STATISTICAL NETWORK ANALYZER

SAIT ISS 2020 CAPSTONE

Charity Becker-Lewis

Table of Contents

Executive Summary 2
Program Requirements 3
Usage 3
Challenges 10
Security Risks 11
Conclusion 12
References 14

Executive Summary

In the final semester of SAIT's two-year Information Systems Security (ISS) diploma course, students are required to research or develop a project related to systems security and incorporate it with the knowledge they have obtained from the course. Throughout the course, I had come to notice that when analyzing network traffic data, that there was a lack of data visualization and statistical representation. Some network traffic analyzers, such as Wireshark, are limited to text representation and to analyze the data, it had to be done manually or by sorting, which is rather tedious and time consumptive. As such, for the ISS project I decided to develop a python tool that could read network traffic data, statistically analyze the data and produce graphical representations of values. In addition to this, the tool can be run completely in a command-line interface, which allows the program to be run in non-graphical user interface systems. The Statistical Network Analyzer (SNA) tool intends to simplify the process of generating statistical network traffic analysis in a command-line environment with visual and text-based results.

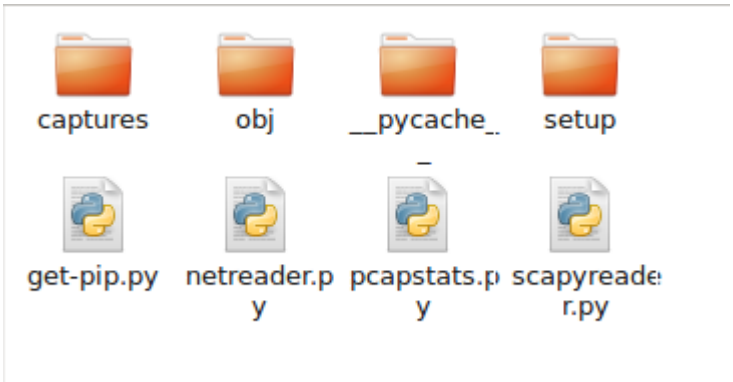
Program Requirements

Supported OS: Debian Linux (Tested and developed on Ubuntu 16.04)
Program Language(s): Python 3.7 and Bash Shell (for setup)
Default modules: collections, curses, datetime, grp, os, pickle, pwd, socket, subprocess, sys, time and traceback
Externally downloaded modules: matplotlib, pandas, prettytable, scapy.all, seaborn, tabulate, and termcolor
Total size on disk: 1.9 MiB (1,957,888 bytes)

Usage

The Statistical Network Analyzer (SNA) tool is comprised of three main scrips (netreader.py, pcapstats.py and scapyreader.py) and three main directories (captures, obj and setup).

netreader.py is the script that starts the setup for reading the network. This is comprised of five main steps, which will be detailed later.



scapyreader.py is the script that records the network traffic and parses the data into a pcap file. This script is called from within netreader.py and cannot be called on its own.

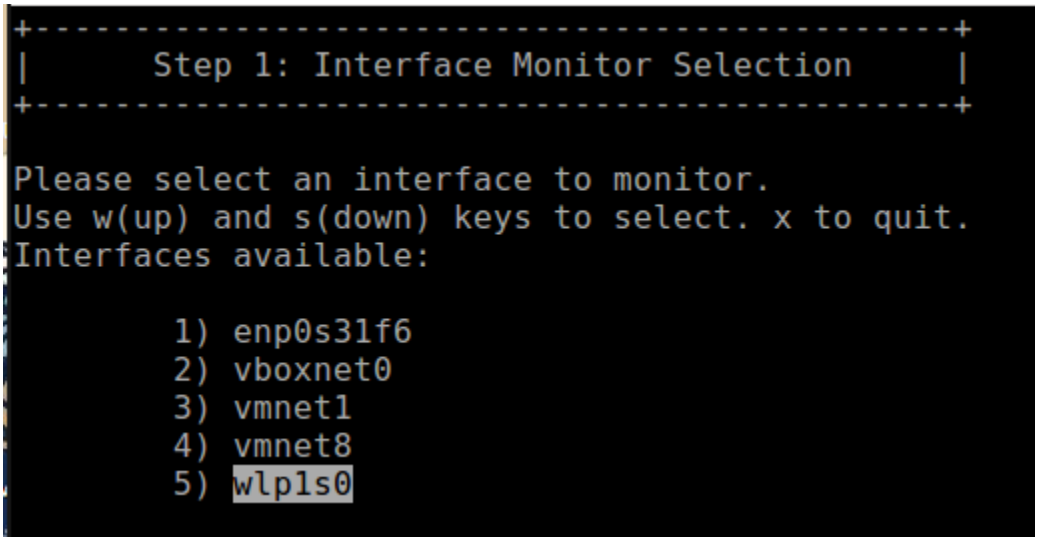
pcapstats.py takes in a pcap file and analyzes the file for statistical data and displays it into tables and png graphs. This script can be optionally run after netreader.py but can also be run on its own.

/captures will by default hold the generated pcap files from netreader.py and the graphs generated from pcapstats.py. /obj holds a pickle object that stores the global variables that contain the settings chosen in netreader.py to share with scapyreader.py. /setup contains a small bash script that will verify and gather any modules that SNA requires to run.

```
SNA$ sudo ./netreader.py
```

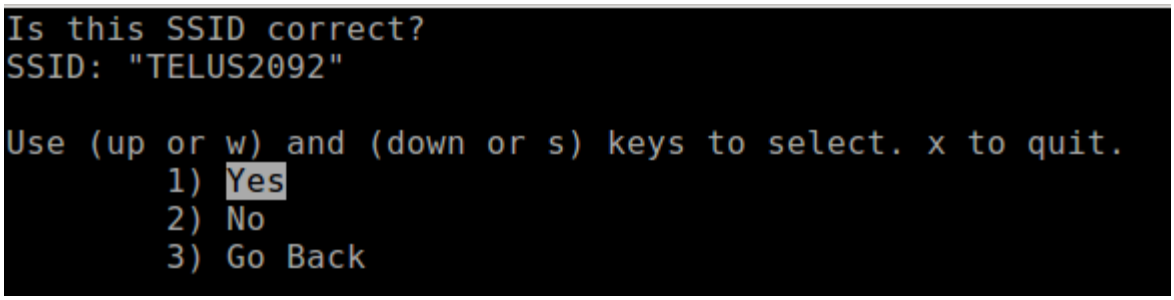
netreader.py is the primary script for the tool and in order to run it requires SuperUser (sudo) permissions. This is in due to that the scapy module used to read network traffic requires higher level permissions to read from the network interface cards, like the software Wireshark. The script will verify that it is running with the appropriate privileges and will quit with an error message if not. netreader.py can be run in two ways: via "sudo ./" or via "sudo python3". Either method will start
the script and no other parameters are required to run.

Once the script starts, the user will be brought to the first page in the setup process: Interface Monitor Selection.

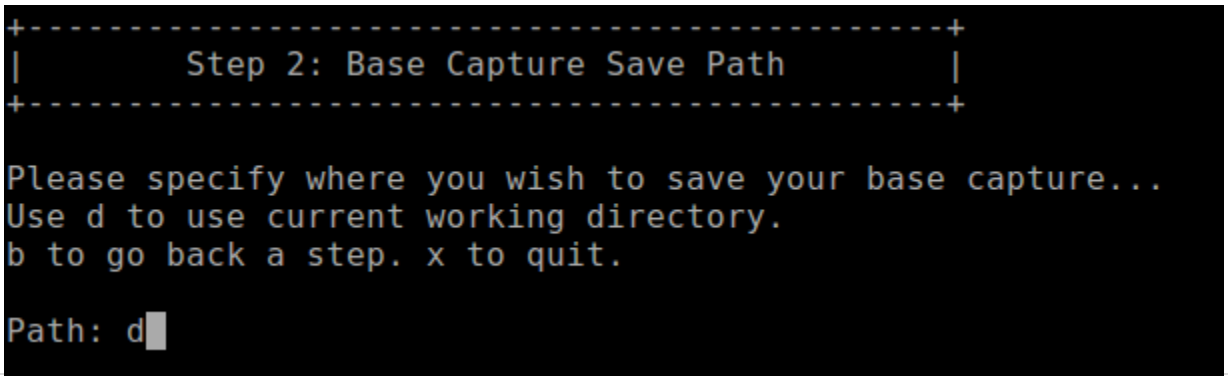


In this screen, the user will be prompted to select which of their network interfaces they wish to run the capture on. The users can interact with the options via the up and down arrows or the w or s keys and hitting enter to select the highlighted option. Throughout the tests, these controls are used, and ^C or the x key can be used to exit the program. The options are dynamically added by internally performing the "ifconfig" and parsing the interfaces that are outputted. This setting is required for the scapy module to determine which interface the module should look for incoming and outcoming traffic.

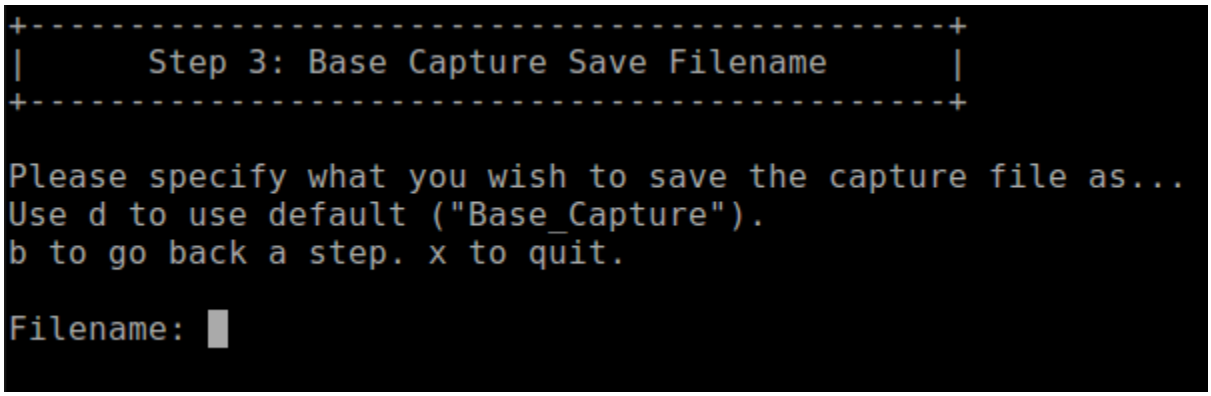
After selecting an interface, the user is prompted to verify the detected SSID of the connected network. This step is used to verify that the user is looking at the intended interface and network.



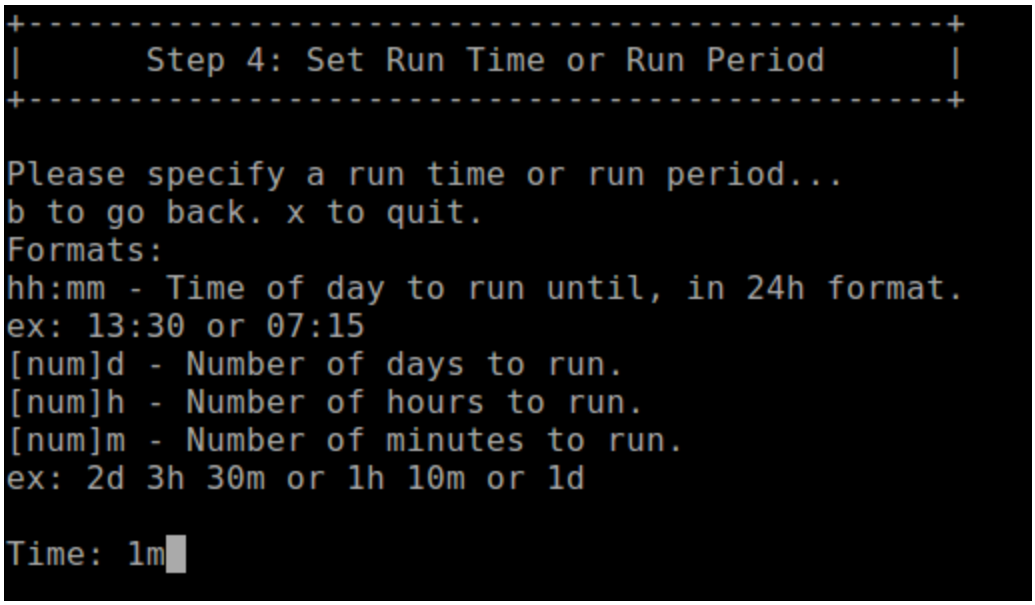
After verifying the SSID, the user is brought to decide what directory they wish to save their capture at. By default, the path will be in the SNA /captures directory.



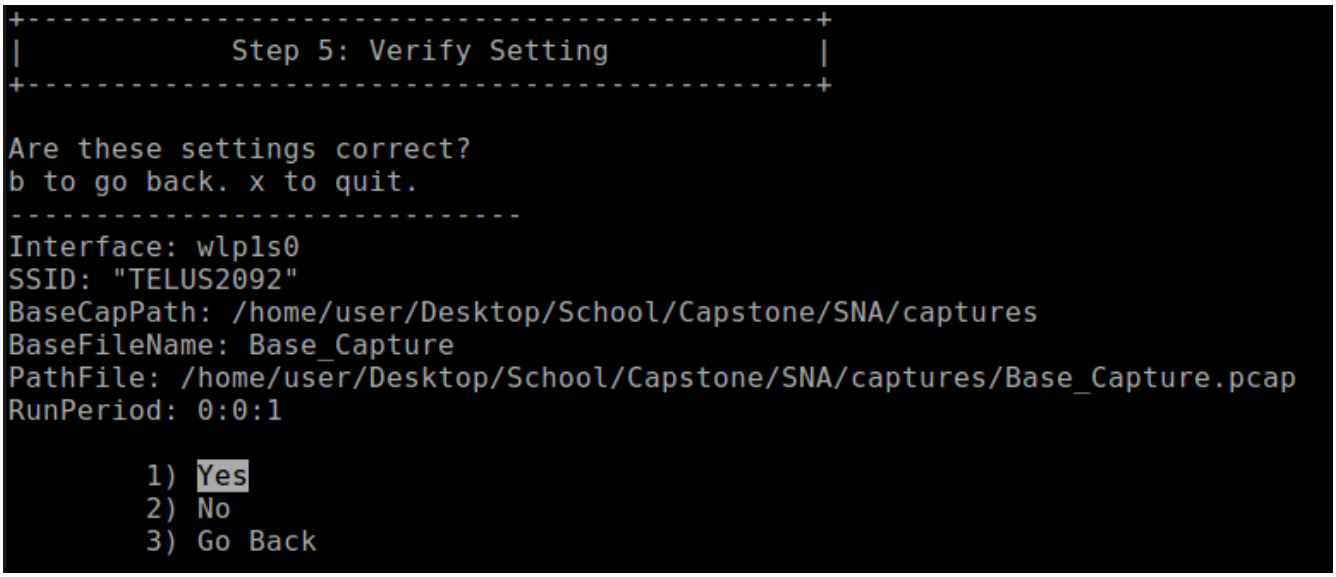
Step 3 allows the user to decide what to name their files as. The user can use the default of "BaseCapture" which would output a pcap file of "BaseCapture.pcap". If in the case that the file already exists, the program will append a number to the name that increments until that filename isn't present. This is to prevent overwriting of existing files.



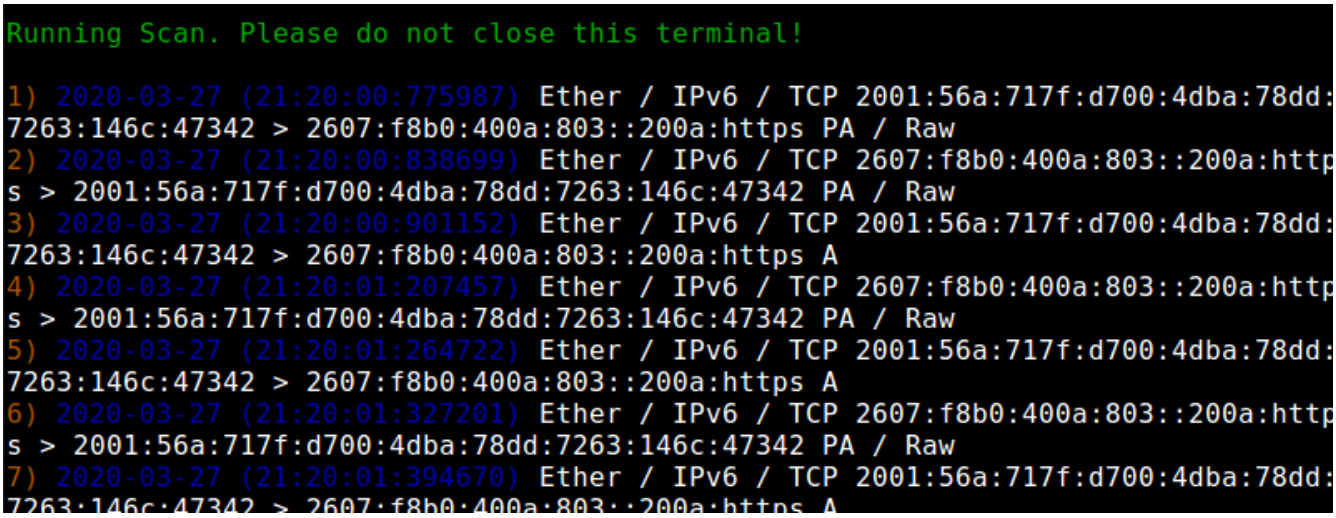
Step 4 sets the duration for the network capture to run for. The user can specify, minutes, hours, day or a time of day to run until. For example, "1m" would run a one-minute long capture. "3h 30m" would have the program run for three hours and thirty minutes. "07:15" would run until the local time is 7:15 am.



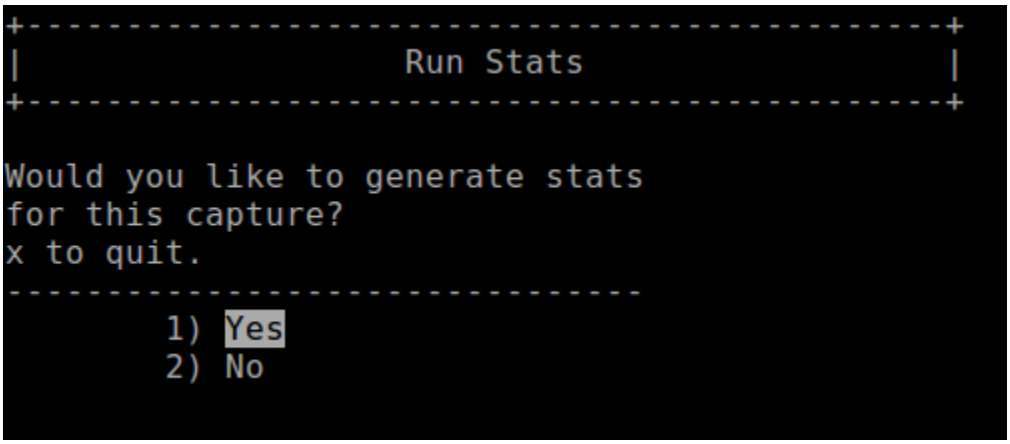
Once the user has completed the prior four steps, they can verify the capture settings selected.



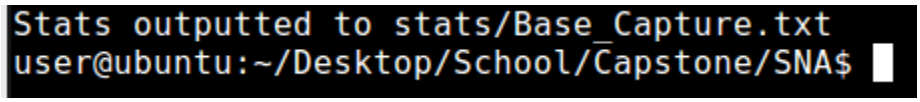
"Interface:" is the selected interface and "SSID:" is the set or discovered network SSID from Step 1. "BaseCapPath:" is the path that the capture file will be saved to from Step 2. "BaseFileName:" is the naming convention the program will use to label the created files, designated in Step 3. "PathFile:" is a demonstration of what the pcap file will be saved as and where. It will also show the appended numbers to the file name in the case that the named file already exists. "RunPeriod:" is used to identify the period that the network scan will run for, as specified in Step 4. The period will be represented by days:hours:minutes. If the user were to specify a time of day rather than a time period, the program will display "RunTime:" with the time specified in 24-hour format. If any of the information is incorrect, then the user can return to the step associated with the value and change the value and will then be brought back to the verification page as to avoid having to redo all proceeding steps. If all the data is correct, then the user can proceed to the traffic capture screen.



In this screen, the user can watch a summary of the packets that the program detects. The program will run until the specified time from Step 4, or the user interrupts the program (such as with ^C). In either case, the user will still receive a pcap file for what has been captured.



If the capture runs until the specified time, then the user will receive the option to generate statistical reports for the capture just made. If "yes", then the script will run the pcapstats.py script with the pcap file made. Otherwise the program will quit. When pcapreader.py is run alongside netreader.py, then what would be outputted as statistical tables, will be outputted into a text file, and the path displayed in the terminal.



pcapstats.py will output 16 tables and 15 graphs:

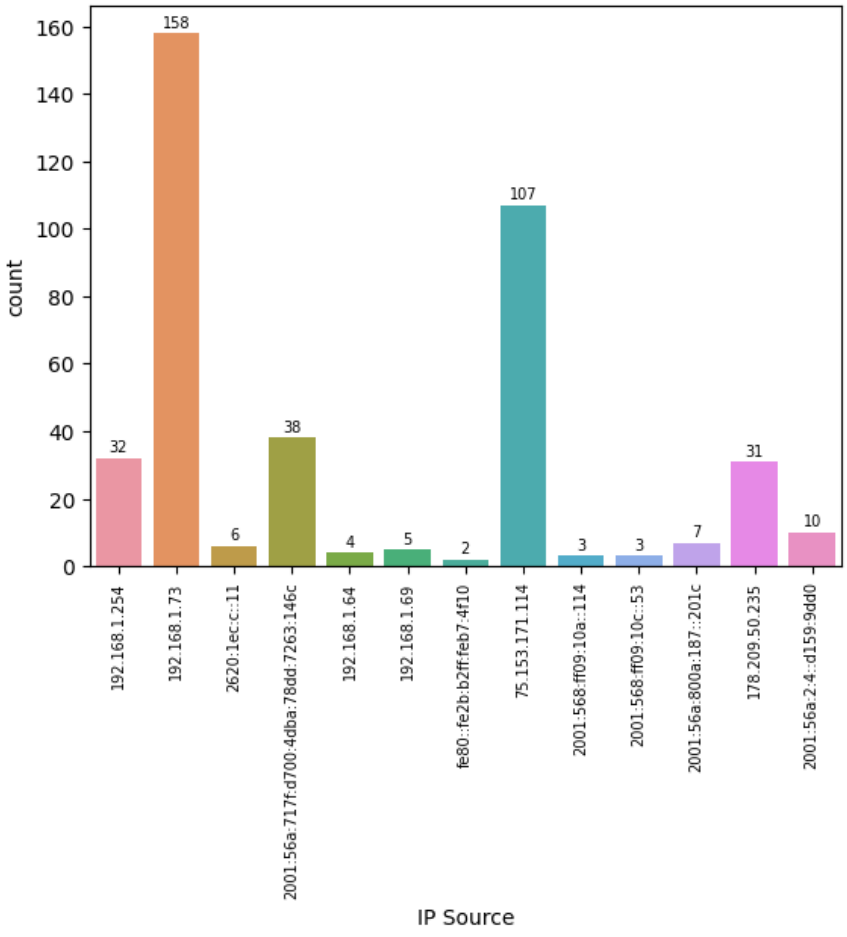
Tables	Graphs
Top Source IPs from Capture	IPSource_Count.png
IP Sources over Time	IPSource_OverTime.png
Top Destination IPs from Capture	IPDest_Count.png
IP Destinations over Time	IPDest_OverTime.png
Layer 3 Packet Count	L3Type_Count.png
Layer 3 Packets over Time	L3Type_OverTime.png
Layer 4 Packet Count	L4Type_Count.png
Layer 4 Packets over Time	L4Type_OverTime.png
Layer 5 Packet Count	L5Type_Count.png
Layer 5 Packets over Time	L5Type_OverTime.png
Top HTTP/HTTPS Requests	
Http/HTTPS Packets over Time	HTTP-HTTPS_OverTime.png
Top DNS name resolutions	DNSNameRes_Count.png
DNS Packets over Time	DNS_OverTime.png
Total ARP Requests by Source	ARPSource_Count.png
ARP Packets over Time	ARP_OverTime.png

Some samples of the graphs produced, and the tables associated with them:

Top Source IPs from Capture

IP Source	Count
192.168.1.73	158
75.153.171.114	107
2001:56a:717f:d700:4dba:78dd:7263:146c	38
192.168.1.254	32
178.209.50.235	31
2001:56a:2:4::d159:9dd0	10
2001:56a:800a:187::201c	7
2620:1ec:c::11	6
192.168.1.69	5
192.168.1.64	4
2001:568:ff09:10a::114	3
2001:568:ff09:10c::53	3
fe80::fe2b:b2ff:feb7:4f10	2

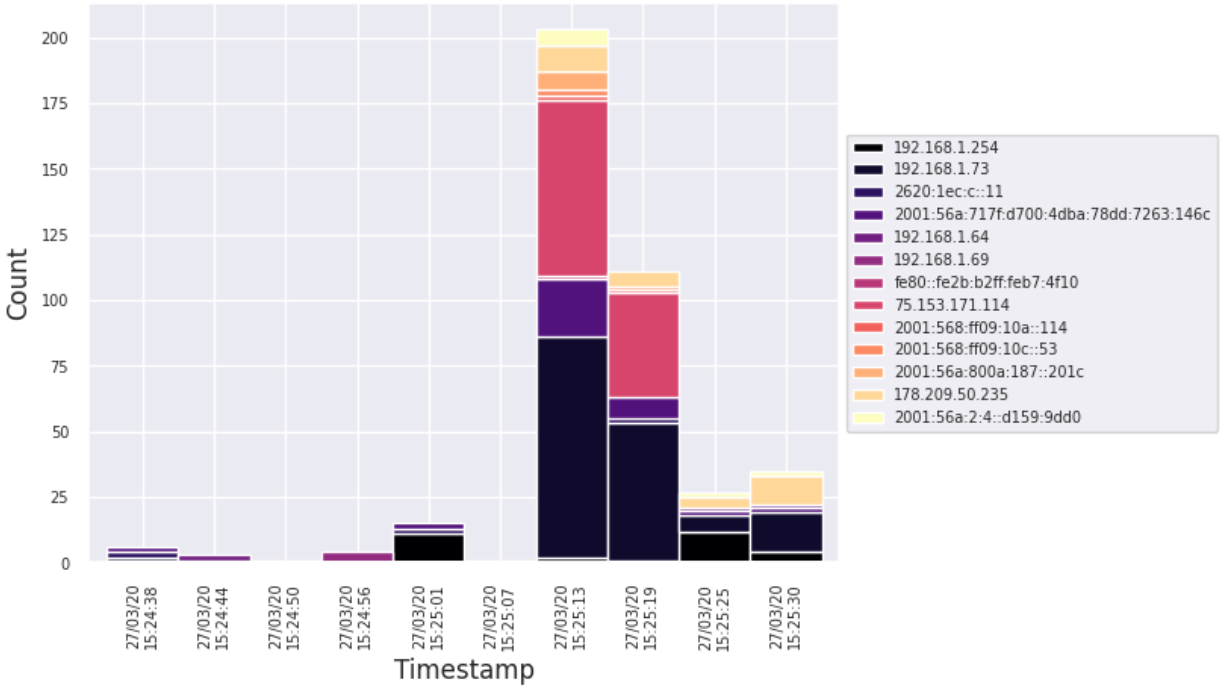
(IPSource_Count.png)



IP Sources over Time

	IP Source	27/03/20 15:24:38	27/03/20 15:24:44	27/03/20 15:24:50	27/03/20 15:24:56	27/03/20 15:25:01	27/03/20 15:25:07	27/03/20 15:25:13	27/03/20 15:25:19	27/03/20 15:25:25	27/03/20 15:25:30
0	192.168.1.254	1	1	0	0	11	0	2	1	12	4
1	192.168.1.73	1	0	0	0	0	0	84	52	6	15
2	2620:1ec:c::11	2	0	0	0	2	0	0	2	0	0
3	2001:56a:717f:d700:4dba:78dd:7263:146c	2	0	0	0	2	0	22	8	2	2
4	192.168.1.64	0	2	0	0	0	0	0	0	1	1
5	192.168.1.69	0	0	1	4	0	0	0	0	0	0
6	fe80::fe2b:b2ff:feb7:4f10	0	0	0	0	0	1	1	0	0	0
7	75.153.171.114	0	0	0	0	0	0	67	40	0	0
8	2001:568:ff09:10a::114	0	0	0	0	0	0	2	1	0	0
9	2001:568:ff09:10c::53	0	0	0	0	0	0	2	1	0	0
10	2001:56a:800a:187::201c	0	0	0	0	0	0	7	0	0	0
11	178.209.50.235	0	0	0	0	0	0	10	6	4	11
12	2001:56a:2:4::d159:9dd0	0	0	0	0	0	0	6	0	2	2

(IPSource_OverTime.png)



Challenges

Initially, when developing the Statistical Network Analyzer (SNA) tool, it was intended to be incorporated into an Intrusion Detection System (IDS). The tool would have started by creating a base capture reading and generating statistical values for the capture (which has now evolved into netreader.py and pcapreader.py). From this base capture, the user would set a threshold for comparison which, when surpassed, would be deemed as abnormal traffic, such as 200% more TCP traffic than the base. Using this, the system would start up a persistent network reader that would constantly be comparing its values to the base reference. If the thresholds were to be surpassed, then the user would be sent an email or a notification altering them of the "abnormal" traffic. In the progress of attempting to build the IDS system, the deadline for the project was approaching and it was perceived that there would not be enough time to implement a functional persistent network reader with interface,

synchronous analyzing and alerting methods. As such, the network reader and statistic generator were extracted and used to form a PCAP File Statistical Analyzer and a Network Traffic Reader.

Primarily, the greatest challenge with developing the Statistical Network Analyzer (SNA) tool was using Python3 to read network traffic and parse the data into a readable format. Initially, the tool manually parsed every single packet it received from a python Socket. The major complication with manual parsing is that it is slow and is limited by what is defined. Manual parsing required to define as many headers as possible for dissection, which lead to only defined packets being dissectible and the remaining limited by whether they are identifiable by a defined code. This often meant that packets could be misinterpreted or not accounted for at all. Similarly, due to manual parsing, the program slowed down with the more headers defined to be dissected. If the tool only parsed a single layer and only the most common header types in that layer, it would not be a problem; however, it is more valuable to be able to analyze a large variety of packet types and data. To resolve this issue, the module Scapy was implemented as it reliably dissects and identifies almost all packet types as well has built-in functionality to parse packets into a structure and save the structure into pacp file format. Although I had discarded the manual parsing, it did provide myself with a more in-depth understanding of common packet header and bit-wise parsing.

Security Risks

The Statistical Network Analyzer (SNA) on its own does not introduce new risks to an existing environment, as the tool does not alter the existing network system only reads traffic and saves files. To read the network traffic, the tool does not open any ports or interface outside of its designated environment. SNA follows the same risks as any network traffic reader and that is hijacking and reading unsecured packet data, such as HTTP traffic sent in plan text.

Although the netreader.py does require SuperUser privileges (sudo) to properly read traffic, it does not alter the traffic; however, since it uses sudo, files saved by the tool will also be created with the same privileges. As such, files could be written to areas where only sudo can edit and despite the program is designed to not overwrite existing files, it could create files or directories in unwanted areas. Users still require entering in their credentials to permit the program to run with sudo permissions as netreader.py will check if the user ran the program with sudo. The program could be used for privilege escalation or putting a file/directory into a sudo required place, but the user still must know the sudo credentials to run the program and execution time is limited.

As a tool that can read network traffic, there is the potential for it to be used maliciously. Like the software Wireshark, it can read the network traffic that is sent and received on a network interface and store the information within the packets into a readable format. Ultimately, the main security risks of the tool are malicious use, such as analyzing weaknesses in target networks, or expanding on the code to perform further actions, such as network traffic logging sent to a server.

Conclusion

The Statistical Network Analyzer (SNA) tool aims to make generating statistical analytics from network traffic more efficient and provide a visual and text-based representation. In addition to statistical analysis, the tool provides a command-line based network traffic reader, which produces pcap files and allows the user to set the reader to scan for a set period, rather than manual starting and stopping. Despite its simplicity, it saves the user time as opposed to manual statistical analysis and is open to further expansion and even integration into other tools and programs.

Modules

1. MATPLOTLIB – https://matplotlib.org/
2. PANDAS – https://pandas.pydata.org/
3. PRETTYTABLE – https://pypi.org/project/PrettyTable/
4. SCAPY – https://scapy.net/
5. SEABORN – https://seaborn.pydata.org/
6. TABULATE – https://pypi.org/project/tabulate/
7. TERMCOLOR – https://pypi.org/project/termcolor/

Primary ISS Course References

1. ITSC-200: Network Protocols and Security
    - Network traffic analysis and protocols
2. STAT-245-ISS: Statistics for Engineering and Technologies
    - Statistical comprehension and visual graphing
3. ITSC-202: Secure Programming Essentials
    - Foundational Programming
4. ITSC-203: Offensive and Defensive Tool Construction
    - Python and Bash Scripting fundamentals, python packet sniffing
5. ITSC-206: Advanced Networking in Offensive and Defensive Environments
    - Utilization of network analysis and security tools

References

1. Automatically harvesting Information using Python - A Basic Approach to Snooping on Network Traffic in Python. (n.d.). Retrieved from https://www.nexsoftsys.com/articles/basic-approach-to-snooping-on-network-traffic-in-python.html
2. Curses Programming with Python. (n.d.). Retrieved from https://docs.python.org/3/howto/curses.html
3. Eli Shlomo. (2020, March 2). Network Data Visualization with Python (part 1). Retrieved from https://www.eshlomo.us/network-data-visualization-with-python-part-1/
4. Extracting URLs from network traffic in just 9 Python lines with Scapy-HTTP. (n.d.). Retrieved from http://www.lucainvernizzi.net/blog/2015/02/12/extracting-urls-from-network-traffic-in-just-9-python-lines-with-scapy-http/
5. Gaffsey & Jona. (1966, July 1). How to make a menu in Python navigable with arrow keys. Retrieved from https://stackoverflow.com/questions/39488788/how-to-make-a-menu-in-python-navigable-with-arrow-keys
6. Get smarter about what matters to you. (n.d.). Retrieved from https://secdevops.ai/learning-packet-analysis-with-data-science-5356a3340d4e
7. How to Intercept / Sniff live traffic data in a network using Python. (n.d.). Retrieved from https://www.shellvoide.com/python/intercept-and-sniff-live-traffic-data-in-a-network-in-python/
8. How to Run a Shell Command from Python and Get The Output? (2018, March 6). Retrieved from https://cmdlinetips.com/2014/03/how-to-run-a-shell-command-from-python-and-get-the-output/
9. krish7919, cronos 1, & eminor. (1962, September 1). Get all the layers in a packet. Retrieved from https://stackoverflow.com/questions/13549294/get-all-the-layers-in-a-packet
10. Ladd, J., Otis, J., Warren, C. N., & Weingart, S. (2017, August 23). Exploring and Analyzing Network Data with Python. Retrieved from https://programminghistorian.org/en/lessons/exploring-and-analyzing-network-data-with-python
11. McManus, J. (n.d.). Data Harvest " Linux Magazine. Retrieved from http://www.linux-magazine.com/Issues/2019/220/Packet-Analysis-with-Scapy
12. McManus, J. (n.d.). Visualizing Network Data Using Python: Part 1. Retrieved from https://blog.automox.com/visualizing-network-data-using-python-part-1
13. Scapy p.04 – Looking at Packets. (2013, October 29). Retrieved from https://thepacketgeek.com/scapy-p-04-looking-at-packets/
14. Scapy Sniffing with Custom Actions, Part 1. (2013, October 2). Retrieved from https://thepacketgeek.com/scapy-sniffing-with-custom-actions-part-1/
15. Secdevopsai. (n.d.). secdevopsai/Packet-Analytics. Retrieved from https://github.com/secdevopsai/Packet-Analytics
16. Singh, S. (n.d.). Create Simple Packet Sniffer Using Python. Retrieved from http://www.bitforestinfo.com/2017/01/how-to-write-simple-packet-sniffer.html