Basic Network Sniffer Report

Prepared by: Vali Gasimli
Date: 04.24.2025

CONTENT

We will create a file using Python code and run it to monitor network traffic.
We open Kali Linux from VirtualBox and install the Scapy library using:

```
sudo apt update
pip3 install scapy
```

After the installation, we will create a file using **`nano`** and run the Python code.

This line imports the **`sniff`** function from the Scapy library.

`sniff` allows you to capture network packets on an interface.

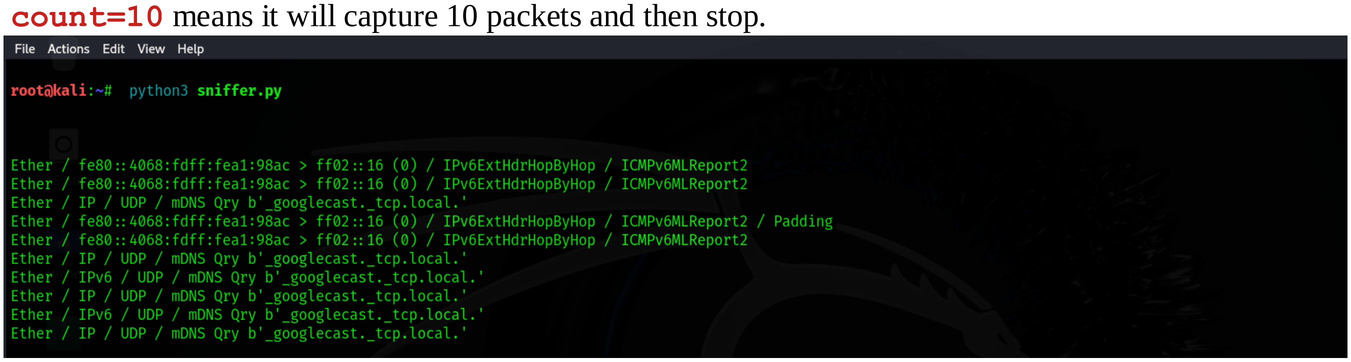
Every time a packet is captured, this function will be called.

`packet.summary()` gives a short description of the packet (for example: **`IP / TCP 192.168.1.10:443 > 192.168.1.5:50678)`**.

This starts the packet sniffing process.

`prn=packet_callback` tells Scapy to run the **`packet_callback`** function on each captured packet.

`count=10` means it will capture 10 packets and then stop.



We run it using python3.

`Ether / fe80::4068:fdff:fea1:98ac > ff02::16 / IPv6ExtHdrHopByHop / ICMPv6MLReport2`

- **Ether**: Ethernet frame (data link layer).
- **fe80::4068:fdff:fea1:98ac**: A link-local IPv6 address (usually auto-assigned by a device).
- **ff02::16**: A multicast IPv6 address used for ICMPv6 (specific for multicast group communication).
- **IPv6ExtHdrHopByHop**: An IPv6 extension header that tells each router on the path to process the packet (used for control messages).
- **ICMPv6MLReport2**: Multicast Listener Report (version 2) — used by a host to report its interest in joining a multicast group.

This means a device is announcing interest in joining a multicast group (often for network service discovery).

Repeated `ICMPv6MLReport2` lines

This is the same message being sent multiple times. Devices may send these reports periodically or in response to certain network conditions.

`Ether / IP / UDP / mDNS Qry b'_googlecast._tcp.local.'`

- **Ether / IP / UDP**: Standard Ethernet, IPv4, and UDP headers.
- **mDNS Qry**: A Multicast DNS query.
- **`_googlecast._tcp.local.`**: A service discovery request — this is specifically querying for devices offering Google Cast (e.g., Chromecast).

This line means the device is searching the local network for Google Cast-enabled devices using mDNS.

Padding

- **`/ Padding`**: Sometimes added to align packets or fill minimum length requirements for Ethernet frames.

IPv6 version of mDNS query

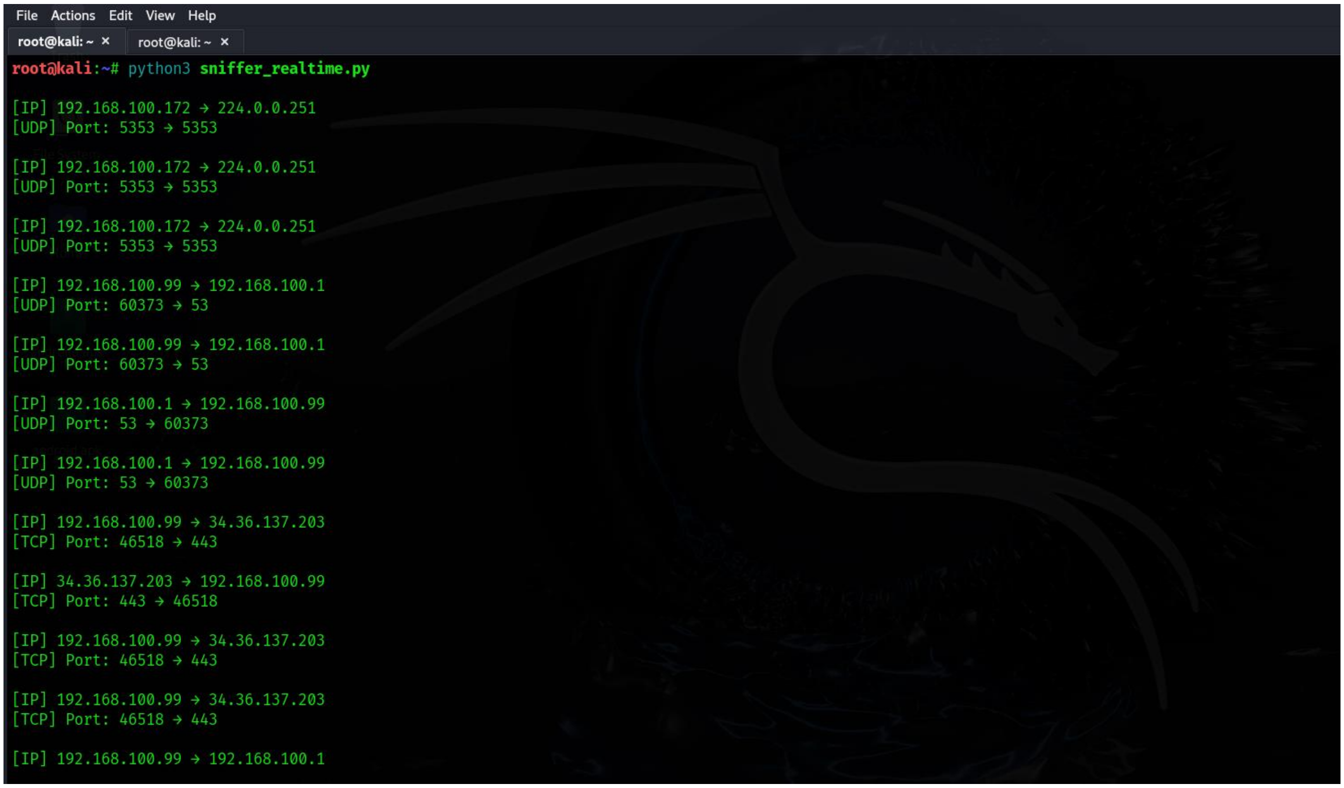
- **`Ether / IPv6 / UDP / mDNS Qry b'_googlecast._tcp.local.'`**
  Same as before, but over **IPv6** instead of IPv4. Many modern devices use both.

We will monitor live traffic at the terminal.Let's create a file with nano and run it using python3.

This script captures live network traffic using **Scapy**.

- It checks if the packet has an **IP** layer and prints the source and destination **IP** addresses.
- If the packet contains a **TCP** or **UDP** layer, it also prints the source and destination ports.
- The **`sniff`** function captures 30 packets and passes each one to the **`packet_callback`** function for processing.

It's useful for monitoring and analyzing basic **IP**, **TCP**, and **UDP** traffic.



Got it — you uploaded a screenshot showing a Python script running on Kali Linux. Here's a breakdown of what's happening in the image:

- You're executing a script called **`sniffer_realtime.py`** with Python 3.
- The script is capturing network packets in real-time.
- It displays **IP** and **UDP/TCP** information for each packet.

Details from the output:

- **Multicast communication:**
  `192.168.100.172` is sending UDP packets to `224.0.0.251` on port `5353` (this is typically Multicast DNS, mDNS).
- **DNS traffic:**
  `192.168.100.99` and `192.168.100.1` are communicating over UDP, where port `53` is involved — classic for DNS queries/responses.
- **HTTPS traffic:**
  `192.168.100.99` is making TCP connections to `34.36.137.203` on port `443` — this is encrypted HTTPS traffic.

- **Bi-directional traffic:**
  You see responses coming back (from `34.36.137.203` to `192.168.100.99`), which means it's showing both sides of the connection.

Let's monitor traffic using nethogs.

sudo apt update
sudo apt install nethogs

You are running **NetHogs** (version 0.8.8-1) — a real-time network traffic monitor that shows how much bandwidth each process (program) is using.

Here's what the output shows:

- **PID** = Process ID (the unique ID of the running program)
- **USER** = The user who owns the process (here, it's all `root`)
- **PROGRAM** = The name and path of the program generating the traffic
- **DEV** = The network interface used (`wlan0` — wireless interface)
- **SENT** = How much data is being sent (upload)
- **RECEIVED** = How much data is being received (download)

Details from the screenshot:

- The `firefox-esr` browser (PID 64936) is actively receiving data:
  2.550 kB/s upload, **91.811 kB/s download**.
- There are two `curl` processes listed (PID 66241 and 66041), but they're not sending or receiving any data at the moment.
- There's also an "unknown TCP" connection that NetHogs couldn't fully identify (shows `?` for PID), but it's not transferring data either.

At the bottom, **TOTAL** shows the overall traffic on `wlan0` — matching what Firefox is doing, since it's the only active process.