Cybersecurity Task 1 – Basic Network Sniffer

Comprehensive step-by-step report and explanation

Short Description (350 chars)

This task uses Python and Scapy to build a basic network sniffer that captures live packets, showing protocol, IP addresses, ports, size, and payload. It helps in learning network monitoring, traffic analysis, and detecting suspicious activity—an essential cybersecurity skill.

Optional Extended Description (4-5 Steps)

1. Install Python and the Scapy library.
2. Write the network sniffer code in a .py file (sniffer.py).
3. Run the script to capture live network packets (python sniffer.py).
4. View packet details like protocol, IPs, ports, size, and payload.
5. Analyze captured data to detect issues, suspicious traffic, or learn protocols.

Full Step-by-Step Guide (each step explained)

Prerequisites
• A Windows (or Linux/macOS) machine with administrator access.
• Python installed (recommend Python 3.8+). Check with `python --version`.
• Internet access to install Scapy via pip.

1. Open Command Prompt / Terminal
• On Windows: Press Start and search 'cmd' or open PowerShell.
• On macOS/Linux: Open Terminal from Applications.
• You will type commands here to navigate folders and run the sniffer.

2. Navigate to Your Project Folder
• Use `cd` (change directory) to move to the folder containing your script. Example:
`cd Desktop` moves you into the Desktop folder of the current user.
• Use `dir` (Windows) or `ls` (macOS/Linux) to list files and confirm your script is present.

3. Install Scapy
• Install Scapy using pip:
`pip install scapy`
• If permission errors occur, add `--user` or run the command in an elevated terminal.

4. Create the Python Script (sniffer.py)
• Create a file named `sniffer.py` and paste the sniffer code (example below).

5. The Example Sniffer Code (complete)
Copy this into sniffer.py. It uses Scapy to capture packets and print details. This is the full code used in this report:

```
from scapy.all import sniff, IP, TCP, UDP, ICMP
from datetime import datetime

def process_packet(packet):
    # Check for IP layer
    if IP in packet:
        # Determine protocol
        if TCP in packet:
            proto = "TCP"
            ports = f"{packet[TCP].sport} → {packet[TCP].dport}"
        elif UDP in packet:
            proto = "UDP"
            ports = f"{packet[UDP].sport} → {packet[UDP].dport}"
        elif ICMP in packet:
            proto = "ICMP"
            ports = ""
        else:
            proto = packet[IP].proto
```

```
            ports = ""
        src_ip = packet[IP].src
        dst_ip = packet[IP].dst
    else:
        proto = "OTHER"
        ports = ""
        src_ip = packet.summary()
        dst_ip = ""

    # Print packet details
    print(f"Time: {datetime.now().strftime('%H:%M:%S')}")
    print(f"Protocol: {proto}")
    print(f"Source IP: {src_ip}")
    print(f"Destination IP: {dst_ip}")
    print(f"Packet Size: {len(packet)} bytes")
    if ports:
        print(f"Ports: {ports}")

    # Try to show payload as ASCII if possible
    try:
        payload = bytes(packet.payload)
        printable = payload.decode('ascii', errors='ignore')
        if printable.strip():
            print(f"Payload (ASCII): {printable}")
    except Exception:
        pass

    print("-" * 60)

if __name__ == "__main__":
    # Start sniffing; requires appropriate permissions
    sniff(prn=process_packet, store=0)
```

Line-by-line Explanation

from scapy.all import ... — Imports Scapy functions for sniffing and common protocol layers (IP, TCP, UDP, ICMP).
from datetime import datetime — Imports datetime to timestamp each captured packet for human-readable logs.
def process_packet(packet): — Defines a function called for every captured packet. `packet` is the Scapy packet object.
if IP in packet: — Checks whether the captured packet contains an IP layer; many network packets will.
if TCP in packet: ... — Detects if the packet uses TCP and extracts source/destination ports.
elif UDP in packet: ... — Detects UDP and extracts ports similarly.
elif ICMP in packet: — ICMP has no ports; used for ping and network control messages.
print(f"Time: ...") — Prints the current time when the packet was processed.
print(f"Packet Size: {len(packet)} bytes") — Shows the full length in bytes of the captured packet.
payload decode block — Attempts to convert the packet payload to ASCII; ignores non-printable bytes.
sniff(prn=process_packet, store=0) — Starts packet capture and calls process_packet() for each packet; `store=0` prevents memory accumulation.

How to Run the Sniffer

1. Open Command Prompt / Terminal.
2. Navigate to the script folder: `cd Desktop` (or where your sniffer.py is located).
3. Confirm file exists: `dir` (Windows) or `ls` (macOS/Linux).
4. Run the script: `python sniffer.py`.
5. Stop the script with Ctrl+C to end sniffing and return to the prompt.

Output Explanation (Short)

When running, the sniffer prints one block per captured packet with the following fields:
- Time: Timestamp when processed.
- Protocol: TCP, UDP, ICMP, or OTHER.
- Source IP / Destination IP: Source and target addresses.
- Ports: Source → Destination ports (if applicable).

- Packet Size: Packet length in bytes.
- Payload (ASCII): Human-readable payload portion if present; otherwise may show symbols or be empty.

Interpreting Example Packets (what to look for)

Port 80 or 443 involved — Likely web (HTTP/HTTPS) traffic. Port 80 is HTTP (unencrypted); 443 is HTTPS (encrypted — payload not readable).
Large packet sizes — May indicate file downloads, media streaming, or large responses from servers.
Unknown external IPs — If you see repeated connections to unknown public IPs, investigate as they could be suspicious.
Repeated failed connection attempts — May indicate port scanning or brute-force attempts from an attacker.

Saving & Logging Packets

To save captured packet details to a file, modify process_packet() to append lines to a log file, for example using Python's open(..., 'a') to write each packet's block to disk.

Troubleshooting Common Errors

Permission error — On some OSes, raw packet capture requires elevated privileges. Run terminal as administrator (Windows) or use sudo (Linux/macOS).
Scapy import error — Ensure Scapy installed (`pip install scapy`). Check Python environment in use (system vs venv).
No packets captured — Check correct network interface, ensure there's network traffic, and disable firewall if blocking local capture. On Windows, consider WinPcap/Npcap installation for certain features.

Security, Ethics & Precautions

• Only sniff networks you own or have explicit permission to analyze.
• Sniffing can capture sensitive information; ensure secure handling and storage.
• Use encryption (HTTPS/VPN) to protect private data. If you find sensitive data accidentally, report it to the system owner.
• Malicious sniffing (without permission) is illegal and unethical.

Embedding a Quiz on Google Sites (summary)

1. Create the quiz in Google Forms and set it as a quiz (Settings → Make this a quiz).
2. Add your questions and correct answers. Example question included in Appendix.
3. In Forms, click Send → Embed (< >) and copy the embed code or link.
4. In Google Sites, choose Insert → Embed → paste the link or code and Insert.
5. Resize and Publish the site (Publish button).

Appendix A — Sample Quiz Question

Q: What is the main purpose of a network sniffer tool?
A. To speed up internet connection B. To block phishing websites C. To capture and analyze network traffic (Correct) D. To remove computer viruses

Appendix B — Command Prompt Cheat Sheet

cd Desktop — change directory to Desktop
dir — list files/folders in current directory (Windows)
ls — list files/folders (macOS/Linux)
python --version — check Python version
pip install scapy — install Scapy library

python sniffer.py — run the sniffer script
Ctrl+C — stop the running script

Report created to guide you step-by-step through building, running, and analyzing a basic network sniffer using Python and Scapy. If you want a version formatted for printing, or an editable DOCX, tell me and I will provide it.