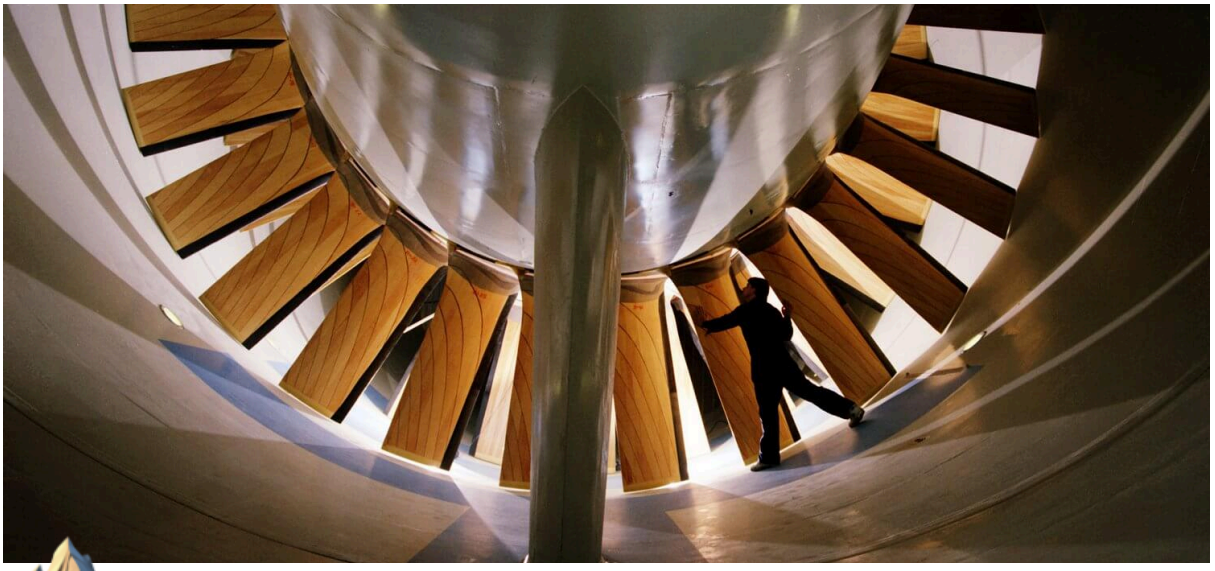


Snort (IDS)

Generative AI for Creation of Effective Signatures for Short Intrusion Detection Systems (IDS)

Day-1 (19/5)

What IDS

- A system that identifies unauthorised access by hackers and malicious actors
- Once a behavioural anomaly is detected, the security admin is warned
- Automated systems are configured according to one’s needs and requirements.

Types of Intruders

- Attacker
- insider

Ways to Detect Intrusion

- Signatures-Based
- Anomaly Detection
- hybrid based

Types of IDS

- Network-based IDS →NIDS monitors the traffic flow from various areas of the network. The aim is to investigate the traffic on the entire subnet. If a signature is identified, an alert is created.
- Host-Based IDS → HIDS monitors the traffic flow from a single endpoint device. The aim is to investigate the traffic on a particular device. If a signature is identified, an alert is created.
- cloud based

DetectionTechniquess

- Signature-Based
- Behaviour-Based (This)
- Policy-Based

Best IDS Tools

- Solarwinds security event manager
- MCAfee Live safe
- Blumaira

Day-2(20/5)

snort

- open-source network IDS & IPS
- Detects malicious network traffic or attacks

Use cases

real-time network traffic monitoring

protocol analysis

content matching(based on rules)

operating system (os) fingerprinting

compadility with any os

Capabilities of Snort;

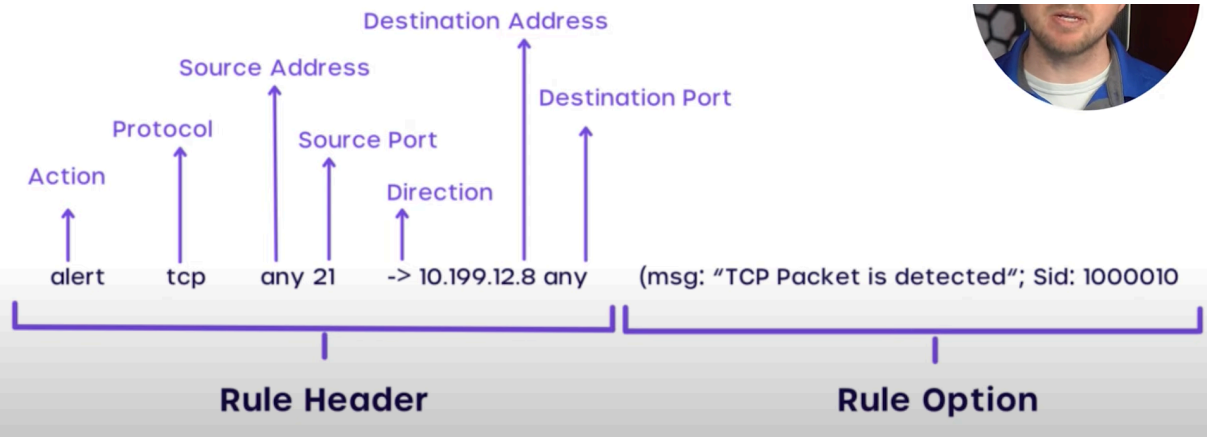
- Live traffic analysis
- Attack and probe detection
- Packet logging
- Protocol analysis
- Real-time alerting
- Modules & plugins
- Pre-processors
- Cross-platform support! (Linux & Windows)

SNORT Modes/flags

- sniffer mode (-v)
- packet logger mode (-l)
- network IDS and IPS mode (-c)

Rule sources

- community rules
- Registered rules
- subscription rules
- Build your own Rules



Snort rule types

- Alert rules
- block rules
- Drop rules
- logging rules
- pass rules

Rule syntax

- Action
- ip add
- port numbers
- Direction of traffic(→ or <>)
- Inspection protocol

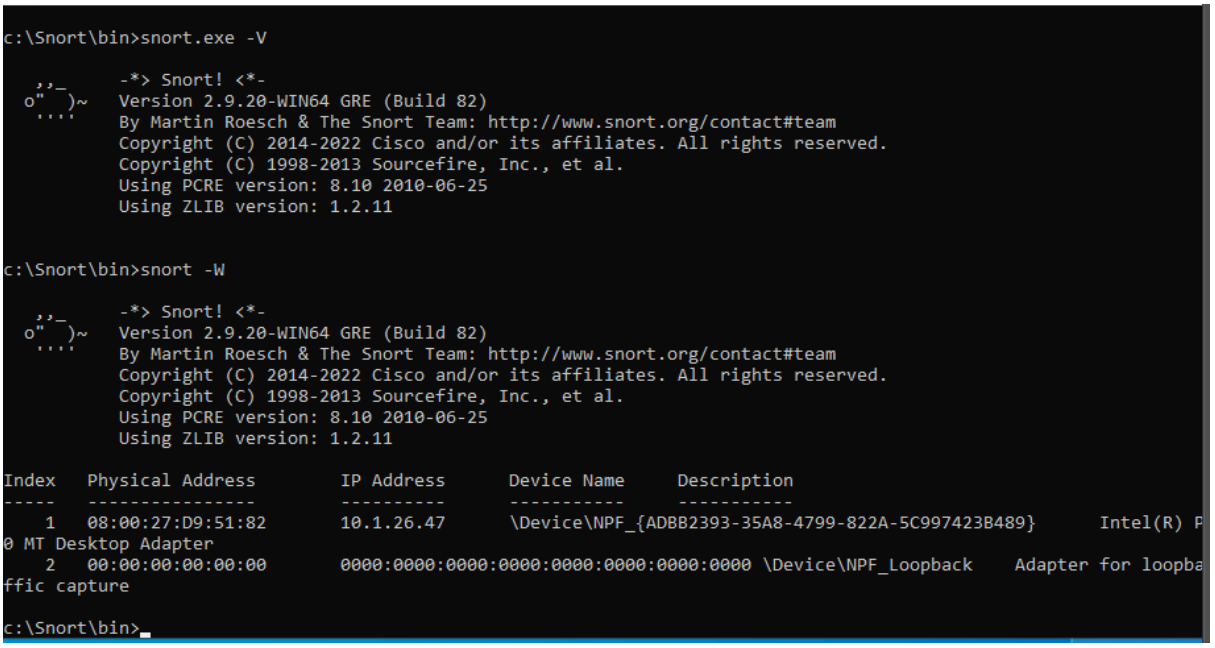
Rule option

- Message to display when a rule matches
- Flow state
- content or pattern
- Service or application protocol
- snort ID (sid) and revision number(rev)

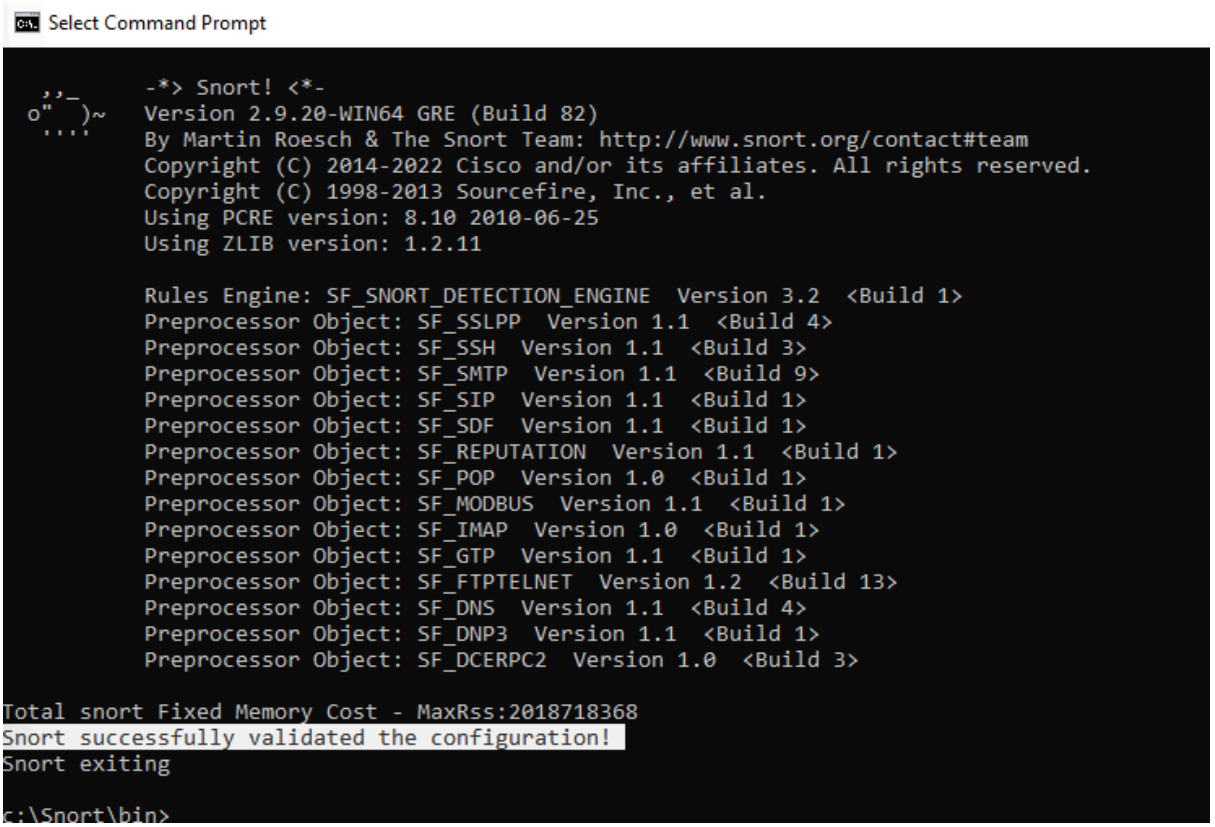
Day-3(21/5)

Run snort

cd c:\Snort\bin
dir
snort.exe -V
snort -W



snort -i 1 -c c:\Snort\etc\snort.conf -T
-i → interface
-c → identifying the config file
-T → used for test config



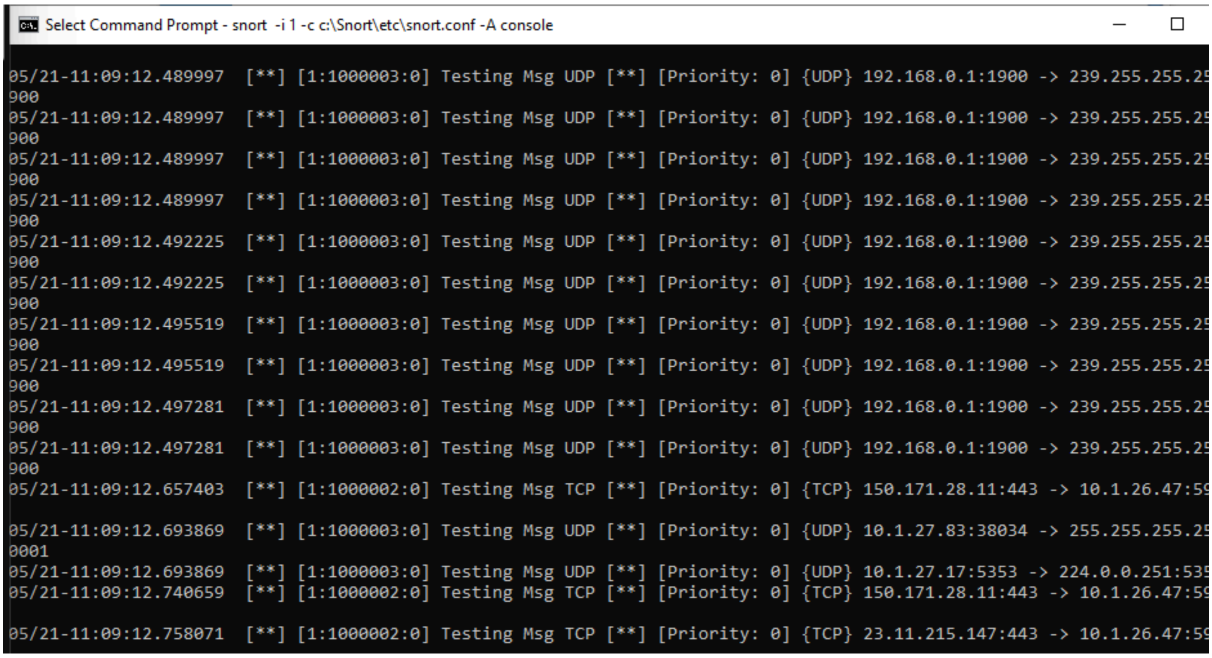
Parameter	Description
-V / --version	This parameter provides information about your instance version.
-c	Identifying the configuration file
-T	Snort's self-test parameter, you can test your setup with this parameter.
-q	Quiet mode prevents snort from displaying the default banner and initial information about your setup.

Snort run alerts

at ⇒ c:\Snort\rules\local.rules

```
alert icmp any any → any any (msg:"Testing Msg ICMP";sid:1000001;)
alert tcp any any → any any (msg:"Testing Msg TCP";sid:1000002;)
alert udp any any → any any (msg:"Testing Msg UDP";sid:1000003;)
```

```
--------------------------------------------------------------------
Command prompt
snort -i 1 -c c:\Snort\etc\snort.conf -A console
```



---

reference:

- https://youtu.be/e_KV3xivSnI?si=-IsA0rIhgR-inOWA
- https://youtu.be/4Z7ii2aI-xQ?si=aVcjnxYoVsPsq7EC
- https://youtu.be/ZEdfnXt-ptM?si=Ftj6sLH7DLSXPDr8

Tryhackme

https://tryhackme.com/room/snort

Sniffer Mode

Parameter	Description
-v	Verbose. Display the TCP/IP output in the console.
-d	Display the packet data (payload).
-e	Display the link-layer (TCP/IP/UDP/ICMP) headers.
-X	Display the full packet details in HEX.

-i	This parameter helps to define a specific network interface to listen/sniff. Once you have multiple interfaces, you can choose a specific interface to sniff.

Day-4(22/5)

Setting Up My Own Private AI Chatbot.pdf

🛡️ Day 5: Saving Snort IDS Alerts into MySQL using Python

Explanation:

We are building a **pipeline** where Snort (an alert system) watches network activity, finds attacks, and stores those alerts into a **MySQL database** using **Python**.

🔧 STEP 1: Set Up the Database

✅ Install XAMPP (Includes MySQL)

1. Go to 👉 https://www.apachefriends.org
2. Download **XAMPP for Windows**
3. Install it → Make sure you **check MySQL** during installation
4. Open the **XAMPP Control Panel**
   - Click **Start** on both `Apache` and `MySQL`
   - Click **Admin** next to MySQL → this opens phpMyAdmin in your browser

---

🛠️ If phpMyAdmin doesn't open, try this:

- Check in browser:

  👉 `http://localhost/phpmyadmin`

- If that fails:

- Change Apache port from 80 → 8080:
  - In XAMPP, click `Config` next to Apache → open `httpd.conf`
  - Change:

    ```
    isten 80 → Listen 8080
    ServerName localhost:80 → ServerName localhost:8080
    ```

  - Restart Apache
  - Now open 👉 `http://localhost:8080/phpmyadmin`

🗃️ STEP 2: Create a Database to Store Snort Alerts

1. In phpMyAdmin, click **Databases**
2. Under "Create database", type:

   👉 `snort_alerts`
3. Leave collation as `utf8mb4_general_ci`
4. Click **Create**

Now, the database is ready 🎉

📋 STEP 3: Create a Table in the Database

Click the `snort_alerts` database on the left

Then:

1. Table name: `alerts`
2. Number of columns: `7`
3. Click **Go**

Fill the table like this:

Column Name	Type	Extra
`id`	INT	AUTO_INCREMENT, PRIMARY KEY
`timestamp`	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP
`signature`	VARCHAR(255)	—
`priority`	INT	—

src_ip	VARCHAR(45)	—
dst_ip	VARCHAR(45)	—
message	TEXT	—

Then click **Save** ✅

Click **SQL** tab and run this:

```
CREATE TABLE alerts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    signature VARCHAR(255),
    priority INT,
    src_ip VARCHAR(45),
    dst_ip VARCHAR(45),
    message TEXT
);
```

✅ This will create the `alerts` table with all necessary fields.

🐍 STEP 4: Python Script to Push Alerts into Database

First install the MySQL connector for Python:

```
install mysql-connector-python
```

Paste this script and save it as `snort_to_sql.py` :

```
import mysql.connector
import os

# Connect to MySQL
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",  # Add password if you set one
```

```
    database="snort_alerts"
)
cursor = conn.cursor()

# Define the alert file location
alert_file_path = "C:\\Snort\\log\\alert.fast"

# Check if the file exists
if not os.path.exists(alert_file_path):
    print("❌ alert.fast file not found.")
    exit()

# Read file content
with open(alert_file_path, "r") as file:
    lines = file.readlines()

# Parse each alert
for i in range(len(lines)):
    line = lines[i]
    if "[**]" in line and "Testing Msg" in line:
        parts = line.split()
        signature = parts[2] + " " + parts[3]
        message = " ".join(parts[4:])
        priority = 1
        src_ip, dst_ip = "0.0.0.0", "0.0.0.0"

        for j in range(i+1, min(i+6, len(lines))):
            if "→" in lines[j]:
                ip_line = lines[j].strip()
                ip_parts = ip_line.split("→")
                src_ip = ip_parts[0].split()[-1]
                dst_ip = ip_parts[1].strip()
                break

        # Insert into MySQL
        cursor.execute("""
            INSERT INTO alerts (signature, priority, src_ip, dst_ip, message)
            VALUES (%s, %s, %s, %s, %s)
```

```
        """, (signature, priority, src_ip, dst_ip, message))

conn.commit()
cursor.close()
conn.close()
print("✅ Alerts inserted into MySQL!")
```

📦 STEP 5: Set Up Snort for Logging

Open this file:

```
C:\Snort\etc\snort.conf
```

📌 At the **very bottom**, add this line:

```
output alert_fast: alert.fast
```

✅ This makes Snort save alerts in a file called `alert.fast`

🚀 STEP 6: Run Snort + Trigger Alerts

🏃 Run Snort like this:

```
snort -i 1 -c C:\Snort\etc\snort.conf -l C:\Snort\log
```

This will:

- Use interface `1`
- Use your config file
- Log alerts in the folder `C:\Snort\log`

🧪 Trigger Alerts

From Kali or another system, run:

```
ping [victim IP]
```

or

```
nmap -A [victim IP]
```

✅ This should trigger Snort alerts!

🧠 STEP 7: Run the Python Script to Insert Alerts

Now go back to Command Prompt:

```
python snort_to_sql.py
```

🎉 All alerts from `alert.fast` will now be inserted into the `alerts` table inside MySQL.

📊 STEP 8: View the Data in phpMyAdmin

1. Open 👉 `http://localhost:8080/phpmyadmin`
2. Click `snort_alerts` → Click `alerts`
3. Click **Browse**

Boom 💣 – you'll see each alert logged with:

- Signature
- Source IP
- Destination IP
- Message
- Timestamp

Day 6: 26/5

Drop and Redirect

🔐 What is "Drop"?

🧠 Definition:

Dropping a packet means:

> ❌ Silently blocking a packet before it reaches its destination.

The attacker sends traffic — but it **disappears mid-way** like a ghost 👻

The destination never receives it.

🛠️ Example:

Attacker IP	→	Victim IP
`192.168.1.10`	Ping →	`192.168.1.20`

If Snort detects this as malicious, it **drops** the packet:

> The Victim never even receives or responds to it.

✅ In Snort (Linux only):

```
drop icmp any any -> $HOME_NET any (msg:"Ping dropped"; sid:1000001;)
```

- `drop` = don't allow the packet to pass
- This only works in **inline/IPS** mode (not available on Windows)

🔀 What is "Redirect"?

🧠 Definition:

Redirection means:

> 🧭 Changing the path of a packet to a different target.

The attacker **thinks they're talking to the victim**,

but they're actually talking to a **decoy (fake)** system.

🛠️ Example:

Attacker sends to	→	Victim (Snort detects)	→	Redirect to Fake IP
`192.168.1.10` →	Ping	`192.168.1.20`	→	`192.168.1.99`

Snort sees the packet, and a rule/script changes the route, so the attacker gets a **reply from a fake system**.

✅ Real-life Use:

- Honeypots
- Deception networks
- Malware sinkholes

🎯 Summary Table:

Action	Goal	Visible to Attacker?	Native in Snort?
Drop	Block harmful packet	❌ No response	✅ (only in IPS mode)
Redirect	Divert to decoy system	✅ Gets reply (fake)	❌ Needs external tools

---

🧠 Why Snort on Windows Can't Do This Alone?

- No **inline/bridge mode** on Windows = can't drop live packets
- No **packet routing control** like iptables
- So you use **Snort to detect**, and **Windows firewall + route commands to act**

Snort Intrusion Detection with Automatic IP Blocking on Windows

Goal:

To **automatically detect ICMP (ping) attacks** using **Snort**, and **block the attacker's IP** using **Windows Firewall** via a custom Python script.

---

1. Snort Setup on Windows

Installed:

- Snort 2.9.20 for Windows (64-bit)
- Snort configured with:

- `snort.conf` file at: `C:\Snort\etc\snort.conf`
- Logging directory: `C:\Snort\log`

✅ Custom Snort Rule (inside `local.rules`):

```
alert icmp any any -> $HOME_NET any (msg:"[DETECTED] ICMP Ping"; sid:1000001;)
```

This rule detects incoming ICMP (ping) packets.

⚙️ 2. Snort Config Update

✅ Edited `snort.conf` to include:

```
include $RULE_PATH/local.rules
output alert_fast: alert.ids
```

This tells Snort to:

- Load your custom rules
- Log alerts to `C:\Snort\log\alert.ids`

▶️ 3. Running Snort

Snort is started using:

```
snort -i 1 -c C:\Snort\etc\snort.conf -l C:\Snort\log
```

- `i 1`: listen on interface 1
- `c`: config file
- `l`: log directory

💻 4. Create Python Script for Dropping IPs

✅ Script: `drop_attacker.py`

This script:

- Monitors `alert.ids`
- Extracts source IP from logs
- Blocks IP using `netsh advfirewall` commands

🧠 What It Uses:

- `os`, `time` → read files and wait
- `subprocess` → add firewall rules

🧾 5. Python Code: `drop_attacker.py`

```
import os
import time
import subprocess
import re

alert_file = "C:\\Snort\\log\\alert.ids"
blocked_ips = set()

print("🚨 DROP script is running...\n")

def extract_ips_from_alerts(content):
    ips = []
    for line in content:
        match = re.search(r'(\d{1,3}(?:\.\d{1,3}){3}) →', line)
        if match:
            ip = match.group(1)
            if ip not in blocked_ips:
                ips.append(ip)
    return ips

while True:
    if not os.path.exists(alert_file):
        print("⚠️ alert.ids not found. Waiting...")
        time.sleep(5)
        continue
```

```
with open(alert_file, "r") as file:
    lines = file.readlines()

new_ips = extract_ips_from_alerts(lines)

for ip in new_ips:
    try:
        print(f"🔒 Blocking IP: {ip}")
        subprocess.run([
            "netsh", "advfirewall", "firewall", "add", "rule",
            f"name=Block_{ip}", "dir=in", "action=block", f"remoteip={ip}"
        ], check=True)
        blocked_ips.add(ip)
    except Exception as e:
        print(f"⚠️ Error blocking {ip}: {e}")

print("\n📋 Blocked IPs so far:")
print("----------------------------------------")
for b_ip in blocked_ips:
    print(f"• {b_ip}")
print("----------------------------------------")

time.sleep(5)
```

🧪 6. Testing Procedure

1. Run Snort:

   ```
   snort -i 1 -c C:\Snort\etc\snort.conf -l C:\Snort\log
   ```

2. Run Python script as **Administrator**:

   ```
   python drop_attacker.py
   ```

3. From **attacker machine**, ping the victim:

   ```
   ping <victim IP>
   ```

4. Watch the script:
    - It will print:

```
🔒 Blocking IP: 192.168.1.10
📋 Blocked IPs so far:
  • 192.168.1.10
```

5. Ping again → it fails (IP is dropped by firewall) ✅

📦 Output Sample

```
🚨 DROP script is running...

🔒 Blocking IP: 192.168.1.10

📋 Blocked IPs so far:
  • 192.168.1.10
----------------------------------------
```

💡 Key Learnings

- Integrated Snort with Python
- Used `alert.ids` to detect attacks
- Blocked malicious IPs in real-time using Windows Firewall

✅ Technologies Used

Component	Tool/Technology
IDS	Snort (v2.9.20 for Windows)
Firewall	Windows Defender Firewall
Scripting	Python 3.13
OS	Windows 10 (Victim)
Attacker System	Kali Linux (VM)

Redirection

Attacker —▶ Victim Proxy (mirror) —▶ Honeypot
◀——————————————————◀

Deception Proxy for Cybersecurity Research (Documentation)

Overview

This project implements a **Deception Proxy** on a Windows-based "Victim" system to redirect all TCP traffic from an attacker to a honeypot. The victim acts as a transparent middleman (man-in-the-middle) and forwards all incoming packets to the honeypot, giving the attacker false feedback. This proxy does not depend on Snort logs and works in real-time using raw socket connections.

System Architecture

Participants:

- **Attacker**: Tries to scan or connect to victim
- **Victim**: Runs the deception proxy script
- **Honeypot**: Receives traffic and sends fake acknowledgments/responses

Network Layout:

```
Attacker (192.168.56.102)
      |
   [Proxy - Victim] (192.168.56.101)
      |
   Honeypot (192.168.56.103)
```

Prerequisites

- OS: Windows 10
- Python 3.9+

- All systems (Attacker, Victim, Honeypot) connected via **Host-Only Adapter** or same LAN

Python Modules Required:

- `socket` (Standard Library)
- `threading` (Standard Library)
- `logging` (Standard Library)

Setup Instructions

Step 1: Install Python

Download and install Python from:
https://www.python.org/downloads/windows/

Make sure to add it to PATH.

Step 2: Save and Run the Script

Save the script as **Redirectionr.py** on the victim system.

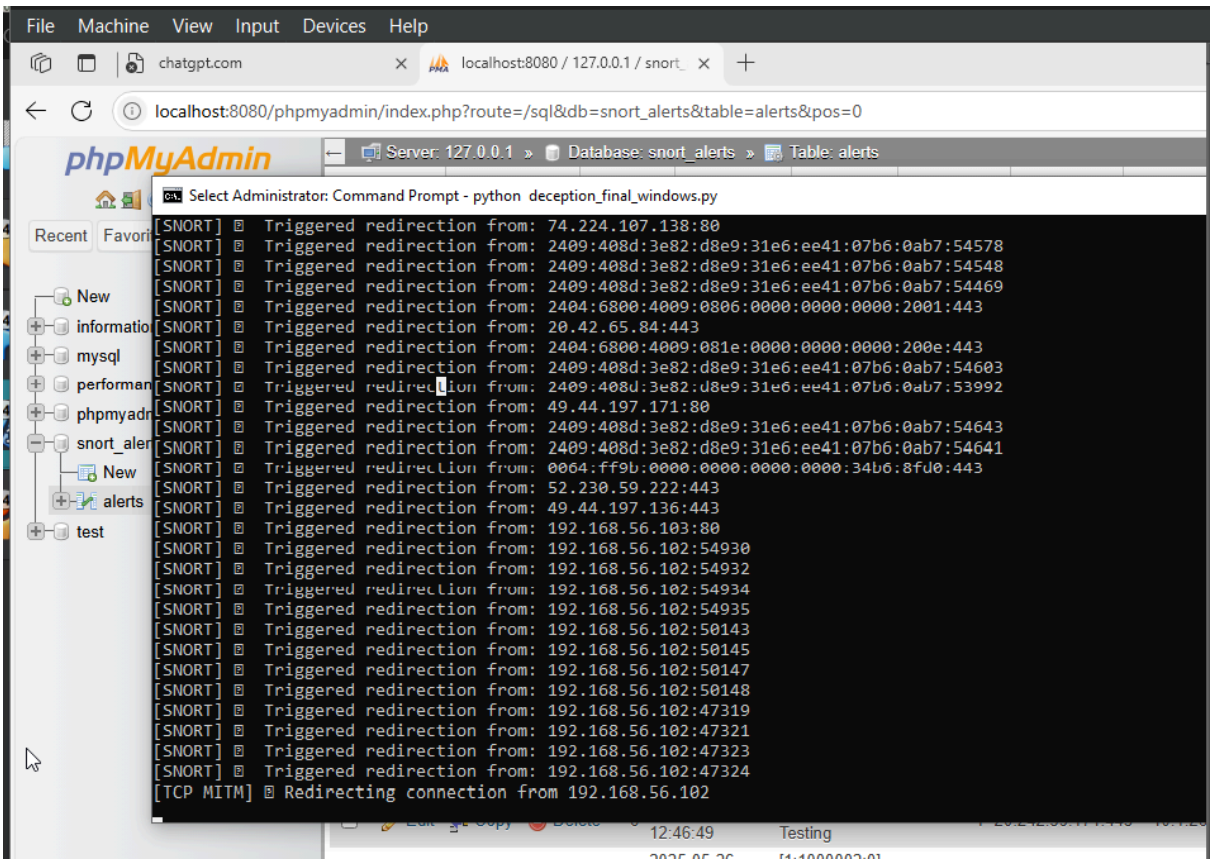
Run it using Administrator privileges:

```
import socket
import threading
import logging

# ==========================
# CONFIGURATION
# ==========================
LISTEN_IP = '0.0.0.0'  # Victim listens on all interfaces
LISTEN_PORT = 80       # Port to listen on (HTTP, can be changed)
HONEYPOT_IP = '192.168.56.103'
HONEYPOT_PORT = 80      # Port on honeypot (match service)

# ==========================
# LOGGING SETUP
# ==========================
logging.basicConfig(
    filename="deception_proxy.log",
```

```
    format='%(asctime)s - %(levelname)s - %(message)s',
    level=logging.INFO
)

print(f"🔁 Deception proxy started on port {LISTEN_PORT}, redirecting to {HO
logging.info("Proxy started: Listening on port %s, redirecting to honeypot at %

def handle_connection(client_socket):
    try:
        # Connect to honeypot
        honeypot_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM
        honeypot_socket.connect((HONEYPOT_IP, HONEYPOT_PORT))

        # Start forwarding in both directions
        threading.Thread(target=forward, args=(client_socket, honeypot_socket,
        threading.Thread(target=forward, args=(honeypot_socket, client_socket,

    except Exception as e:
        logging.error("Connection error: %s", e)
        client_socket.close()

def forward(source, destination, direction, ip):
    try:
        while True:
            data = source.recv(4096)
            if not data:
                break
            logging.info("Forwarding (%s) [%s]: %d bytes", direction, ip, len(data))
            destination.sendall(data)
    except Exception as e:
        logging.warning("Forwarding exception (%s) [%s]: %s", direction, ip, e)
    finally:
        source.close()
        destination.close()

# ==========================
# START LISTENING
# ==========================
```

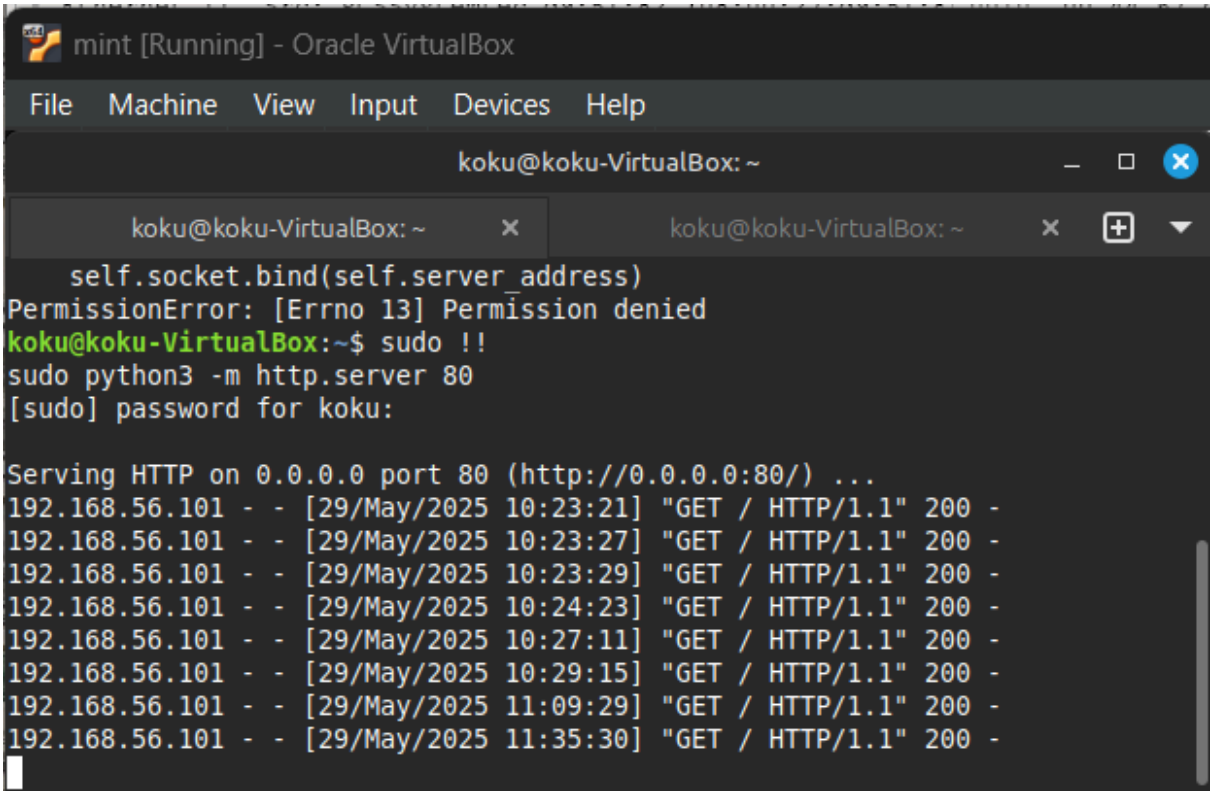
```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((LISTEN_IP, LISTEN_PORT))
server.listen(100)

while True:
    client_socket, addr = server.accept()
    print(f"🚨 New connection from {addr[0]}:{addr[1]}")
    logging.info("New connection from %s:%s", addr[0], addr[1])
    threading.Thread(target=handle_connection, args=(client_socket,)).start()
```

```
python deception_proxy.py
```



Step 3: Configure Honeypot

On the honeypot machine (e.g., Linux):

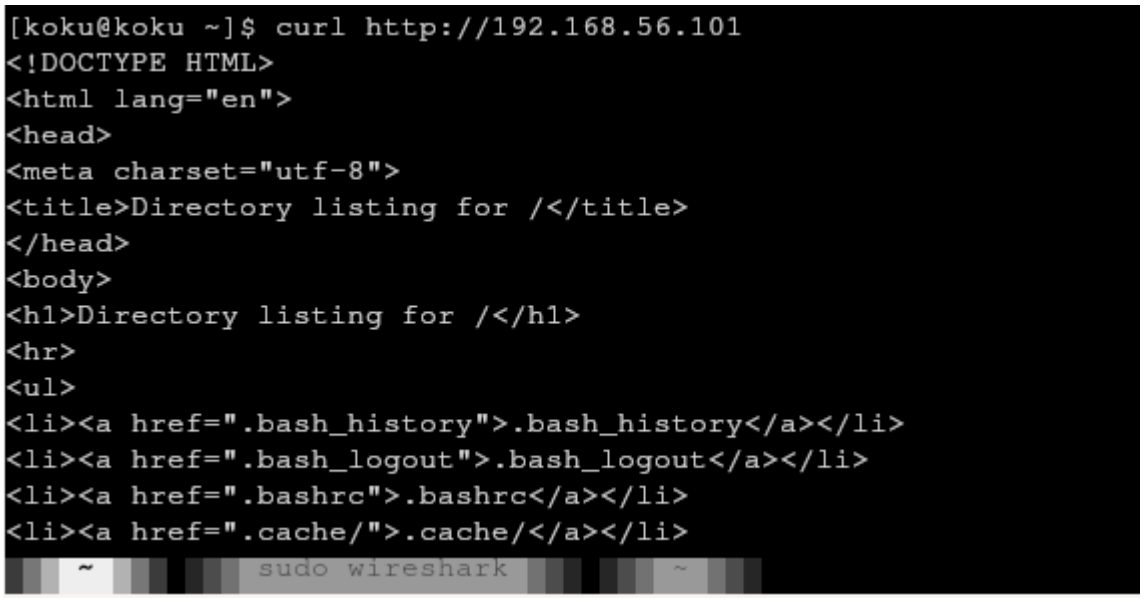



```
sudo python3 -m http.server 80
```

Or use a tool like Cowrie or Dionaea to simulate services.

Step 4: Test from Attacker

On attacker machine:



```
nmap -sV <victim_IP>
curl http://<victim_IP>
```

The response should come from honeypot, not victim.

How the Script Works

Configuration Block

```
LISTEN_IP = '0.0.0.0'      # Bind to all interfaces
LISTEN_PORT = 80           # HTTP traffic
HONEYPOT_IP = '192.168.56.103'
HONEYPOT_PORT = 80         # Honeypot service port
```

Logging

Logs all activity to a file `deception_proxy.log`, including:

- New connections
- Bytes forwarded in both directions
- Errors or warnings

`handle_connection()`

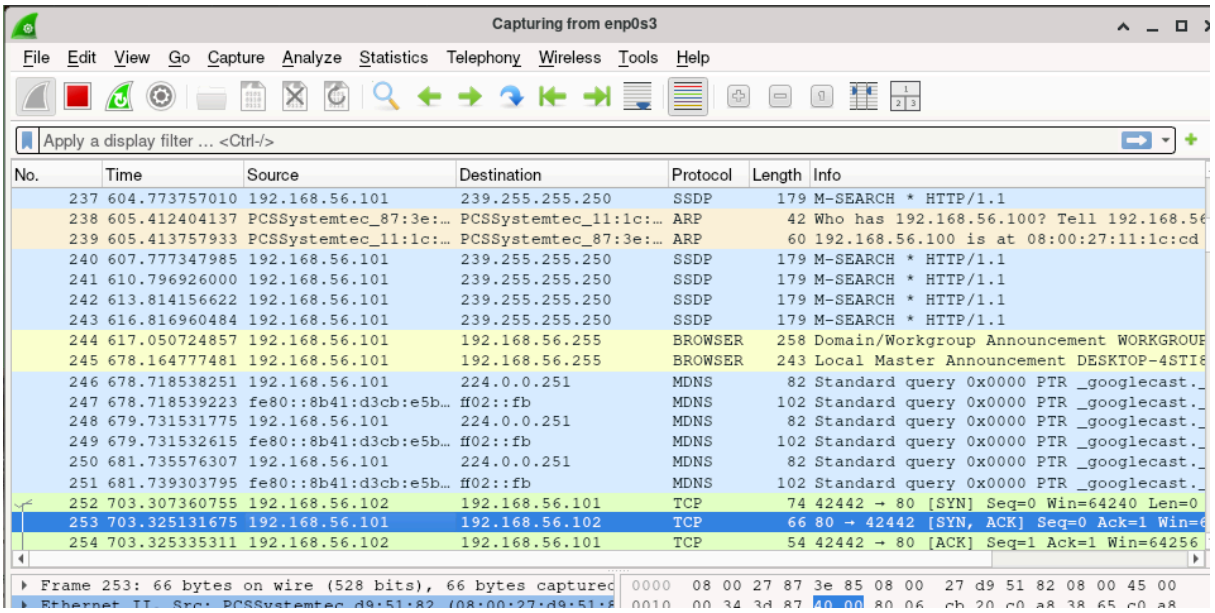
- Accepts attacker connection
- Opens a new connection to honeypot
- Starts two threads to forward traffic both ways

`forward()`

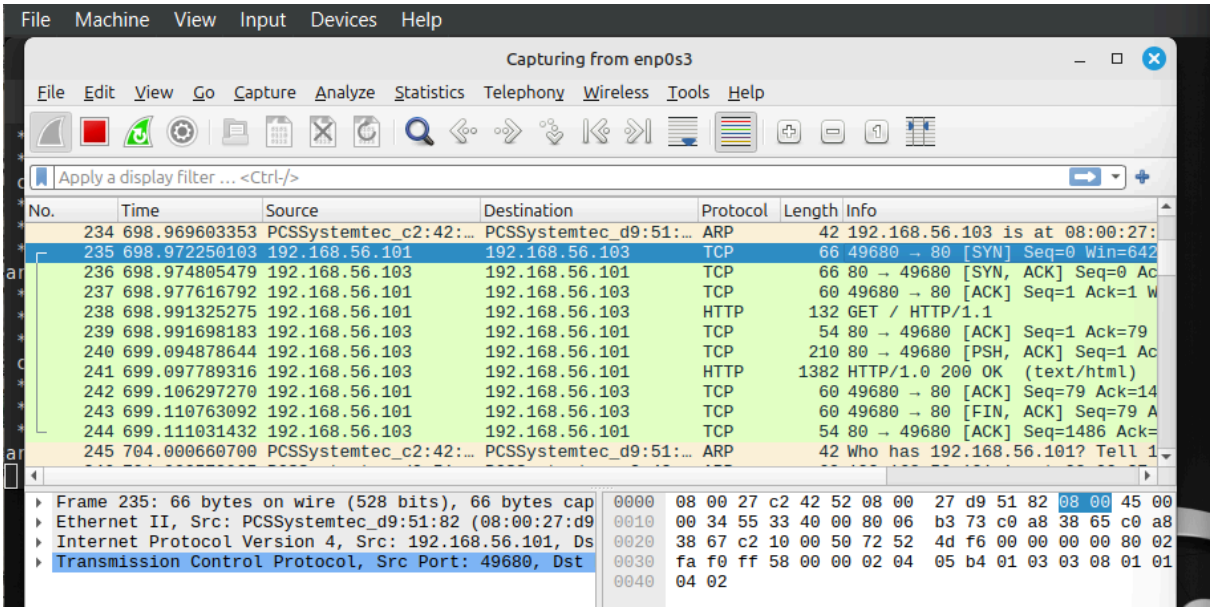
- Listens for packets from source
- Logs the data
- Sends packets to destination

Logs Example (deception_proxy.log)

```
2025-05-25 10:22:35 - INFO - New connection from 192.168.56.102:53422
2025-05-25 10:22:36 - INFO - Forwarding (Attacker → Honeypot) [192.168.
56.102]: 321 bytes
2025-05-25 10:22:36 - INFO - Forwarding (Honeypot → Attacker) [192.168.
56.102]: 158 bytes
```



Attacker



Honeypot

How to Verify It's Working

- Run `nmap` or `curl` from attacker
- Victim logs will show connection from attacker
- Honeypot will show received request
- Attacker will see honeypot response (e.g., fake HTTP banner)
