Practical 6

Aim: Perform practical approach to implement Footprinting-Gathering target information using UAtester.

Theory:

UAtester: UAtester is a Python tool used for passive footprinting, especially for identifying **User-Agent-based responses** from a target web server. It's typically used to determine how a server responds to various User-Agent headers—sometimes exposing different behavior to crawlers, bots, or mobile devices.

Use Case: Footprinting Using UAtester

We'll use UAtester to:

1. **Send multiple HTTP requests to the target** using different User-Agent headers.
2. **Analyze how the server responds** to different UAs (e.g., Googlebot vs. Firefox).

Footprinting & where UA testing fits

Footprinting = the information-gathering phase of a security assessment. Goal: collect as much passive and active information about a target so later testing is focused and lower risk.

Common footprinting categories:

- **Passive**: public data sources (WHOIS, DNS records, certificates, web archives, search engines, social media, job posts). Low likelihood of detection.
- **Active**: direct probes (port scans, service banners, application endpoints). Higher chance of logging/detection.

User-Agent (UA) testing is a targeted web fingerprinting technique. Web servers/applications sometimes:

- Serve different content to crawlers vs browsers (cloaking).
- Block or rate-limit certain clients (curl/wget) via WAF rules.
- Present different features to mobile vs desktop UAs.
- Show debug or staging pages to specific UAs or IPs.

Why test UAs?

- Detect cloaking (important for site owners and examiners).
- Reveal hidden routes or different content for bots (could be sensitive).
- Find filtering/WAF behavior (helps tailor further tests).
- Confirm whether server does UA-based routing/logic.

Send a few HTTP requests to a target with different **User-Agent** headers, then report status and response length so you can spot different behavior (cloaking, blocking, etc.).

Code:

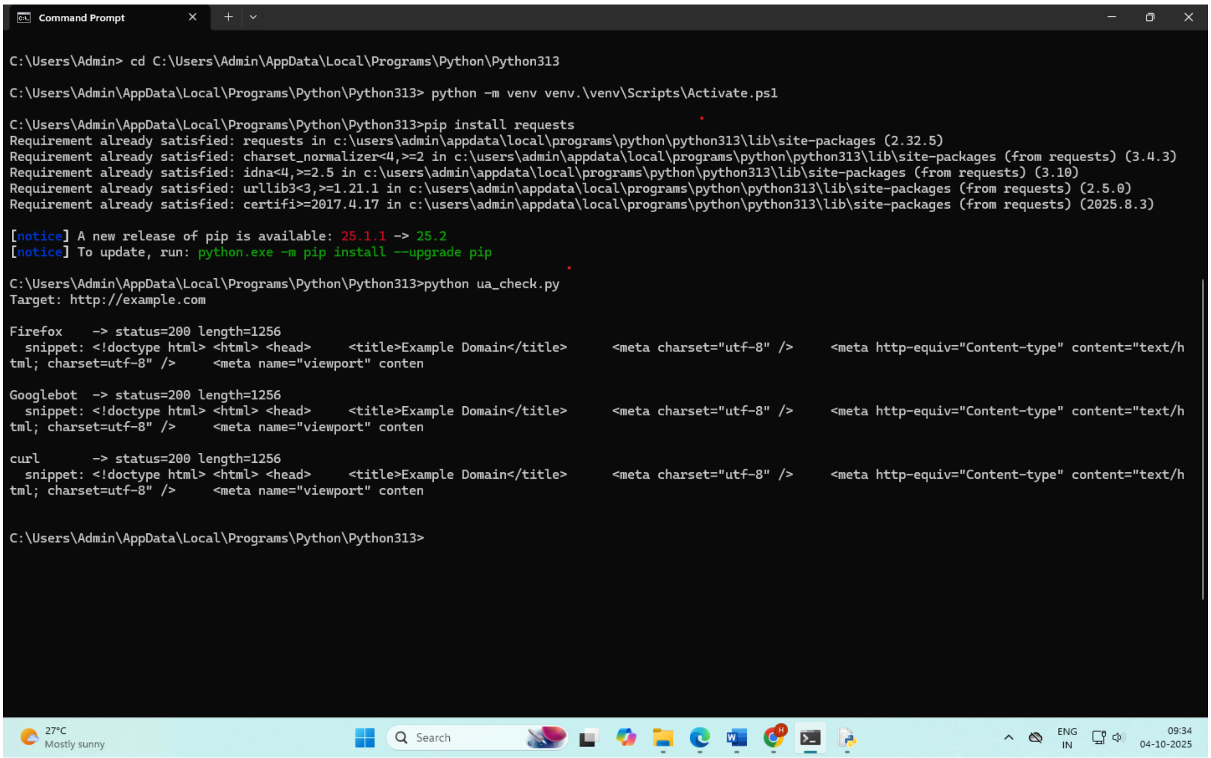
```
# ua_check.py (save code with ua_check.py)
import requests
TARGET = "http://example.com"   # <-- change to your target (or use http://localhost)
USER_AGENTS = {
    "Firefox": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:104.0) Gecko/20100101 Firefox/104.0",
    "Googlebot": "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)",
    "curl": "curl/7.68.0"
}
def check_ua(url):
    results = {}
    for name, ua in USER_AGENTS.items():
        try:
            r = requests.get(url, headers={"User-Agent": ua}, timeout=8)
            results[name] = {
                "status": r.status_code,
                "length": len(r.text),
                "snippet": r.text[:200].replace("\n", " ")
            }
        except Exception as e:
            results[name] = {"status": "ERROR", "error": str(e)}
    return results

if __name__ == "__main__":
    res = check_ua(TARGET)
```

```
print(f"Target: {TARGET}\n")
for ua_name, info in res.items():
    if info.get("status") == "ERROR":
        print(f"{ua_name:10} -> ERROR: {info['error']}")
    else:
        print(f"{ua_name:10} -> status={info['status']} length={info['length']}")
        print(f"  snippet: {info['snippet']}\n")
```

How to run on command prompt:

1. python -m venv venv.\venv\Scripts\Activate.ps1
2. pip install requests
3. python ua_check.py

Output:



```
C:\Users\Admin> cd C:\Users\Admin\AppData\Local\Programs\Python\Python313

C:\Users\Admin\AppData\Local\Programs\Python\Python313> python -m venv venv.\venv\Scripts\Activate.ps1

C:\Users\Admin\AppData\Local\Programs\Python\Python313>pip install requests
Requirement already satisfied: requests in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (2.32.5)
Requirement already satisfied: charset_normalizer<4,>=2 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from requests) (3.4.3)
Requirement already satisfied: idna<4,>=2.5 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from requests) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from requests) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from requests) (2025.8.3)

[notice] A new release of pip is available: 25.1.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\Admin\AppData\Local\Programs\Python\Python313>python ua_check.py
Target: http://example.com

Firefox    -> status=200 length=1256
  snippet: <!doctype html> <html> <head>     <title>Example Domain</title>      <meta charset="utf-8" />     <meta http-equiv="Content-type" content="text/h
tml; charset=utf-8" />     <meta name="viewport" conten

Googlebot  -> status=200 length=1256
  snippet: <!doctype html> <html> <head>     <title>Example Domain</title>      <meta charset="utf-8" />     <meta http-equiv="Content-type" content="text/h
tml; charset=utf-8" />     <meta name="viewport" conten

curl       -> status=200 length=1256
  snippet: <!doctype html> <html> <head>     <title>Example Domain</title>      <meta charset="utf-8" />     <meta http-equiv="Content-type" content="text/h
tml; charset=utf-8" />     <meta name="viewport" conten


C:\Users\Admin\AppData\Local\Programs\Python\Python313>
```

Practical 7

Aim: Working with sniffers for monitoring network communication (Wireshark).

Theory:

A packet sniffer captures raw network frames from a network interface and decodes protocol layers (Ethernet → IP → TCP/UDP → HTTP, DNS, etc.).
Use cases:

- Troubleshooting connectivity and performance
- Debugging application protocols (HTTP, DNS, SMTP)
- Detecting anomalies (ARP spoofing, scanning, exfiltration)
- Security analysis / incident response (with permission)

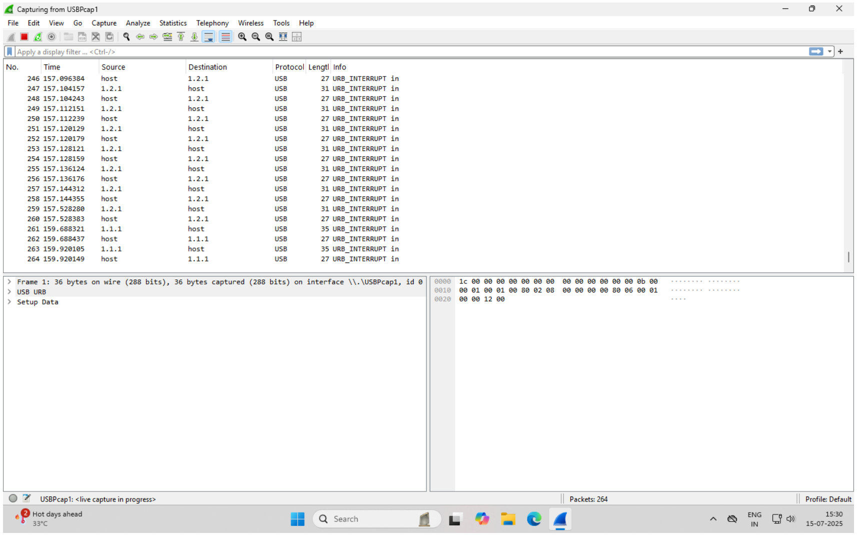
Install & first capture (GUI & CLI)

Wireshark (GUI) — install from your OS packages or wireshark.org.
tshark is Wireshark's CLI equivalent. tcpdump and ngrep are lighter CLI tools.

Start a simple capture with Wireshark:

1. Open Wireshark → select the correct interface (e.g., eth0, wlan0, en0).
2. Click the shark-fin Start button.
3. Use a simple capture filter immediately (see next section) to limit traffic volume.
4. Stop and save (File → Save).

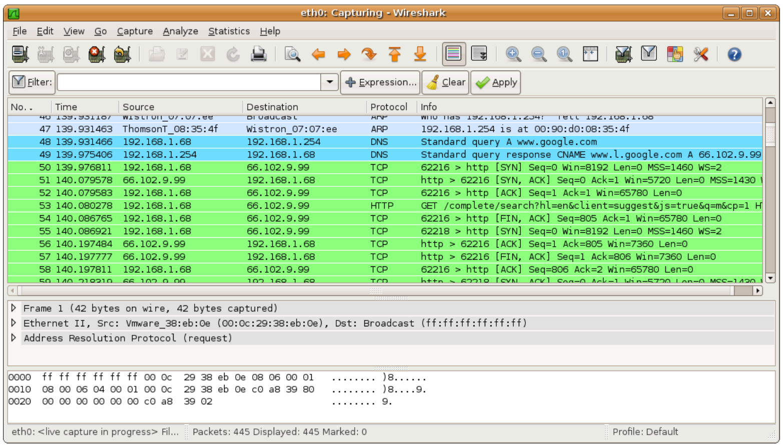
Output


Capturing from USBPcap1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
USBPcap1: <live capture in progress>
Packets: 264
Profile: Default
Hot days ahead
33°C
Search
ENG
IN
15:30
15-07-2025

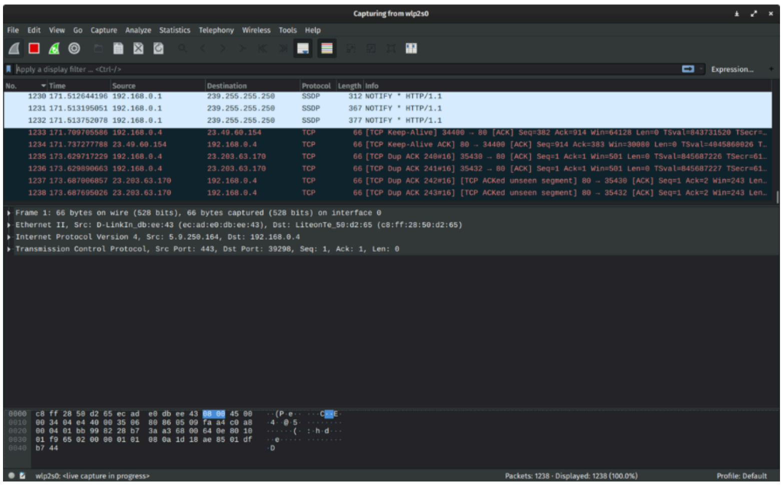
```
No.    Time        Source    Destination   Protocol Lengtl Info
   246 157.096384  host      1.2.1         USB        27 URB_INTERRUPT in
   247 157.104157  1.2.1     host          USB        31 URB_INTERRUPT in
   248 157.104243  host      1.2.1         USB        27 URB_INTERRUPT in
   249 157.112151  1.2.1     host          USB        31 URB_INTERRUPT in
   250 157.112239  host      1.2.1         USB        27 URB_INTERRUPT in
   251 157.120129  1.2.1     host          USB        31 URB_INTERRUPT in
   252 157.120179  host      1.2.1         USB        27 URB_INTERRUPT in
   253 157.128121  1.2.1     host          USB        31 URB_INTERRUPT in
   254 157.128159  host      1.2.1         USB        27 URB_INTERRUPT in
   255 157.136124  1.2.1     host          USB        31 URB_INTERRUPT in
   256 157.136176  host      1.2.1         USB        27 URB_INTERRUPT in
   257 157.144312  1.2.1     host          USB        31 URB_INTERRUPT in
   258 157.144355  host      1.2.1         USB        27 URB_INTERRUPT in
   259 157.528280  1.2.1     host          USB        31 URB_INTERRUPT in
   260 157.528383  host      1.2.1         USB        27 URB_INTERRUPT in
   261 159.688321  1.1.1     host          USB        35 URB_INTERRUPT in
   262 159.688437  host      1.1.1         USB        27 URB_INTERRUPT in
   263 159.920105  1.1.1     host          USB        35 URB_INTERRUPT in
   264 159.920149  host      1.1.1         USB        27 URB_INTERRUPT in
```

```
> Frame 1: 36 bytes on wire (288 bits), 36 bytes captured (288 bits) on interface \\.\USBPcap1, id 0
> USB URB
> Setup Data
```

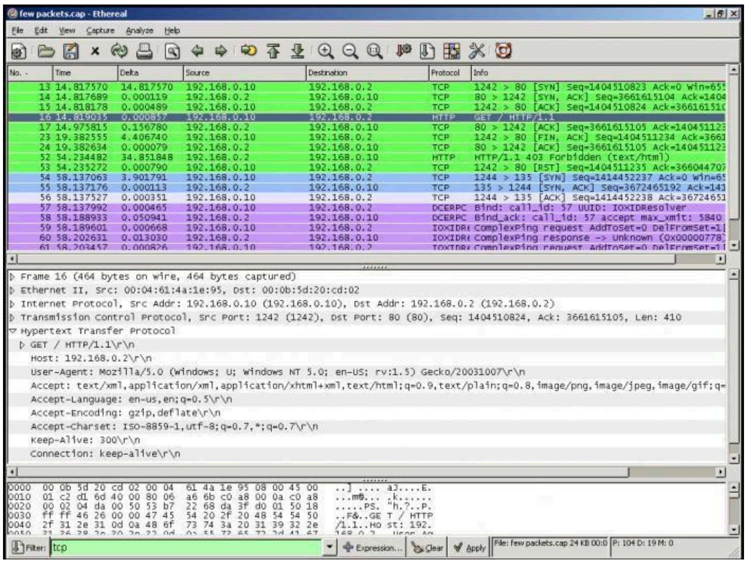
```
0000  1c 00 00 00 00 00 00 00  00 00 00 00 00 00 0b 00   ········ ········
0010  00 01 00 01 00 80 02 08  00 00 00 00 80 06 00 01   ········ ········
0020  00 00 12 00                                        ····
```


eth0: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Clear
Apply

```
No. .  Time        Source             Destination        Protocol  Info
   46 139.931187  Wistron_07:07:ee   Broadcast          ARP       Who has 192.168.1.254?  Tell 192.168.1.68
   47 139.931463  ThomsonT_08:35:4f  Wistron_07:07:ee   ARP       192.168.1.254 is at 00:90:d0:08:35:4f
   48 139.931466  192.168.1.68       192.168.1.254      DNS       Standard query A www.google.com
   49 139.975406  192.168.1.254      192.168.1.68       DNS       Standard query response CNAME www.l.google.com A 66.102.9.99
   50 139.976811  192.168.1.68       66.102.9.99        TCP       62216 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
   51 140.079578  66.102.9.99        192.168.1.68       TCP       http > 62216 [SYN, ACK] Seq=0 Ack=1 Win=5720 Len=0 MSS=1430
   52 140.079583  192.168.1.68       66.102.9.99        TCP       62216 > http [ACK] Seq=1 Ack=1 Win=65780 Len=0
   53 140.080278  192.168.1.68       66.102.9.99        HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=m&cp=1 H
   54 140.086765  192.168.1.68       66.102.9.99        TCP       62216 > http [FIN, ACK] Seq=805 Ack=1 Win=65780 Len=0
   55 140.086921  192.168.1.68       66.102.9.99        TCP       62218 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
   56 140.197484  66.102.9.99        192.168.1.68       TCP       http > 62216 [ACK] Seq=1 Ack=805 Win=7360 Len=0
   57 140.197777  66.102.9.99        192.168.1.68       TCP       http > 62216 [FIN, ACK] Seq=1 Ack=806 Win=7360 Len=0
   58 140.197811  192.168.1.68       66.102.9.99        TCP       62216 > http [ACK] Seq=806 Ack=2 Win=65780 Len=0
   59 140.218319  66.102.9.99        192.168.1.68       TCP       http > 62218 [SYN, ACK] Seq=0 Ack=1 Win=5720 Len=0 MSS=1430

▷ Frame 1 (42 bytes on wire, 42 bytes captured)
▷ Ethernet II, Src: Vmware_38:eb:0e (00:0c:29:38:eb:0e), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▷ Address Resolution Protocol (request)

0000  ff ff ff ff ff ff 00 0c  29 38 eb 0e 08 06 00 01   ........ )8......
0010  08 00 06 04 00 01 00 0c  29 38 eb 0e c0 a8 39 80   ........ )8....9.
0020  00 00 00 00 00 00 c0 a8  39 02                     ........ 9.

eth0: <live capture in progress> Fil...   Packets: 445 Displayed: 445 Marked: 0   Profile: Default
```


Capturing from wlp2s0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.    Time           Source          Destination      Protocol  Length  Info
  1230 171.512644196  192.168.0.1     239.255.255.250  SSDP      312 NOTIFY * HTTP/1.1
  1231 171.513195051  192.168.0.1     239.255.255.250  SSDP      367 NOTIFY * HTTP/1.1
  1232 171.513752078  192.168.0.1     239.255.255.250  SSDP      377 NOTIFY * HTTP/1.1
  1233 171.709705586  192.168.0.4     23.49.60.154     TCP        66 [TCP Keep-Alive] 34400 → 80 [ACK] Seq=382 Ack=914 Win=64128 Len=0 TSval=843731520 TSecr=…
  1234 171.737277788  23.49.60.154    192.168.0.4      TCP        66 [TCP Keep-Alive ACK] 80 → 34400 [ACK] Seq=914 Ack=383 Win=30080 Len=0 TSval=4045860026 T…
  1235 173.629717229  192.168.0.4     23.203.63.170    TCP        66 [TCP Dup ACK 240#16] 35430 → 80 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=845687226 TSecr=61…
  1236 173.629890663  192.168.0.4     23.203.63.170    TCP        66 [TCP Dup ACK 241#16] 35432 → 80 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=845687227 TSecr=61…
  1237 173.687006857  23.203.63.170   192.168.0.4      TCP        66 [TCP Dup ACK 242#16] [TCP ACKed unseen segment] 80 → 35430 [ACK] Seq=1 Ack=2 Win=243 Len…
  1238 173.687695026  23.203.63.170   192.168.0.4      TCP        66 [TCP Dup ACK 243#16] [TCP ACKed unseen segment] 80 → 35432 [ACK] Seq=1 Ack=2 Win=243 Len…

▸ Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: D-LinkIn_db:ee:43 (ec:ad:e0:db:ee:43), Dst: LiteonTe_50:d2:65 (c8:ff:28:50:d2:65)
▸ Internet Protocol Version 4, Src: 5.9.250.164, Dst: 192.168.0.4
▸ Transmission Control Protocol, Src Port: 443, Dst Port: 39298, Seq: 1, Ack: 1, Len: 0

0000  c8 ff 28 50 d2 65 ec ad  e0 db ee 43 08 00 45 00   ··(P·e·· ···C··E·
0010  00 34 04 e4 40 00 35 06  80 86 05 09 fa a4 c0 a8   ·4··@·5· ········
0020  00 04 01 bb 99 82 28 b7  3a a3 68 00 64 0e 80 10   ······(· :·h·d···
0030  01 f9 65 02 00 00 01 01  08 0a 1d 18 ae 85 01 df   ··e····· ········
0040  b7 44                                              ·D

wlp2s0: <live capture in progress>   Packets: 1238 · Displayed: 1238 (100.0%)   Profile: Default
```


few packets.cap - Ethereal
File Edit View Capture Analyze Help
Filter:
Expression...
Clear
Apply

```
No. -  Time       Delta      Source         Destination    Protocol  Info
   13 14.817570  14.817570  192.168.0.10   192.168.0.2    TCP       1242 > 80 [SYN] Seq=1404510823 Ack=0 Win=65
   14 14.817689  0.000119   192.168.0.2    192.168.0.10   TCP       80 > 1242 [SYN, ACK] Seq=3661615104 Ack=1404
   15 14.818178  0.000489   192.168.0.10   192.168.0.2    TCP       1242 > 80 [ACK] Seq=1404510824 Ack=36616151(
   16 14.819035  0.000857   192.168.0.10   192.168.0.2    HTTP      GET / HTTP/1.1
   17 14.975815  0.156780   192.168.0.2    192.168.0.10   TCP       80 > 1242 [ACK] Seq=3661615105 Ack=14045112:
   23 19.382555  4.406740   192.168.0.10   192.168.0.2    TCP       1242 > 80 [FIN, ACK] Seq=1404511234 Ack=3661
   24 19.382634  0.000079   192.168.0.2    192.168.0.10   TCP       80 > 1242 [ACK] Seq=3661615105 Ack=14045112:
   52 54.234482  34.851848  192.168.0.2    192.168.0.10   HTTP      HTTP/1.1 403 Forbidden (text/html)
   53 54.235272  0.000790   192.168.0.10   192.168.0.2    TCP       1242 > 80 [RST] Seq=1404511235 Ack=366044707
   54 58.137063  3.901791   192.168.0.10   192.168.0.2    TCP       1244 > 135 [SYN] Seq=1414452237 Ack=0 Win=65
   55 58.137176  0.000113   192.168.0.2    192.168.0.10   TCP       135 > 1244 [SYN, ACK] Seq=3672465192 Ack=141
   56 58.137527  0.000351   192.168.0.10   192.168.0.2    TCP       1244 > 135 [ACK] Seq=1414452238 Ack=36724651
   57 58.137992  0.000465   192.168.0.10   192.168.0.2    DCERPC    Bind: call_id: 57 UUID: IOXIDResolver
   58 58.188933  0.050941   192.168.0.2    192.168.0.10   DCERPC    Bind_ack: call_id: 57 accept max_xmit: 5840
   59 58.189601  0.000668   192.168.0.10   192.168.0.2    IOXIDRE   ComplexPing request AddToSet=0 DelFromSet=1[
   60 58.202631  0.013030   192.168.0.2    192.168.0.10   IOXIDRE   ComplexPing response -> Unknown (0x00000778)
   61 58.203457  0.000826   192.168.0.10   192.168.0.2    IOXIDRE   ComplexPing request AddToSet=0 DelFromSet=1[

▷ Frame 16 (464 bytes on wire, 464 bytes captured)
▷ Ethernet II, Src: 00:04:61:4a:1e:95, Dst: 00:0b:5d:20:cd:02
▷ Internet Protocol, Src Addr: 192.168.0.10 (192.168.0.10), Dst Addr: 192.168.0.2 (192.168.0.2)
▷ Transmission Control Protocol, Src Port: 1242 (1242), Dst Port: 80 (80), Seq: 1404510824, Ack: 3661615105, Len: 410
▽ Hypertext Transfer Protocol
  ▷ GET / HTTP/1.1\r\n
    Host: 192.168.0.2\r\n
    User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.5) Gecko/20031007\r\n
    Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,image/jpeg,image/gif;q=
    Accept-Language: en-us,en;q=0.5\r\n
    Accept-Encoding: gzip,deflate\r\n
    Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
    Keep-Alive: 300\r\n
    Connection: keep-alive\r\n

0000  00 0b 5d 20 cd 02 00 04  61 4a 1e 95 08 00 45 00   ..] .... aJ....E.
0010  01 c2 d1 6d 40 00 80 06  a6 6b c0 a8 00 0a c0 a8   ...m@... .k......
0020  00 02 04 da 00 50 53 b7  22 68 da 3f d0 01 50 18   .....PS. "h.?..P.
0030  ff ff 46 26 00 00 47 45  54 20 2f 20 48 54 54 50   ..F&..GE T / HTTP
0040  2f 31 2e 31 0d 0a 48 6f  73 74 3a 20 31 39 32 2e   /1.1..Ho st: 192.

Filter: tcp
File: few packets.cap 24 KB 00:0   P: 104 D: 19 M: 0
```

Practical 8

Aim: Demonstrate a python script to capture and analyze network packets.

Theory:

Packet Sniffing

Packet sniffing is the process of intercepting and analyzing network packets that pass through a network interface. It's useful for:

- Network diagnostics
- Intrusion detection
- Performance monitoring
- Security auditing

Scapy

Scapy is a powerful Python-based interactive packet manipulation program and library. It can:

- Capture live packets
- Decode and dissect packet layers (like IP, TCP, UDP)
- Send crafted packets
- Analyze network traffic

Python Script: Basic Packet Sniffer & Analyzer

🔧 Features:

- Captures a user-defined number of packets
- Identifies protocol types (TCP, UDP, ICMP, others)
- Shows top source and destination IPs

Expected Behavior When the Script Runs

1. Starts Sniffing Packets:
   - Captures up to 100 packets (as per packet_count=100).
   - Sniffing occurs on the default interface unless another is specified via interface.

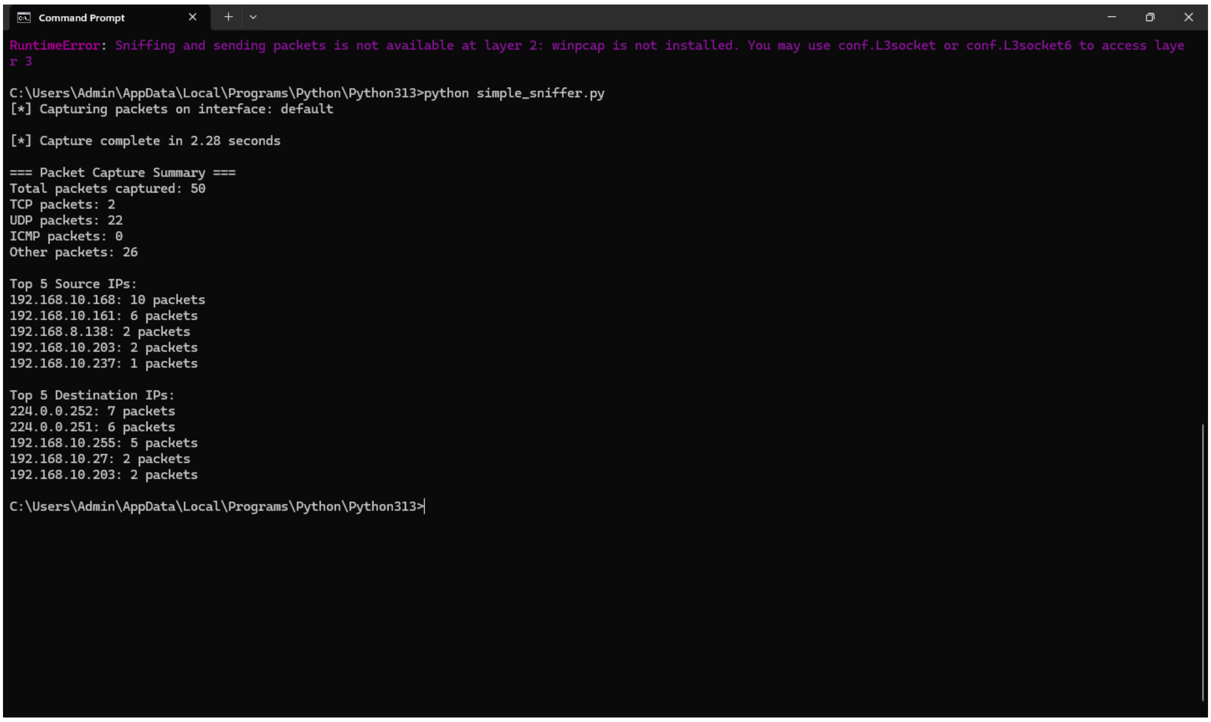
2. Analyzes Each Packet:
   - Checks if the packet contains an IP layer.
   - Increments counters for:
     - Total packets
     - TCP packets
     - UDP packets
     - ICMP packets
     - Other types
   - Tracks:
     - Source IPs
     - Destination IPs
3. After Capturing:
   - Prints the total number of packets captured.
   - Displays counts for each protocol.
   - Lists the Top 5 source and destination IPs based on frequency.

Code:

```
from scapy.all import sniff, IP, TCP, UDP, ICMP
from collections import Counter
import time

# Stores statistics
packet_stats = {
    "total": 0,
    "tcp": 0,
    "udp": 0,
    "icmp": 0,
    "other": 0,
    "src_ips": Counter(),
    "dst_ips": Counter()
}

# Function to analyze each captured packet
def analyze_packet(packet):
    packet_stats["total"] += 1

    if IP in packet:
        ip_layer = packet[IP]
        packet_stats["src_ips"][ip_layer.src] += 1
        packet_stats["dst_ips"][ip_layer.dst] += 1

        if TCP in packet:
            packet_stats["tcp"] += 1
        elif UDP in packet:
            packet_stats["udp"] += 1
        elif ICMP in packet:
            packet_stats["icmp"] += 1
        else:
            packet_stats["other"] += 1
    else:
        packet_stats["other"] += 1

# Function to print analysis results
def print_stats():
    print("\n=== Packet Capture Summary ===")
    print(f"Total packets captured: {packet_stats['total']}")
    print(f"TCP packets: {packet_stats['tcp']}")
    print(f"UDP packets: {packet_stats['udp']}")
    print(f"ICMP packets: {packet_stats['icmp']}")
    print(f"Other packets: {packet_stats['other']}")

    print("\nTop 5 Source IPs:")
    for ip, count in packet_stats["src_ips"].most_common(5):
        print(f"{ip}: {count} packets")

    print("\nTop 5 Destination IPs:")
    for ip, count in packet_stats["dst_ips"].most_common(5):
        print(f"{ip}: {count} packets")

# Main function to start packet capture
def start_sniffing(interface=None, packet_count=50, timeout=None):
    print(f"[*] Capturing packets on interface: {interface or 'default'}")
    start_time = time.time()

    sniff(
        iface=interface,          # Interface (None = default)
        prn=analyze_packet,       # Function called for each packet
        count=packet_count,       # Number of packets to capture
        timeout=timeout,          # Stop after timeout (in seconds)
        store=False               # Do not store packets in memory
    )

    duration = time.time() - start_time
    print(f"\n[*] Capture complete in {duration:.2f} seconds")
    print_stats()

if __name__ == "__main__":
```

```
# Run the sniffer

start_sniffing(packet_count=50)
```

Output:



```
RuntimeError: Sniffing and sending packets is not available at layer 2: winpcap is not installed. You may use conf.L3socket or conf.L3socket6 to access laye
r 3

C:\Users\Admin\AppData\Local\Programs\Python\Python313>python simple_sniffer.py
[*] Capturing packets on interface: default

[*] Capture complete in 2.28 seconds

=== Packet Capture Summary ===
Total packets captured: 50
TCP packets: 2
UDP packets: 22
ICMP packets: 0
Other packets: 26

Top 5 Source IPs:
192.168.10.168: 10 packets
192.168.10.161: 6 packets
192.168.8.138: 2 packets
192.168.10.203: 2 packets
192.168.10.237: 1 packets

Top 5 Destination IPs:
224.0.0.252: 7 packets
224.0.0.251: 6 packets
192.168.10.255: 5 packets
192.168.10.27: 2 packets
192.168.10.203: 2 packets

C:\Users\Admin\AppData\Local\Programs\Python\Python313>
```

Practical 9

Aim: Demonstrate the penetration testing tasks using python.

Theory:

Reconnaissance (passive/active): gather host/IP/service info (DNS, whois, HTTP headers).

Scanning / Enumeration: discover open ports, services, versions.

Vulnerability analysis: map services to potential weaknesses (in lab, using safe checks).

Code:

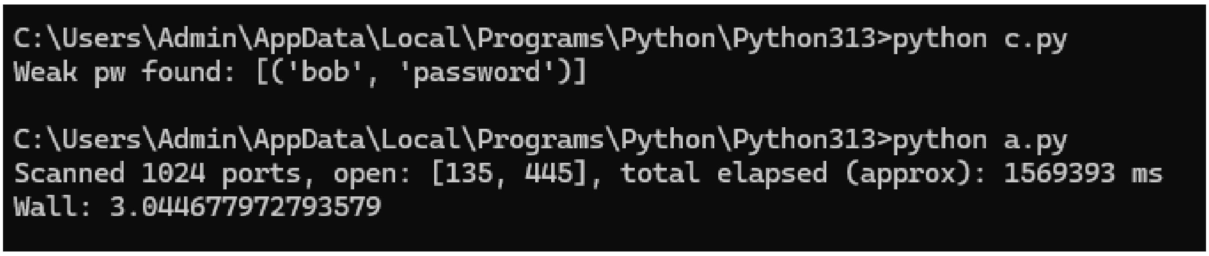
A: Fast async TCP port scanner

```
import asyncio, time


TARGET = "127.0.0.1"
PORTS = range(1, 1025)
CONCURRENCY = 500
TIMEOUT = 1.0
async def try_connect(sema, host, port):
    start = time.perf_counter()
    try:
        async with sema:
            r, w = await asyncio.wait_for(asyncio.open_connection(host, port), TIMEOUT)
            w.close()
            await w.wait_closed()
            return port, True, (time.perf_counter()-start)*1000
    except:
        return port, False, (time.perf_counter()-start)*1000
async def main():
    sema = asyncio.Semaphore(CONCURRENCY)
    tasks = [asyncio.create_task(try_connect(sema, TARGET, p)) for p in PORTS]
    results = await asyncio.gather(*tasks)
    openp = sorted(p for p, ok, _ in results if ok)
    elapsed = sum(r[2] for r in results)  # not perfect but cheap
    print(f"Scanned {len(results)} ports, open: {openp}, total elapsed (approx): {elapsed:.0f} ms")
if __name__ == "__main__":
    start=time.time(); asyncio.run(main()); print("Wall:", time.time()-start)
```

B: Local password-audit demo (safe)

```
# pw_audit.py — local password check (lab-only)
import hashlib
WEAK = ["123456","password","qwerty"]
DB = {"bob": hashlib.sha256(b"password").hexdigest(), "alice": hashlib.sha256(b"strong").hexdigest()}
found=[]
for user,h in DB.items():
    for pw in WEAK:
        if hashlib.sha256(pw.encode()).hexdigest()==h:
            found.append((user,pw))
if found:
    print("Weak pw found:", found)
else:
    print("No weak passwords found")
```

Output



```
C:\Users\Admin\AppData\Local\Programs\Python\Python313>python c.py
Weak pw found: [('bob', 'password')]

C:\Users\Admin\AppData\Local\Programs\Python\Python313>python a.py
Scanned 1024 ports, open: [135, 445], total elapsed (approx): 1569393 ms
Wall: 3.044677972793579
```

Practical 10

Aim: Develop a simple Intrusion Detection System (IDS) using Python.

Theory:

IDS

- Captures live packets (requires libpcap / Npcap).
- Maintains sliding-window stats keyed by source IP.
- Detects:
  - Port scan: many different destination ports from same source in short time window.
  - SYN flood: many TCP SYNs from same source with few completions.
  - High packet rate: excessive packets from a source.
- Logs alerts and prints them to console (could be extended to email/SIEM).

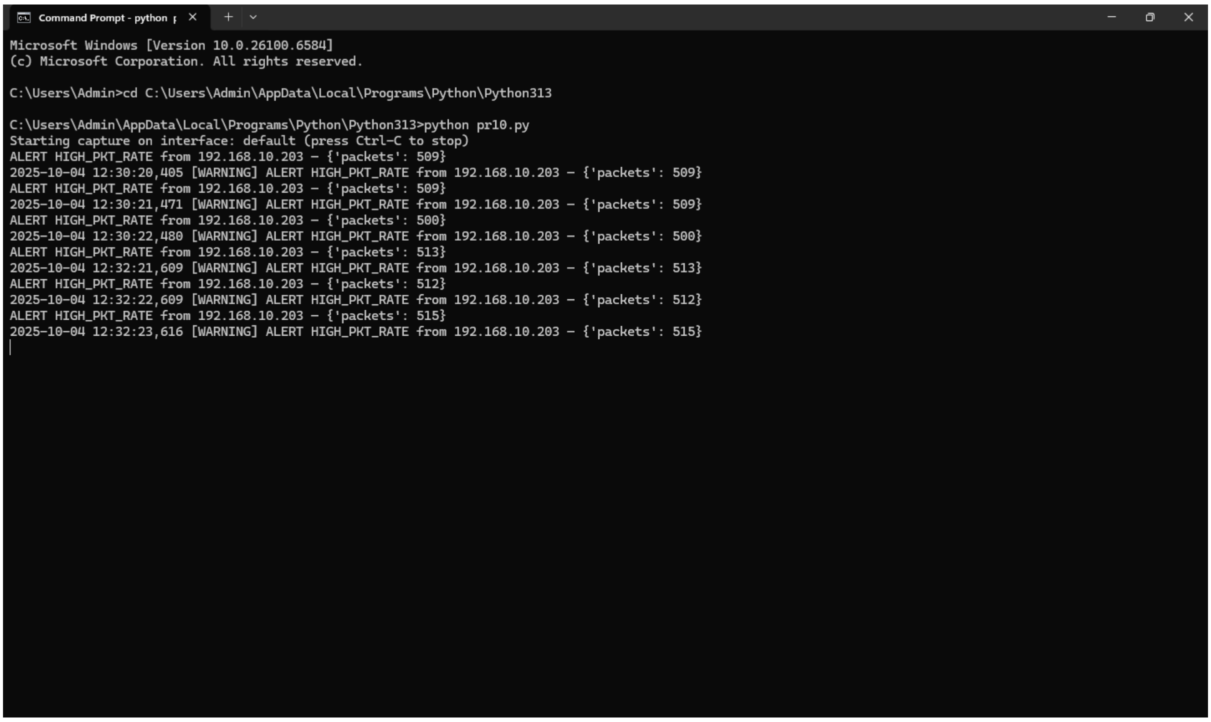
Code

```
#!/usr/bin/env python3
"""
simple_ids.py — a small, lab-only IDS demo.

Features:
 - Packet capture via scapy
 - Sliding-window counters per source IP
 - Detects: port scans, SYN floods, high packet rates
 - Uses an analyzer thread to avoid blocking packet capture

Run as root / admin:
  sudo python3 simple_ids.py
"""
import time
import threading
import queue
from collections import defaultdict, deque, Counter
from dataclasses import dataclass
import logging
import sys

# Scapy import (requires libpcap / Npcap)
try:
    from scapy.all import sniff, IP, TCP, UDP, ICMP
except Exception as e:
    print("Scapy import failed:", e)
    print("Make sure scapy is installed and libpcap/npcap is available.")
    sys.exit(1)

# ----- Configuration -----
CAPTURE_INTERFACE = None          # None = default, or "eth0", "wlan0", etc.
PACKET_QUEUE_SIZE = 10000
ANALYSIS_INTERVAL = 1.0           # seconds between analysis passes
WINDOW_SECONDS = 10               # sliding window size for detections
PORTSCAN_PORT_THRESHOLD = 20      # distinct dst ports in WINDOW => port scan
SYN_FLOOD_SYN_THRESHOLD = 100     # SYNs in WINDOW => possible SYN flood
PKT_RATE_THRESHOLD = 500          # packets from same src in WINDOW => high rate

# Logging / alerting
```

```
logging.basicConfig(level=logging.INFO, format="%(asctime)s [%(levelname)s]
%(message)s")
logger = logging.getLogger("simple_ids")

# ----- Data structures -----
# We'll queue simple packet summaries from capture thread to analyzer thread
pkt_queue = queue.Queue(maxsize=PACKET_QUEUE_SIZE)

@dataclass
class PktSummary:
    ts: float
    src: str
    dst: str
    sport: int
    dport: int
    proto: str
    flags: str  # for TCP flags, else ""

# Per-src sliding window state
class SrcState:
    def __init__(self):
        # Deques store timestamps of events for sliding window
        self.pkts = deque()              # timestamps of packets
            self.dst_ports = defaultdict(deque)  # dst_port -> deque of timestamps (for distinct
count)
        self.tcp_syns = deque()          # timestamps of SYN attempts
        self.tcp_acks = deque()          # timestamps of ACKs (optional)

    def cleanup(self, now, window):
        # Remove older entries outside the window
        cutoff = now - window
        while self.pkts and self.pkts[0] < cutoff:
            self.pkts.popleft()
        # cleanup dst_ports entries (and remove keys with empty deque)
        for port in list(self.dst_ports.keys()):
            dq = self.dst_ports[port]
            while dq and dq[0] < cutoff:
                dq.popleft()
            if not dq:
                del self.dst_ports[port]
        while self.tcp_syns and self.tcp_syns[0] < cutoff:
            self.tcp_syns.popleft()
        while self.tcp_acks and self.tcp_acks[0] < cutoff:
            self.tcp_acks.popleft()

# Global per-src map and lock
src_map = defaultdict(SrcState)
src_lock = threading.Lock()

# ----- Capture callback -----
def capture_packet(pkt):
    """
    Called by scapy for every packet. We quickly summarize and push to queue.
    Keep this function fast and non-blocking.
    """
    if IP not in pkt:
        return
    ip = pkt[IP]
    src = ip.src
    dst = ip.dst
    sport = 0
    dport = 0
    proto = "OTHER"
    flags = ""

    if TCP in pkt:
        proto = "TCP"
        sport = pkt[TCP].sport
        dport = pkt[TCP].dport
        flags = pkt[TCP].sprintf("%flags%")
    elif UDP in pkt:
        proto = "UDP"
        sport = pkt[UDP].sport
        dport = pkt[UDP].dport
    elif ICMP in pkt:
        proto = "ICMP"

    summary = PktSummary(ts=time.time(), src=src, dst=dst, sport=sport, dport=dport, proto=proto, flags=flags)
    try:
        pkt_queue.put_nowait(summary)
    except queue.Full:
        # If the queue is full, drop packet (and optionally log once per interval)
        # Avoid flooding logs
        pass

# ----- Analyzer thread -----
def analyzer():
```

```
"""
Consume packet summaries and maintain sliding-window stats.
Periodically evaluate detection rules.
"""
last_eval = time.time()
dropped_log_ts = 0
while True:
    try:
        # Drain queue with short timeout so we periodically run detections
        try:
            ps = pkt_queue.get(timeout=ANALYSIS_INTERVAL)
            now = ps.ts
            with src_lock:
                state = src_map[ps.src]
                state.pkts.append(now)
                # record destination port usage for portscan detection
                if ps.dport:
                    state.dst_ports[ps.dport].append(now)
                # record TCP SYNs/ACKs
                if ps.proto == "TCP":
                    if "S" in ps.flags and "A" not in ps.flags:
                        state.tcp_syns.append(now)
                    if "A" in ps.flags:
                        state.tcp_acks.append(now)
            pkt_queue.task_done()
        except queue.Empty:
            # no packet arrived in interval; run evaluation
            now = time.time()
            # Periodic cleanup and detection
            if time.time() - last_eval >= ANALYSIS_INTERVAL:
                last_eval = time.time()
                run_detections(now)
        except Exception as e:
            logger.exception("Analyzer error: %s", e)


def run_detections(now):
    alerts = []
    to_delete = []
    with src_lock:
        for src, state in list(src_map.items()):
            state.cleanup(now, WINDOW_SECONDS)


            pkt_count = len(state.pkts)
            distinct_ports = len(state.dst_ports)
            syn_count = len(state.tcp_syns)
            ack_count = len(state.tcp_acks)


            # Detection rules
            if distinct_ports >= PORTSCAN_PORT_THRESHOLD and pkt_count >= PORTSCAN_PORT_THRESHOLD:
                alerts.append((src, "PORT_SCAN", {
                    "distinct_ports": distinct_ports,
                    "packets": pkt_count
                }))


            # Heuristic: many SYNs but few ACKs -> possible SYN flood or half-open connections
            if syn_count >= SYN_FLOOD_SYN_THRESHOLD and ack_count < (syn_count * 0.1):
                alerts.append((src, "SYN_FLOOD", {
                    "syns": syn_count,
                    "acks": ack_count
                }))

            if pkt_count >= PKT_RATE_THRESHOLD:
                alerts.append((src, "HIGH_PKT_RATE", {
                    "packets": pkt_count
                }))

            # If the source hasn't produced any packets in window, remove its state to save memory
            if pkt_count == 0 and distinct_ports == 0 and syn_count == 0:
                to_delete.append(src)

        # Remove idle src states
        for s in to_delete:
            del src_map[s]

    # Emit alerts (outside lock)
    for src, typ, info in alerts:
        emit_alert(src, typ, info)

def emit_alert(src, typ, info):
    # Simple console and logger alert
    msg = f"ALERT {typ} from {src} — {info}"
    print(msg)
    logger.warning(msg)
    # TODO: add email, webhook, or other alerting


# ----- Runner -----
def main():
    # Start analyzer thread
    t = threading.Thread(target=analyzer, daemon=True)
    t.start()
    iface = CAPTURE_INTERFACE
    print(f"Starting capture on interface: {iface or 'default'} (press Ctrl-C to stop)")
    try:
        # sniff is blocking; it will call capture_packet for each packet
        sniff(iface=iface, prn=capture_packet, store=False)
    except KeyboardInterrupt:
        print("Stopping...")
    except Exception as e:
        logger.exception("sniff failed: %s", e)


if __name__ == "__main__":
    main()
```

Output:



```
Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Admin>cd C:\Users\Admin\AppData\Local\Programs\Python\Python313

C:\Users\Admin\AppData\Local\Programs\Python\Python313>python pr10.py
Starting capture on interface: default (press Ctrl-C to stop)
ALERT HIGH_PKT_RATE from 192.168.10.203 - {'packets': 509}
2025-10-04 12:30:20,405 [WARNING] ALERT HIGH_PKT_RATE from 192.168.10.203 - {'packets': 509}
ALERT HIGH_PKT_RATE from 192.168.10.203 - {'packets': 509}
2025-10-04 12:30:21,471 [WARNING] ALERT HIGH_PKT_RATE from 192.168.10.203 - {'packets': 509}
ALERT HIGH_PKT_RATE from 192.168.10.203 - {'packets': 500}
2025-10-04 12:30:22,480 [WARNING] ALERT HIGH_PKT_RATE from 192.168.10.203 - {'packets': 500}
ALERT HIGH_PKT_RATE from 192.168.10.203 - {'packets': 513}
2025-10-04 12:32:21,609 [WARNING] ALERT HIGH_PKT_RATE from 192.168.10.203 - {'packets': 513}
ALERT HIGH_PKT_RATE from 192.168.10.203 - {'packets': 512}
2025-10-04 12:32:22,609 [WARNING] ALERT HIGH_PKT_RATE from 192.168.10.203 - {'packets': 512}
ALERT HIGH_PKT_RATE from 192.168.10.203 - {'packets': 515}
2025-10-04 12:32:23,616 [WARNING] ALERT HIGH_PKT_RATE from 192.168.10.203 - {'packets': 515}
```