Packet Sniffer in Python using Scapy and SSLstrip

Siddhu chelluru

B.tech Cyber security and Digital forensics

VIT University, Bhopal

Siddhuchelluru@lovecoder.tech

Abstract

This paper presents a packet sniffer implementation in Python that captures and analyzes network packets using the Scapy library and the SSLstrip tool. The packet sniffer allows users to specify a network interface to sniff on, apply a BPF filter to customize which packets are captured, and write the captured packets to a PCAP file if desired. The packet sniffer also supports the use of a network tap device to capture all traffic on a network segment, and uses SSLstrip to downgrade HTTPS connections to HTTP. The packet sniffer prints a summary of each packet captured and waits for a keyboard interrupt (Ctrl-C) to stop the sniffer. If an output file is specified, the captured packets are written to the file in PCAP format when the sniffer is stopped.

Introduction

Packet sniffing is a technique used to capture and analyze network packets in order to monitor network traffic, troubleshoot network issues, and perform security assessments. Packet sniffers can be used to capture packets on a network interface, apply filters to customize which packets are captured, and write the captured packets to a file for later analysis.

There are many tools and libraries available for packet sniffing, each with their own set of features and capabilities. In this paper, we present a packet sniffer implementation in Python using the Scapy library and the SSLstrip tool. Scapy is a powerful library for interacting with

network interfaces, sending and receiving packets, and dissecting and modifying packet contents. SSLstrip is a tool that downgrades HTTPS connections to HTTP, allowing the sniffer to capture and analyze the unencrypted traffic.

Requirements

To use the packet sniffer presented in this paper, the following requirements must be met:

- Python 3.6 or later
- The Scapy library, which can be installed using **pip install scapy**
- The SSLstrip tool, which can be installed using **pip install sslstrip**
- Root privileges to run the packet sniffer on a network interface (sudo can be used to run the script with root privileges)

Instructions

To use the packet sniffer, follow these instructions:

1. Save the packet sniffer code to a file, such as **packet_sniffer.py**.
2. Run the script with the **-i** flag to specify the network interface to sniff on, and optional **-f**, **-o**, and **-t** flags to specify a BPF filter, output file, and network tap device.

   For example: `./packet_sniffer.py -i eth0 -f "tcp port 80" -o http_traffic.pcap -t tap0`

This will start SSLstrip on the **eth0** interface, start the sniffer on the **tap0** device, and only capture TCP packets on port 80, writing the packets to the **http_traffic.pcap** file.

Code Overview

The packet sniffer script is organized as follows:

1. The **parse_arguments** function uses the **argparse** module to parse command-line arguments passed to the script. The script accepts the following arguments:

- **-i** or **--interface**: the network interface to sniff on (required).

- **-f** or **--filter**: a BPF filter to apply to the packets captured by the sniffer.
- **-o** or **--output**: the name of a PCAP file to save the captured packets to.
- **-t** or **--tap**: the name of a network tap device to use to capture all traffic on a network segment.

2. The **run_sslstrip** function starts the SSLstrip tool on the specified network interface. SSLstrip is used to downgrade HTTPS connections to HTTP, allowing the sniffer to capture and analyze the unencrypted traffic.
3. The **sniffer** function creates an instance of Scapy's **AsyncSniffer** class. If a network tap device is specified, the sniffer will capture all traffic on the network segment. Otherwise, the sniffer will only capture traffic to or from the host. The sniffer applies the specified BPF filter and prints a summary of each packet captured. The sniffer is then started and waits for a keyboard interrupt (Ctrl-C) to stop it. If an output file is specified, the captured packets are written to the file in PCAP format when the sniffer is stopped.
4. The script then calls the **parse_arguments** and **sniffer** functions with the command-line arguments passed to the script. If a network tap device is specified, the **run_sslstrip** function is called to start SSLstrip before starting the sniffer.

Issues

Here are some common issues that you may encounter when using the packet sniffer presented in this paper:

- Permission issues: Packet sniffing requires low-level access to network interfaces, which may not be granted to all users. To overcome this, you can run the packet sniffer with root privileges using **sudo**.
- Filtering out noise: On busy networks, you may receive a large volume of packets that are not relevant to your analysis. You can use BPF filters to specify the types of packets you want to capture, or add additional filtering logic to your code to exclude unwanted packets.
- Handling encrypted traffic: Encrypted traffic cannot be easily analyzed, as the contents of the packets are not visible. One solution is to use a tool like SSLstrip to downgrade HTTPS connections to HTTP, allowing you to analyze the unencrypted traffic. However, this can be a security risk and should be used with caution.
- Capturing packets from other hosts: By default, most sniffers will only capture packets that are sent to or from the host running the sniffer. To capture packets from other hosts, you will need to use a network tap or switch that allows you to see all of the traffic on a network segment.

- Handling large volumes of traffic: If you are capturing packets at high rates or for extended periods of time, you may need to optimize your code to handle the large volume of data efficiently. One option is to use a tool like TShark, which is optimized for high-speed packet capture and can write packets directly to disk.

Conclusion

The packet sniffer presented in this paper is a useful tool for capturing and analyzing network packets in Python using the Scapy library and the SSLstrip tool. It allows users to specify a network interface to sniff on, apply a BPF filter to customize which packets are captured, and write the captured packets to a PCAP file if desired. The packet sniffer also supports the use of a network tap device to capture all traffic on a network segment, and uses SSLstrip to downgrade HTTPS connections to HTTP. The packet sniffer prints a summary of each packet captured and waits for a keyboard interrupt (Ctrl-C) to stop the sniffer. If an output file is specified, the captured packets are written to the file in PCAP format when the sniffer is stopped.

Reference

- Moxie Marlinspike. (2010). SSLstrip. Retrieved from https://github.com/moxie0/sslstrip
- Scapy. (2021). Scapy. Retrieved from https://scapy.net/
- TShark. (2021). TShark. Retrieved from https://www.wireshark.org/docs/man-pages/tshark.html