



# Snort (IDS)

## Generative AI for Creation of Effective Signatures for Snort 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;**

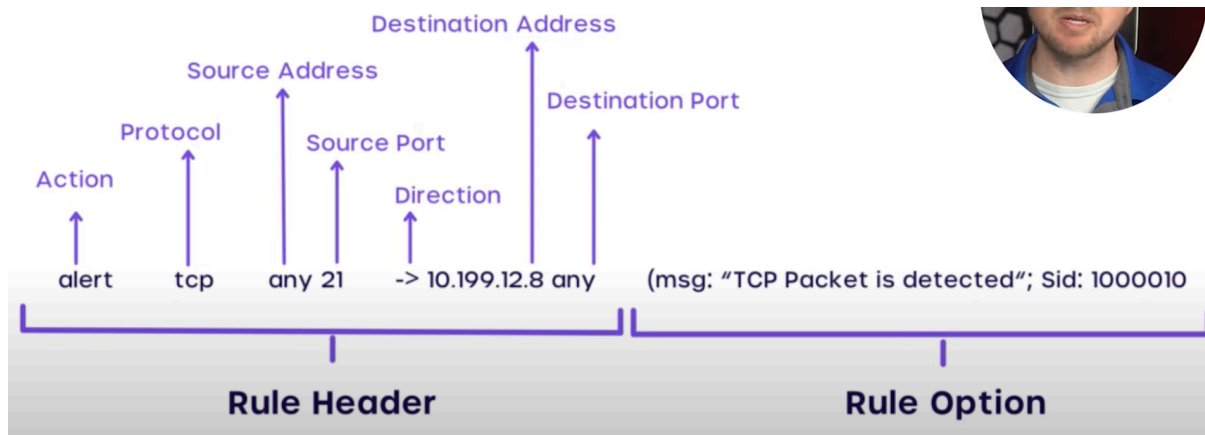
- Live traffic analysis
- Attack and probe detection
- Packet logging
- Protocol analysis
- Real-time alerting
- Modules & plugins
- Pre-processors
- Cross-platform support! (Linux & Windows)

## **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
```

```
c:\Snort\bin>snort.exe -V

  __-
o"  )~
  '~~~

-*> Snort! <*-
Version 2.9.20-WIN64 GRE (Build 82)
By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
Copyright (C) 2014-2022 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using PCRE version: 8.10 2010-06-25
Using ZLIB version: 1.2.11

c:\Snort\bin>snort -W

  __-
o"  )~
  '~~~

-*> Snort! <*-
Version 2.9.20-WIN64 GRE (Build 82)
By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
Copyright (C) 2014-2022 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using PCRE version: 8.10 2010-06-25
Using ZLIB version: 1.2.11

Index  Physical Address      IP Address      Device Name      Description
-----
1      08:00:27:D9:51:82      10.1.26.47      \Device\NPF_{ADBB2393-35A8-4799-822A-5C997423B489} Intel(R) P
0 MT Desktop Adapter
2      00:00:00:00:00:00      0000:0000:0000:0000:0000:0000:0000:0000 \Device\NPF_Loopback Adapter for loopba
ffic capture

c:\Snort\bin>
```

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

```

o''~
'''~

-*> Snort! <*-
Version 2.9.20-WIN64 GRE (Build 82)
By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
Copyright (C) 2014-2022 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using PCRE version: 8.10 2010-06-25
Using ZLIB version: 1.2.11

Rules Engine: SF_SNORT_DETECTION_ENGINE Version 3.2 <Build 1>
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>
Preprocessor Object: SF_SIP Version 1.1 <Build 1>
Preprocessor Object: SF_SDF Version 1.1 <Build 1>
Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1>
Preprocessor Object: SF_POP Version 1.0 <Build 1>
Preprocessor Object: SF_MODBUS Version 1.1 <Build 1>
Preprocessor Object: SF_IMAP Version 1.0 <Build 1>
Preprocessor Object: SF_GTP Version 1.1 <Build 1>
Preprocessor Object: SF_FTPTELNET Version 1.2 <Build 13>
Preprocessor Object: SF_DNS Version 1.1 <Build 4>
Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>
Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3>

Total snort Fixed Memory Cost - MaxRss:2018718368
Snort successfully validated the configuration!
Snort exiting
c:\Snort\bin>

```

Parameter	Description
<b>-V / --version</b>	This parameter provides information about your instance version.
<b>-c</b>	Identifying the configuration file
<b>-T</b>	Snort's self-test parameter, you can test your setup with this parameter.
<b>-q</b>	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

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

05/21-11:09:12.489997  [**] [1:1000003:0] Testing Msg UDP [**] [Priority: 0] {UDP} 192.168.0.1:1900 -> 239.255.255.25
900
05/21-11:09:12.489997  [**] [1:1000003:0] Testing Msg UDP [**] [Priority: 0] {UDP} 192.168.0.1:1900 -> 239.255.255.25
900
05/21-11:09:12.489997  [**] [1:1000003:0] Testing Msg UDP [**] [Priority: 0] {UDP} 192.168.0.1:1900 -> 239.255.255.25
900
05/21-11:09:12.489997  [**] [1:1000003:0] Testing Msg UDP [**] [Priority: 0] {UDP} 192.168.0.1:1900 -> 239.255.255.25
900
05/21-11:09:12.492225  [**] [1:1000003:0] Testing Msg UDP [**] [Priority: 0] {UDP} 192.168.0.1:1900 -> 239.255.255.25
900
05/21-11:09:12.492225  [**] [1:1000003:0] Testing Msg UDP [**] [Priority: 0] {UDP} 192.168.0.1:1900 -> 239.255.255.25
900
05/21-11:09:12.495519  [**] [1:1000003:0] Testing Msg UDP [**] [Priority: 0] {UDP} 192.168.0.1:1900 -> 239.255.255.25
900
05/21-11:09:12.495519  [**] [1:1000003:0] Testing Msg UDP [**] [Priority: 0] {UDP} 192.168.0.1:1900 -> 239.255.255.25
900
05/21-11:09:12.497281  [**] [1:1000003:0] Testing Msg UDP [**] [Priority: 0] {UDP} 192.168.0.1:1900 -> 239.255.255.25
900
05/21-11:09:12.497281  [**] [1:1000003:0] Testing Msg UDP [**] [Priority: 0] {UDP} 192.168.0.1:1900 -> 239.255.255.25
900
05/21-11:09:12.657403  [**] [1:1000002:0] Testing Msg TCP [**] [Priority: 0] {TCP} 150.171.28.11:443 -> 10.1.26.47:59
05/21-11:09:12.693869  [**] [1:1000003:0] Testing Msg UDP [**] [Priority: 0] {UDP} 10.1.27.83:38034 -> 255.255.255.25
0001
05/21-11:09:12.693869  [**] [1:1000003:0] Testing Msg UDP [**] [Priority: 0] {UDP} 10.1.27.17:5353 -> 224.0.0.251:535
05/21-11:09:12.740659  [**] [1:1000002:0] Testing Msg TCP [**] [Priority: 0] {TCP} 150.171.28.11:443 -> 10.1.26.47:59
05/21-11:09:12.758071  [**] [1:1000002:0] Testing Msg TCP [**] [Priority: 0] {TCP} 23.11.215.147:443 -> 10.1.26.47:59
```

reference:

- [https://youtu.be/e\\_KV3xivSnl?si=-lsA0rlhgR-inOWA](https://youtu.be/e_KV3xivSnl?si=-lsA0rlhgR-inOWA)
- <https://youtu.be/4Z7ii2al-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:

```
listen 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
<code>id</code>	INT	AUTO_INCREMENT, PRIMARY KEY
<code>timestamp</code>	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP
<code>signature</code>	VARCHAR(255)	—
<code>priority</code>	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
-----
```

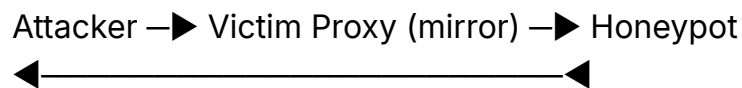
### Key Learnings

- 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



# Deception Proxy for Cybersecurity Research (Documentation)

## Overview

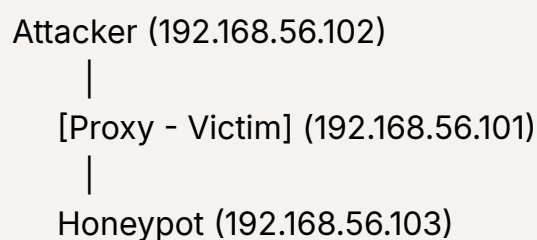
This project implements a **Deception Proxy** on a Windows-based "Victim" system to redirect all TCP traffic from an attacker to a honeypot. The victim acts as a transparent middleman (man-in-the-middle) and forwards all incoming packets to the honeypot, giving the attacker false feedback. This proxy 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:



## 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 {HONEYPOT_IP}")
logging.info("Proxy started: Listening on port %s, redirecting to honeypot at %s", LISTEN_PORT, HONEYPOT_IP)

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, 'client_to_honeypot')).start()
        threading.Thread(target=forward, args=(honeypot_socket, client_socket, 'honeypot_to_client')).start()

    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
# =====

```

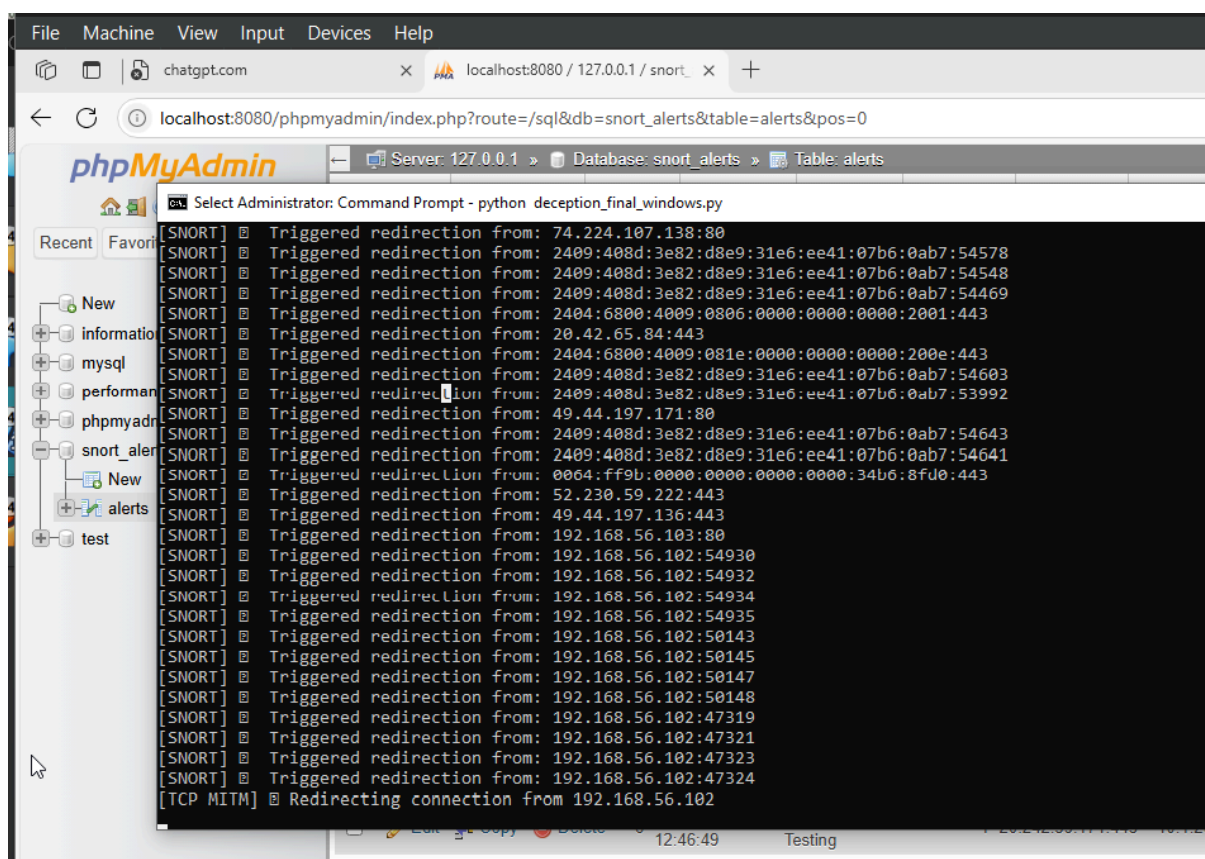
```

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()

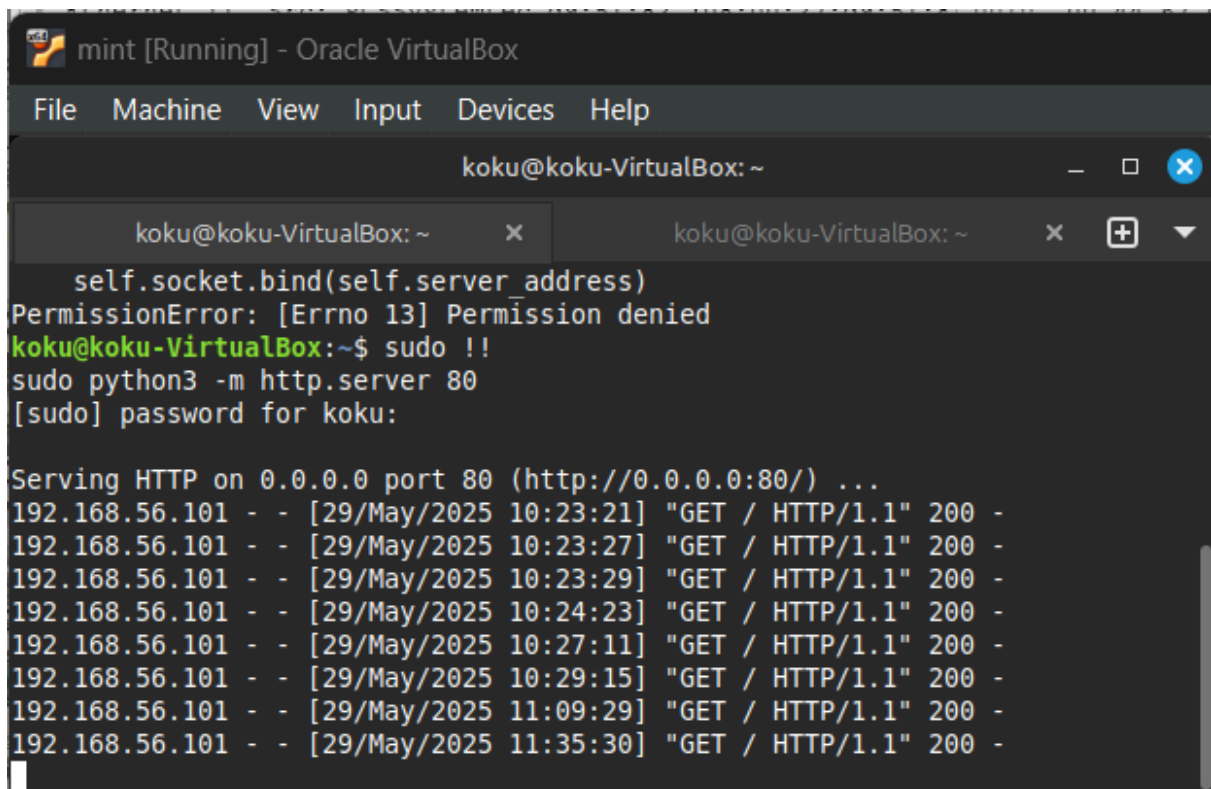
```

python deception\_proxy.py



## Step 3: Configure Honeytrap

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



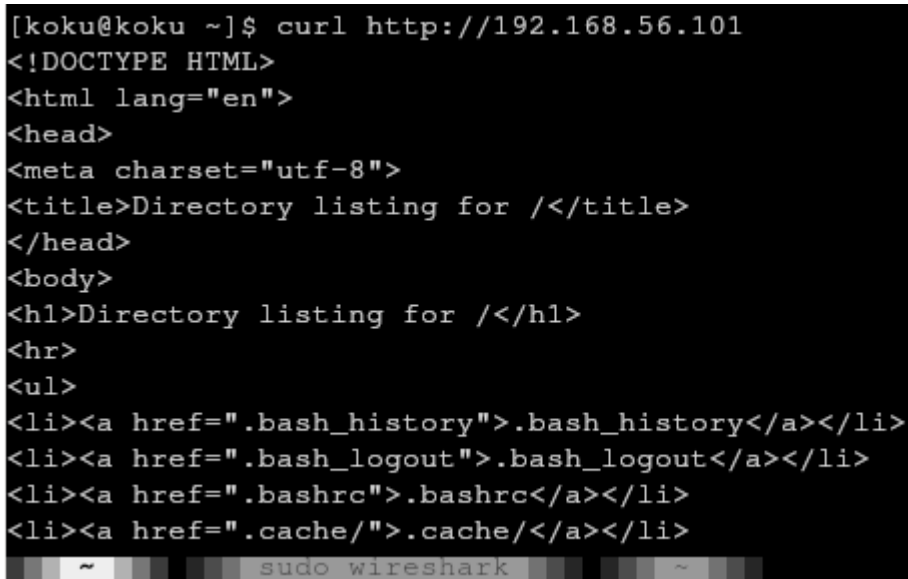
```
mint [Running] - Oracle VirtualBox
File Machine View Input Devices Help
koku@koku-VirtualBox: ~
koku@koku-VirtualBox: ~ x koku@koku-VirtualBox: ~ x + v
self.socket.bind(self.server_address)
PermissionError: [Errno 13] Permission denied
koku@koku-VirtualBox:~$ sudo !!
sudo python3 -m http.server 80
[sudo] password for koku:
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.56.101 - - [29/May/2025 10:23:21] "GET / HTTP/1.1" 200 -
192.168.56.101 - - [29/May/2025 10:23:27] "GET / HTTP/1.1" 200 -
192.168.56.101 - - [29/May/2025 10:23:29] "GET / HTTP/1.1" 200 -
192.168.56.101 - - [29/May/2025 10:24:23] "GET / HTTP/1.1" 200 -
192.168.56.101 - - [29/May/2025 10:27:11] "GET / HTTP/1.1" 200 -
192.168.56.101 - - [29/May/2025 10:29:15] "GET / HTTP/1.1" 200 -
192.168.56.101 - - [29/May/2025 11:09:29] "GET / HTTP/1.1" 200 -
192.168.56.101 - - [29/May/2025 11:35:30] "GET / HTTP/1.1" 200 -
```

```
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:



```
[koku@koku ~]$ curl http://192.168.56.101
<!DOCTYPE HTML>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href=".bash_history">.bash_history</a></li>
<li><a href=".bash_logout">.bash_logout</a></li>
<li><a href=".bashrc">.bashrc</a></li>
<li><a href=".cache/">.cache/</a></li>
</ul>
~ sudo wireshark ~
```



```
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

No.	Time	Source	Destination	Protocol	Length	Info
237	604.773757010	192.168.56.101	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
238	605.412404137	PCSSystemtec_87:3e:...	PCSSystemtec_11:1c:...	ARP	42	Who has 192.168.56.100? Tell 192.168.56.100
239	605.413757933	PCSSystemtec_11:1c:...	PCSSystemtec_87:3e:...	ARP	60	192.168.56.100 is at 08:00:27:11:1c:cd
240	607.777347985	192.168.56.101	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
241	610.796926000	192.168.56.101	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
242	613.814156622	192.168.56.101	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
243	616.816960484	192.168.56.101	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
244	617.050724857	192.168.56.101	192.168.56.255	BROWSER	258	Domain/Workgroup Announcement WORKGROUP
245	678.164777481	192.168.56.101	192.168.56.255	BROWSER	243	Local Master Announcement DESKTOP-4STI8
246	678.718538251	192.168.56.101	224.0.0.251	MDNS	82	Standard query 0x0000 PTR _googlecast.
247	678.718539223	fe80::8b41:d3cb:e5b...	#02::fb	MDNS	102	Standard query 0x0000 PTR _googlecast.
248	679.731531775	192.168.56.101	224.0.0.251	MDNS	82	Standard query 0x0000 PTR _googlecast.
249	679.731532615	fe80::8b41:d3cb:e5b...	#02::fb	MDNS	102	Standard query 0x0000 PTR _googlecast.
250	681.735576307	192.168.56.101	224.0.0.251	MDNS	82	Standard query 0x0000 PTR _googlecast.
251	681.739303795	fe80::8b41:d3cb:e5b...	#02::fb	MDNS	102	Standard query 0x0000 PTR _googlecast.
252	703.307360755	192.168.56.102	192.168.56.101	TCP	74	42442 → 80 [SYN] Seq=0 Win=64240 Len=0
253	703.325131675	192.168.56.101	192.168.56.102	TCP	66	80 → 42442 [SYN, ACK] Seq=0 Ack=1 Win=6
254	703.325335311	192.168.56.102	192.168.56.101	TCP	54	42442 → 80 [ACK] Seq=1 Ack=1 Win=64256

Attacker

No.	Time	Source	Destination	Protocol	Length	Info
234	698.969603353	PCSSystemtec_c2:42:...	PCSSystemtec_d9:51:...	ARP	42	192.168.56.103 is at 08:00:27:11:1c:cd
235	698.972250103	192.168.56.101	192.168.56.103	TCP	66	49680 → 80 [SYN] Seq=0 Win=642
236	698.974805479	192.168.56.103	192.168.56.101	TCP	66	80 → 49680 [SYN, ACK] Seq=0 Win=6
237	698.977616792	192.168.56.101	192.168.56.103	TCP	60	49680 → 80 [ACK] Seq=1 Ack=1 W
238	698.991325275	192.168.56.101	192.168.56.103	HTTP	132	GET / HTTP/1.1
239	698.991698183	192.168.56.103	192.168.56.101	TCP	54	80 → 49680 [ACK] Seq=1 Ack=79
240	699.094878644	192.168.56.103	192.168.56.101	TCP	210	80 → 49680 [PSH, ACK] Seq=1 Ac
241	699.097789316	192.168.56.103	192.168.56.101	HTTP	1382	HTTP/1.0 200 OK (text/html)
242	699.106297270	192.168.56.101	192.168.56.103	TCP	60	49680 → 80 [ACK] Seq=79 Ack=14
243	699.110763092	192.168.56.101	192.168.56.103	TCP	60	49680 → 80 [FIN, ACK] Seq=79 A
244	699.111031432	192.168.56.103	192.168.56.101	TCP	54	80 → 49680 [ACK] Seq=1486 Ack=
245	704.000660700	PCSSystemtec_c2:42:...	PCSSystemtec_d9:51:...	ARP	42	Who has 192.168.56.101? Tell 1

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)
-