable not set.
sudo: unable to resolve host PC3-L: Temporary failure in name resolution
root@PC3-L:/tmp/pycore.1/PC3-L.conf# tcpdump -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

Figure 68 - Starting TCPDump in PC3-L

432py-1.py

Now that TCPDump is running on PC3-L, we can run the first Python script, 432py-1.py, by use of the PC1 terminal. After opening the terminal, I needed to change directories to the folder created previously ‘py’ which holds our Python scripts, then we are able to run the script (Figure 69 - 70). The output for this script is organized by section headers (sending packets, fuzzing, etc.), using lines like “print(“---”)” and blank lines like “print(“ ”)” (Figure 66). The output is also very similar to the outputs seen in the previous sections when using commands from the interactive tutorial. This is because they are the same commands and will provide the same output, this also applies for any traffic captured on PC3-L with TCPDump. The commands used to run the Python script are listed below.

- “cd /home/j/Desktop/432/py”
 - “sudo python3 432py-1.py”

```
root@PC1:/tmp/pycore.1/PC1.conf# cd /home/j/Desktop/432/pyve Timestamp:  
root@PC1:/home/j/Desktop/432/py# sudo python3 432py-1.PYve Timestamp:  
sudo: unable to resolve host PC1: Temporary failure in name resolution  
#####  
Sending Packets  
. .  
Sent 1 packets.  
----  
...  
Sent 4 packets.  
----  
.....  
.....  
.....  
Sent 200 packets.
```

Figure 69 - Running 432py-1.py - 1

```

#####
#0.22/24
#2001:16/64
Fuzzing - Use 'Ctrl + C' to Stop Script!
Hub: PC3-1

10.0.0.20/24 10.0.0.21/24
2001:14/64 2001:15/64

PC1 PC2 ^C

Sent 550 packets.

-----
Done.

#####
Injecting Bytes

-----

b'H\x00NotAnInt\x0f\xb3er\x00\x01\x00\x00@\x00\x00\x00\x7f\x00\x00\x01\x7f
\x00\x00\x01\x00\x00'

-----
Done.

```

Figure 70 - Running 432py-1.py - 2

432py-2.py

After running the previous command, I closed the terminal for PC1 and re-opened it so I could run the second Python script we created, 432py-2.py (Figure 71 – 74). This second Python script has a longer output because it has more commands. The same rules from the prior script apply here. After the terminal was opened, I changed directories to the ‘py’ folder which holds our Python scripts, then I ran the script (Figure 69 - 70). The output for this script is organized by section headers (send & receive packets pt.1 & pt2, etc.), using lines like “print(“----”)” and

blank lines like “print(“ ”)” (Figure 71 - 74). The output is also very similar to the outputs seen in the previous sections when using commands from the interactive tutorial. This is because they are the same commands and will provide the same output, this also applies for any traffic captured on PC3-L with TCPDump. The commands used to run the Python script are listed below.

- “cd /home/j/Desktop/432/py”
- “sudo python3 432py-2.py”

```
root@PC1:/tmp/pycore.1/PC1.conf# cd /home/j/Desktop/432/py
root@PC1:/home/j/Desktop/432/py# sudo python3 432py-2.py
sudo: unable to resolve host PC1: Temporary failure in name resolution
#####
Send & Receive Packet(s) - Pt.1
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
WARNING: Calling str(pkt) on Python 3 makes no sense!
b"E\x00\x00\xda\x00\x00@\x01\x8c>\n\x00\x00\x15\x00\x00\x14\x00\xeeE\x00\x00\x00XXXXXXXXXX"
----
```

Figure 71 - Running 432py-2.py - 1

Canvas 1

```
###[ IP ]### 10.0.0.22/24
version = 4
ihl = 5
tos = 0x0
len = 39
id = 55919
flags = 10.0.0.20/24 10.0.0.21/24
          2001:14/64 2001:15/64
frag = 0
ttl = 64
proto = icmp
chksum = 0x8c3e PC2
src = 10.0.0.21
dst = 10.0.0.20
options \
###[ ICMP ]###
    type = echo-reply
    code = 0
    checksum = 0xee45
    id = 0x0
    seq = 0x0
    unused =
###[ Raw ]###
    load = 'XXXXXXXXXXXX'
----
```

Send 6 Device Packet(s) - Pt. 2

```
Begin emission: 10.0.0.22/24 2001:16/64
Finished sending 1 packets. PC3-L
* Hub
Received 1 packets, got 1 answers, remaining 0 packets
Begin emission:
Finished sending 3 packets.
*** 10.0.0.20/24 10.0.0.21/24
          2001:14/64 2001:15/64
Received 3 packets, got 3 answers, remaining 0 packets
-----
```

<Results: TCP:3 UDP:0 ICMP:0 Other:0> <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>

```
IP / TCP 10.0.0.20:ftp-data > 10.0.0.21:ftp S ==> IP / TCP 10.0.0.21:ftp > 10.0.0.20:ftp-data RA
IP / TCP 10.0.0.20:ftp-data > 10.0.0.21:ssh S ==> IP / TCP 10.0.0.21:ssh > 10.0.0.20:ftp-data RA
IP / TCP 10.0.0.20:ftp-data > 10.0.0.21:telnet S ==> IP / TCP 10.0.0.21:telnet > 10.0.0.20:ftp-data RA
None
```

Done.

Figure 72 - Running 432py-2.py - 2

SYN Scan Canvas 1

```
Begin emission: 10.0.0.22/24 2001:16/64
Finished sending 1 packets. PC3-L
* Hub
Received 1 packets, got 1 answers, remaining 0 packets
Begin emission:
Finished sending 4 packets.
*** 10.0.0.20/24 10.0.0.21/24
          2001:14/64 2001:15/64
Received 4 packets, got 4 answers, remaining 0 packets
-----
```

<Results: TCP:4 UDP:0 ICMP:0 Other:0> <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>

```
IP / TCP 10.0.0.20:666 > 10.0.0.21:440 S ==> IP / TCP 10.0.0.21:440 > 10.0.0.20:666 RA
IP / TCP 10.0.0.20:666 > 10.0.0.21:441 S ==> IP / TCP 10.0.0.21:441 > 10.0.0.20:666 RA
IP / TCP 10.0.0.20:666 > 10.0.0.21:442 S ==> IP / TCP 10.0.0.21:442 > 10.0.0.20:666 RA
IP / TCP 10.0.0.20:666 > 10.0.0.21:https S ==> IP / TCP 10.0.0.21:https > 10.0.0.20:666 RA
None
```

Figure 73 - Running 432py-2.py - 3

```

440
441
442
https
None
-----
Begin emission:
Finished sending 4 packets.
*****
Received 4 packets, got 4 answers, remaining 0 packets
-----
Done.

root@PC1:/home/j/Desktop/432/py# sudo python3 432py-2.py

```

Figure 74 - Running 432py-2.py - 4

Scapy Results

After running each script, I gathered the TCPDump capture by taking screenshots and reviewed the data. This section will be the data which was collected. Above each Figure will be a description of what it contains.

432py-1.py

Let's first look at the traffic sniffed when running the first Python script, 432py-1.py (Figures 75 -77). After running the command “tcpdump -v” in the terminal of PC3-L, and then running the first script, we can see that some data has been gathered (Figure 75). This mainly consists of the sending and receiving IP Addresses, ARP Ethernet who-has requests and is-at reply packets. There is also ICMP and ICMP echo reply packets. All of these were seen earlier in the interactive tutorial section.

```
root@PC3-L:/tmp/pycore.1/PC3-L.conf# tcpdump -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
15:18:06.350530 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.21 tell 10.0.0.20, length 28
15:18:06.350550 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.21 is-at 00:00:00:aa:00:01 (oui Ethernet), length 28
15:18:06.370210 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto ICMP (1), length 28)
  10.0.0.20 > 10.0.0.21: ICMP echo request, id 0, seq 0, length 8
15:18:06.370231 IP (tos 0x0, ttl 64, id 44880, offset 0, flags [none], proto ICMP (1), length 28)
  10.0.0.21 > 10.0.0.20: ICMP echo reply, id 0, seq 0, length 8
15:18:06.903046 IP (tos 0x0, ttl 1, id 1, offset 0, flags [none], proto Options (0), length 20)
  10.0.0.20 > 10.0.0.21: hopopt 0
15:18:06.903068 IP (tos 0xc0, ttl 64, id 44980, offset 0, flags [none], proto ICMP (1), length 48)
  10.0.0.21 > 10.0.0.20: ICMP 10.0.0.21 protocol 0 unreachable, length 28
    IP (tos 0x0, ttl 1, id 1, offset 0, flags [none], proto Options (0), length 20)
    10.0.0.20 > 10.0.0.21: hopopt 0
15:18:06.903451 IP (tos 0x0, ttl 2, id 1, offset 0, flags [none], proto Options (0), length 20)
  10.0.0.20 > 10.0.0.21: hopopt 0
15:18:06.903461 IP (tos 0xc0, ttl 64, id 44981, offset 0, flags [none], proto ICMP (1), length 48)
  10.0.0.21 > 10.0.0.20: ICMP 10.0.0.21 protocol 0 unreachable, length 28
    IP (tos 0x0, ttl 2, id 1, offset 0, flags [none], proto Options (0), length 20)
    10.0.0.20 > 10.0.0.21: hopopt 0
```

Figure 75 - Results from 432py-1.py - 1

After scrolling down, I saw again, unknown packets which were sending and receiving data. This consists of offsets, what looks like the headers to a file, then the plain text view of the ‘message’ (Figure 76).

```
15:18:07.483354 00:00:40:11:7c:80 (oui Unknown) > 45:00:00:6a:00:00 (oui Unknown), ethertype Unknown (0x7f00), length 10
6:
 0x0000: 0001 7f00 0002 2265 2265 0056 e792 2139 ..... "e"e.V..!9
 0x0010: 004a 0000 0000 ffff ffff ff00 0085 .J..... .
 0x0020: 0013 0d36 6856 656f 0b79 2fd7 30a6 a32e ..6hVeo.y/.0...
 0x0030: 92d7 3b1c 4700 0f00 2300 8e00 0101 0013 ..;.G...#...
 0x0040: 0001 4b00 7c00 0100 007c 0001 ff00 2600 ..K.|....|....6.
 0x0050: 0b03 0008 396f 1d03 8198 2306 ..... 9o....#
15:18:07.483447 00:00:40:11:7c:b6 (oui Unknown) > 45:00:00:34:00:00 (oui Unknown), ethertype Unknown (0x7f00), length 52
:
 0x0000: 0001 7f00 0002 2265 2265 0020 9b2f 2139 ..... "e"e.../!9
 0x0010: 0014 0000 0000 0000 0000 0000 0060 ..... .
 0x0020: 0004 0000 0000 ..... python 3.7.0 |.....
15:18:07.483524 00:00:40:11:7c:5e (oui Unknown) > 45:00:00:8c:00:00 (oui Unknown), ethertype Unknown (0x7f00), length 14
0:
 0x0000: 0001 7f00 0002 2265 2265 0078 f7e3 2103 ..... "e"e.x..!.
 0x0010: 006c 0000 0000 0006 ffff ff00 0092 ..l.....
 0x0020: 0004 0000 0001 0072 0005 0100 0000 1400 ..... x.....
 0x0030: 2600 0a00 0007 b652 9532 3cbd d200 6700 ..6...R.2<...g.
 0x0040: 0502 cb26 5caa 0017 0011 0500 0009 7377 ..&\.....sw
 0x0050: b587 7297 b0ac 2100 0000 0000 2b00 02bd ..x...!....+...
 0x0060: 0200 8d00 1903 0733 3938 3235 3530 0f32 ..... 3982550.2
 0x0070: 3838 3530 3337 3337 3533 3230 3538 88503737532058
15:18:07.483604 00:00:40:11:7c:97 (oui Unknown) > 45:00:00:53:00:00 (oui Unknown), ethertype Unknown (0x7f00), length 83
:
 0x0000: 0001 7f00 0002 2265 2265 003f 0c2e 2139 ..... "e"e.z..!9
 0x0010: 0033 0000 0000 ffff ffff ffff ff00 008a ..z.....
 0x0020: 0004 ffff ffff 0017 000b 0e00 0000 0000 ..... .
 0x0030: 0002 0000 2481 f400 0c48 f983 50a3 f4fa ..$...H..P...
 0x0040: 9aa6 d5c1 ad ..
```

Figure 76 - Results from 432py-1.py - 2

Here is the NTPv4 packet capture again. This has a lot of valuable information which splits into two sections: Server and Root (Figure 77). The Server section contains: leap indicator, stratum, poll, and precision (Figure 77). The root section contains: delay, root dispersion, and reference ID (Figure 77). Below this, timestamp information is displayed.

```

15:18:09.543394 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 76)
  10.0.0.20.ntp > 10.0.0.21.ntp: NTPv4, length 48
    Server, Leap indicator: (0), Stratum 166 (reserved), poll 71 (128s), precision 90
    Root Delay: 23167.646148, Root dispersion: 55067.228424, Reference-ID: 58.228.105.40
      Reference Timestamp: 1871022091.256863930 (2095/05/23 10:49:47)
      Originator Timestamp: 3858779889.543334960 (2022/04/12 15:18:09)
      Receive Timestamp: 4046147814.853152172 (2028/03/20 05:56:54)
      Transmit Timestamp: 3858779889.543344497 (2022/04/12 15:18:09)
      Originator - Receive Timestamp: +187367925.309817211
      Originator - Transmit Timestamp: +0.000009536
15:18:09.545155 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 76)
  10.0.0.20.ntp > 10.0.0.21.ntp: NTPv4, length 48
    Reserved, Leap indicator: -1s (128), Stratum 45 (reserved), poll 245 (2097152s), precision -22
    Root Delay: 15408.071548, Root dispersion: 48759.203903, Reference-ID: 185.6.132.77
      Reference Timestamp: 3737516310.227223934 (2018/06/09 02:58:30)
      Originator Timestamp: 3858779889.545086860 (2022/04/12 15:18:09)
      Receive Timestamp: 3185635590.378312874 (2000/12/12 13:46:30)
      Transmit Timestamp: 3858779889.545096874 (2022/04/12 15:18:09)
      Originator - Receive Timestamp: -673144299.166773986
      Originator - Transmit Timestamp: +0.000010013
15:18:09.546768 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 76)
  10.0.0.20.ntp > 10.0.0.21.ntp: NTPv4, length 48
    Broadcast, Leap indicator: (0), Stratum 77 (reserved), poll 156 (268435456s), precision 76
    Root Delay: 23264.552398, Root dispersion: 45088.580871, Reference-ID: 50.67.239.123
      Reference Timestamp: 1233710462.695518067 (2075/03/13 03:49:18)
      Originator Timestamp: 3858779889.546685218 (2022/04/12 15:18:09)
      Receive Timestamp: 2073580511.097394125 (2101/10/23 21:03:27)
      Transmit Timestamp: 3858779889.546694755 (2022/04/12 15:18:09)
      Originator - Receive Timestamp: -1785199378.449291093
      Originator - Transmit Timestamp: +0.000009536

```

Figure 77 - Results from 432py-1.py - 3

432py-2.py

Now we can review the sniffed traffic after running our second Python script 432py-2.py (Figure 78 – 79). For the length of this Python script, the TCPDump capture surprisingly didn't sniff a lot of traffic. I preformed this on three occasions, the first was a test run. The second, I ran the script three times and still only this amount of data was collected, which I found odd. This data output is exactly the same as the interactive tutorial section. It contains the ARP Ethernet who-has request and is-at reply packets, ICMP and ICMP echo request packets, and some packets have FTP and checksum data (Figures 78 -79).

```
root@PC3-L:/tmp/pycore.1/PC3-L.conf# tcpdump -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
15:20:12.203621 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.21 tell 10.0.0.20, length 28
15:20:12.203654 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.21 is-at 00:00:00:aa:00:01 (oui Ethernet), length 28
15:20:12.222099 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto ICMP (1), length 39)
  10.0.0.20 > 10.0.0.21: ICMP echo request, id 0, seq 0, length 19
15:20:12.222135 IP (tos 0x0, ttl 64, id 55919, offset 0, flags [none], proto ICMP (1), length 39)
  10.0.0.21 > 10.0.0.20: ICMP echo reply, id 0, seq 0, length 19
15:20:14.763119 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 55)
  Fin 10.0.0.20.domain > 10.0.0.21.domain: 0+ A? 10.0.0.20. (27)
15:20:14.763152 IP (tos 0xc0, ttl 64, id 56035, offset 0, flags [none], proto ICMP (1), length 83)
  Recv 10.0.0.21 > 10.0.0.20: ICMP 10.0.0.21 udp port domain unreachable, length 63
    IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 55)
    10.0.0.20.domain > 10.0.0.21.domain: 0+ A? 10.0.0.20. (27)
15:20:14.795573 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
  Done 10.0.0.20.ftp-data > 10.0.0.21.ftp: Flags [S], cksum 0x7b91 (correct), seq 0, win 8192, length 0
```

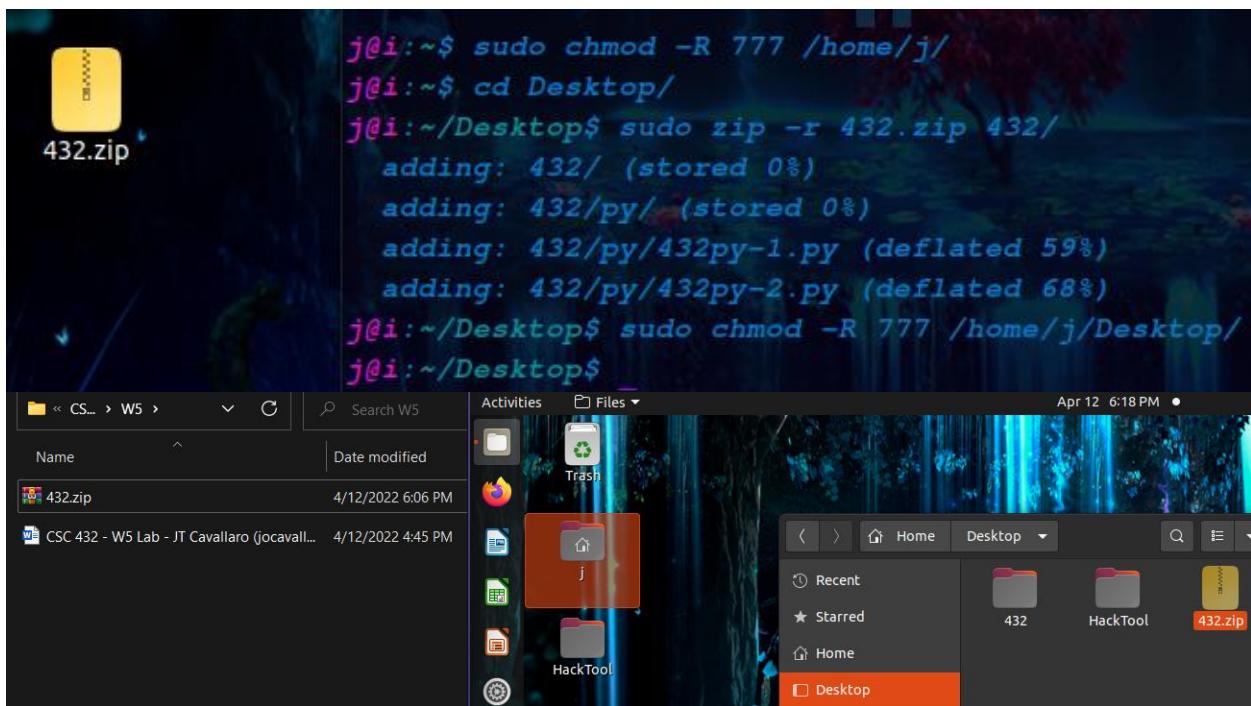
Figure 78 - Results from 432py-1.py - 4

```
15:20:19.425172 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
  10.0.0.20.ftp-data > 10.0.0.21.442: Flags [S], cksum 0x79ec (correct), seq 0, win 8192, length 0
15:20:19.425191 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
  10.0.0.21.442 > 10.0.0.20.ftp-data: Flags [R.], cksum 0x99d9 (correct), seq 0, ack 1, win 0, length 0
15:20:19.425797 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 40)
  10.0.0.20.ftp-data > 10.0.0.21.https: Flags [S], cksum 0x79eb (correct), seq 0, win 8192, length 0
15:20:19.425804 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 40)
  10.0.0.21.https > 10.0.0.20.ftp-data: Flags [R.], cksum 0x99d8 (correct), seq 0, ack 1, win 0, length 0
15:20:19.425864 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.21 tell 10.0.0.20, length 28
15:20:25.382582 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.21 is-at 00:00:00:aa:00:01 (oui Ethernet), length 28
15:20:25.402565 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto ICMP (1), length 39)
  10.0.0.20 > 10.0.0.21: ICMP echo request, id 0, seq 0, length 19
15:20:25.402586 IP (tos 0x0, ttl 64, id 58145, offset 0, flags [none], proto ICMP (1), length 39)
  10.0.0.21 > 10.0.0.20: ICMP echo reply, id 0, seq 0, length 19
15:20:27.939368 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 55)
  Fin 10.0.0.20.domain > 10.0.0.21.domain: 0+ A? 10.0.0.20. (27)
15:20:27.939393 IP (tos 0xc0, ttl 64, id 58398, offset 0, flags [none], proto ICMP (1), length 83)
  Recv 10.0.0.21 > 10.0.0.20: ICMP 10.0.0.21 udp port domain unreachable, length 63
    IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 55)
    10.0.0.20.domain > 10.0.0.21.domain: 0+ A? 10.0.0.20. (27)
```

Figure 79 - Results from 432py-1.py - 5

Wrap-Up

After completing this lab, and closing CORE, I needed to provide a zip file which contains the two Python scripts we created. So do this, I first used the chmod command to change the privileges of the directory ‘/home/j/’ as well all the recursive directories and their contents (Figure 80). I then changed to the Desktop directory and used the zip command to compress the directory ‘/432/’ as well using the recursive flag, ‘-r’, to automatically compress the contents of the other directories within. In this case, the ‘/py/’ directory which contains are two Python scripts 432py-1.py and 432py-2.py (Figure 80). I then again used the chmod again to change the privilege level of the Desktop directory and opened the File Manager and directed to the Desktop folder (Figure 80). This is where the newly created zip file ‘432.zip’ resides, and in doing so, I am able to drag and drop the .zip file from my Ubuntu VM to my host PC for turn in (Figure 80).



The screenshot shows a Linux desktop environment with a terminal window open and a file manager window visible.

Terminal Output:

```
j@i:~$ sudo chmod -R 777 /home/j/
j@i:~$ cd Desktop/
j@i:~/Desktop$ sudo zip -r 432.zip 432/
adding: 432/ (stored 0%)
adding: 432/py/ (stored 0%)
adding: 432/py/432py-1.py (deflated 59%)
adding: 432/py/432py-2.py (deflated 68%)
j@i:~/Desktop$ sudo chmod -R 777 /home/j/Desktop/
j@i:~/Desktop$
```

File Manager:

- The desktop has a file named "432.zip".
- The file manager sidebar shows a folder named "j" which contains "432" and "HackTool".
- The desktop panel shows icons for "432", "HackTool", and "432.zip".
- The status bar at the bottom right indicates the date and time: "Apr 12 6:18 PM".

Figure 80 - Preparing Python Scripts for Turn-in

Conclusion

In the end of this lab, I was able to read through the Python3 Scapy Library Documentation, install Scapy and the Python3 Scapy Library, as well all of the needed dependencies, successfully. Then tested scapy out in the terminal of my Ubuntu VM to verify that everything was working properly. After building a network topology in CORE using three nodes and a hub, I ran the network topology and continued through examples in the docs. After reviewing the results, I created two Python scripts to be ran on the network topology. Using TCPDump, I was able to capture packets created and sent by the scripts and reviewed the results.

References

- CORE, . (2022). *Using the core gui*. core. Retrieved March 20, 2022, from
<https://coreemu.github.io/core/gui.html> & <http://coreemu.github.io/core/install.html> &
<https://github.com/coreemu/core>
- Scapy, . (2022). *Welcome to Scapy's documentation!* Welcome to Scapy's documentation! -
Scapy 2.4.5. documentation. Retrieved April 10, 2022, from
<https://scapy.readthedocs.io/en/latest/> & <https://scapy.readthedocs.io/en/latest/usage.html>
& <https://scapy.readthedocs.io/en/latest/installation.html#installing-scapy-v2-x>
- TCPDump, . (2012). *Ubuntu Manpage: Tcpdump*. Ubuntu Manpage: Tcpdump - dump traffic on
a network. Retrieved April 5, 2022, from
<http://manpages.ubuntu.com/manpages/trusty/man8/tcpdump.8.html>

Appendix A: Tools –

Tool	Version
Google Chrome	99.0.4844.51
Engage	N/A
VMWare Horizon Client	2111
Ubuntu LTS Linux VM	20.04
CORE	8.2.0
Python2 & Python3	2.7.18 & 3.8.10
Scapy	2.4.5rc1.dev228

Appendix B: Host & Virtual Machine Specifications –

Host PC:

- Computer Name: PCJ
- Operating System (OS) Name: Microsoft Windows 11 Home
- OS Version: 10.0.22000 Build 22000
- System Make/Model: Dell Inc. G7 7588 x64
- System Serial Number: N/A
- Time Zone of Examiner Machine: Eastern

Ubuntu VM:

- Computer Name: ubuntu (Before Lab Completion)
- Computer Name: i2 (After Lab Completion)
- Operating System (OS) Name & Version: Ubuntu 20.04 LTS x64
- GNOME Version: 3.36.8
- Kernel: 5.4.0-104-generic
- LSB Version: core-11.1.0ubuntu2-noarch:security-11.1.0ubuntu2-noarch
- Time Zone of Examiner Machine: Eastern