**🔐 What is Network Exploitation?**

Network exploitation refers to the act of leveraging vulnerabilities in a network or system to gain unauthorized access or control. This often involves:

1. **Network Scanning** – Discovering devices, IPs, ports, and services.
2. **Enumeration** – Collecting more detailed data (users, shares, banners).
3. **Exploitation** – Using known vulnerabilities to break in (e.g., buffer overflow, unpatched services).

**Disclaimer:** All code below is for **educational purposes only**. Never test against systems without **explicit permission**.

**🔁 What is Recursion?**

Recursion is when a function **calls itself** to repeat a task — useful when tasks follow a nested or repeatable structure.

**🧪 Example: Recursive Port Scanner (Simple Exploitation + Recursion)**

This example will:

1. Take an IP or subnet.
2. Scan ports recursively (1–N range).
3. Try banner grabbing for each open port.

**✅ Step-by-Step Code Explanation**

import socket

# Step 1: Recursive function to scan a range of ports

def recursive\_port\_scan(ip, start\_port, end\_port):

# Base case: if start is greater than end, stop

if start\_port > end\_port:

return

try:

# Step 2: Create a socket object

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

sock.settimeout(1)

# Step 3: Attempt connection

result = sock.connect\_ex((ip, start\_port))

if result == 0:

print(f"[+] Port {start\_port} is OPEN")

# Step 4: Try banner grabbing (exploitation step)

try:

sock.send(b'Hello\r\n')

banner = sock.recv(1024)

print(f" Banner: {banner.decode().strip()}")

except Exception as e:

print(f" No banner: {e}")

sock.close()

except Exception as e:

print(f"Error on port {start\_port}: {e}")

# Step 5: Recursive call to scan the next port

recursive\_port\_scan(ip, start\_port + 1, end\_port)

# 🎯 Entry Point

if \_\_name\_\_ == "\_\_main\_\_":

target\_ip = "192.168.1.1" # Replace with target IP

port\_range\_start = 75

port\_range\_end = 85

print(f"Scanning {target\_ip} from port {port\_range\_start} to {port\_range\_end}")

recursive\_port\_scan(target\_ip, port\_range\_start, port\_range\_end)

**🔍 How This Ties into Exploitation**

* **Scanning Open Ports**: First step in network recon and exploitation.
* **Banner Grabbing**: Useful to identify the service/version (e.g., "Apache 2.4.7") which may have known vulnerabilities (e.g., CVE-XXXX-XXXX).
* **Recursion**: This replaces a for loop to show how tasks can be handled via self-calling functions.

**⚡ Advanced Idea: Recursive Directory Brute-forcing on a Web Server**

This could simulate scanning web directories for a vulnerable endpoint.

Let me know if you'd like this second example!

**✅ Part 1: Multithreaded Port Scanner (Fast Network Exploitation)**

This builds on the earlier recursive port scanner but uses threading for parallel scanning.

**🚀 Code: Multithreaded Port Scanner**

import socket

import threading

# Function to scan a single port

def scan\_port(ip, port):

try:

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

sock.settimeout(1)

result = sock.connect\_ex((ip, port))

if result == 0:

print(f"[+] Port {port} is OPEN")

try:

sock.send(b'Hello\r\n')

banner = sock.recv(1024)

print(f" Banner: {banner.decode(errors='ignore').strip()}")

except:

print(" No banner")

sock.close()

except Exception as e:

print(f"Error on port {port}: {e}")

# Main function to create threads for each port

def threaded\_scan(ip, start\_port, end\_port):

threads = []

for port in range(start\_port, end\_port + 1):

t = threading.Thread(target=scan\_port, args=(ip, port))

t.start()

threads.append(t)

for thread in threads:

thread.join()

if \_\_name\_\_ == "\_\_main\_\_":

target = "192.168.1.1" # Replace with your target IP

print(f"Scanning {target} using multithreading...")

threaded\_scan(target, 75, 85)

🟢 **Why it's better**: Threads allow multiple ports to be scanned in parallel, massively increasing speed.

**✅ Part 2: Recursive Web Directory Scanner (Web Exploitation)**

This simulates a basic directory brute-force scanner to uncover hidden or sensitive paths.

**🚨 Disclaimer: Do not run against live sites without permission.**

**🌐 Code: Recursive Web Directory Brute-Forcer**

import requests

# Recursively try paths from a list

def recursive\_dir\_scan(base\_url, paths, depth=0, max\_depth=2):

if depth > max\_depth:

return

for path in paths:

url = f"{base\_url.rstrip('/')}/{path}"

try:

response = requests.get(url, timeout=3)

if response.status\_code == 200:

print(f"[+] Found: {url} ({response.status\_code})")

# Simulate recursion into deeper directory

recursive\_dir\_scan(url, paths, depth + 1, max\_depth)

elif response.status\_code in [301, 302]:

print(f"[~] Redirected: {url}")

except requests.RequestException as e:

print(f"[!] Error: {url} - {e}")

if \_\_name\_\_ == "\_\_main\_\_":

base = "http://testphp.vulnweb.com" # Replace with your target URL

wordlist = ["admin", "login", "images", "uploads", "includes", "backup"]

recursive\_dir\_scan(base, wordlist, max\_depth=2)

🟢 **How it works**:

* Takes a base\_url.
* Tries to access common directories.
* If a directory exists (200 OK), it tries subdirectories recursively.