Week 07 - Penetration Testing

Whois Domain Lookup (Passive Reconnaissance)

Below python script resolves the domain's IP and retrieves the public OSIN data that includes Organization's name, city and country

Key Findings

This is called passive reconnaissance that means collectig publicly available infromation without interacting directly with the target system. It helps to map out ownership & infrastructure before any scanning occurs.

```
import socket
import requests

def get_domain_info(domain):
    try:
        # Resolve the domain to an IP address (active DNS lookup)
        ip = socket.gethostbyname(domain)
        print(f"IP Address: {ip}")

        # Query public IP information (organization, geolocation, ASN)
        # This is passive OSINT and does not interact with the target server
        response = requests.get(f"https://ipapi.co/{ip}/json/")

        if response.status_code == 200:
            data = response.json()
            print(f"Organization: {data.get('org', 'Unknown')}")
            print(f"City: {data.get('city', 'Unknown')}")
            print(f"Country: {data.get('country_name', 'Unknown')}")
        else:
            print("Could not fetch WHOIS-style data.")

    except Exception as e:
        # Handles DNS failures, bad responses, or network errors
        print(f"Error: {e}")

# Passive reconnaissance on a public domain (permission required for deeper probing)
get_domain_info("python.com")
```

```
IP Address: 185.158.133.1
Organization: CLOUDFLARENET
City: Frankfurt am Main
Country: Germany
```

Reflection

As we can see that how much information is being exposed through DNS and third-party data services. And even without active probing, attackers can build a profile of the target's environment. Thus it reinforces the need for organizations to minimise leakage of metadata.

Black Box Testing (HTTP Header Analysis) With Python

The below script will send an **HTTP HEAD** request and it will extract the server name banners such as Server and Content-Type.

Key Findings

It is basic black box testing as it gathers system information without internal access. And servers often restrict and obfuscates the headers as a security control to limit fingerprinting.

```
import requests

def black_box_recon(url):
    try:
        # HEAD request: retrieves headers without downloading the page
        response = requests.head(url)
```

```
            print("Black Box Findings:")

            # Server banner (may reveal software/version if not hardened)
            print(f"Server: {response.headers.get('Server', 'Unknown')}")

            # Content type returned by the server
            print(f"Content-Type: {response.headers.get('Content-Type', 'Unknown')}")

        except Exception as e:
            # Network or connection error
            print(f"Error: {e}")

    # Target for header-based reconnaissance
    url = "http://python.com"

    # Reference info for manual comparison (not used by script)
    known_info = {"server": "Apache 2.4", "vulns": "Check CVE-2021-1234"}

    black_box_recon(url)
```

```
Black Box Findings:
Server: cloudflare
Content-Type: Unknown
```

Reflection

Enumeration of header is low-effort but it can reveal software version or misconfigurations. Even the "Unknown" responses are meaningful in this context as it shows us system's defensive hardening.

Basic TCP Port Scanning (Raw Sockets) Using Python

This below script tests the selected ports on localhost and identifies which ones were open.

Key Findings

It basically simulates a basic tcp connect scan. As it reveals how open ports expose assecible services and why port inspection is an important step in penetration testing.

```
import socket

def scan_ports(host, ports):
    open_ports = []
    for port in ports:
        # Create a TCP socket for each port tested
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(1)   # Avoid hanging on unresponsive ports

        # Attempt TCP connection; 0 = port open, otherwise closed/filtered
        result = sock.connect_ex((host, port))

        if result == 0:
            open_ports.append(port)

        sock.close()  # Close socket to avoid resource leaks

    return open_ports

# Localhost scanning (safe and authorised)
host = "127.0.0.1"
ports = [80, 443, 22, 8080]

open_ports = scan_ports(host, ports)
print(f"Open ports on {host}: {open_ports}")
```

```
Open ports on 127.0.0.1: [8080]
```

Reflection

This scanner works but it is slow and limited. But shows how attackers can systematically test for exposed services. Therefore, organizations must close the unused ports, and should monitor unusual scanning behaviour.

Port Scanning & Service Enumeration With Python-Nmap

Here, python-nmap library is used to perform a more structured scan with service & version detection (Nmap -sV equivalent) on an authorised host.

Key Findings

It provides much richer results than raw python scans by detailing open ports, running services and version details, which is essential for identifying potential security vulnerabilities.

```
# Install Nmap and the Python wrapper (for environments like Colab)
!apt-get install nmap
!pip install python-nmap

import nmap

def nmap_scan(host, port_range='1-1024'):
    nm = nmap.PortScanner()
    try:
        # Perform an Nmap scan with service/version detection (-sV)
        nm.scan(host, port_range, arguments='-sV')

        for host in nm.all_hosts():
            print(f"Host: {host} ({nm[host].hostname()})")
            print(f"State: {nm[host].state()}")  # up / down

            # Iterate through detected protocols (usually TCP/UDP)
            for proto in nm[host].all_protocols():
                print(f"Protocol: {proto}")

                # All discovered ports under this protocol
                lport = nm[host][proto].keys()

                for port in sorted(lport):
                    service = nm[host][proto][port]

                    # Report port state and identified service/version
                    print(
                        f"Port: {port}\t"
                        f"State: {service['state']}\t"
                        f"Service: {service.get('name', 'unknown')} "
                        f"{service.get('version', '')}"
                    )

    except Exception as e:
        # Handles environment issues, invalid ranges, or scan failures
        print(f"Error: {e}")

# Safe, authorised localhost scan
nmap_scan('127.0.0.1', '1-10')
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
nmap is already the newest version (7.91+dfsg1+really7.80+dfsg1-2ubuntu0.1).
0 upgraded, 0 newly installed, 0 to remove and 41 not upgraded.
Requirement already satisfied: python-nmap in /usr/local/lib/python3.12/dist-packages (0.7.1)
Host: 127.0.0.1 (localhost)
State: up
Protocol: tcp
Port: 1 State: closed   Service: tcpmux
Port: 2 State: closed   Service: compressnet
Port: 3 State: closed   Service: compressnet
Port: 4 State: closed   Service: unknown
Port: 5 State: closed   Service: rje
Port: 6 State: closed   Service: unknown
Port: 7 State: closed   Service: echo
Port: 8 State: closed   Service: unknown
Port: 9 State: closed   Service: discard
Port: 10        State: closed   Service: unknown
```

Reflection

It is clear that automated tools drastically improve accuracy and reduce mental efforts. However, unrestricted scanning without explicit permission is illegal, and output must always be interpreted carefully as part of a broader assessment.

Conclusion

- Open ports show which services are exposed, so they directly define the organisation's attack surface.
- Basic scans get detected easily because they use full TCP connections and create traffic that no real user would produce like many fast connection attempts across different ports, all from the same source, and with no real interaction afterward.
- Nmap's advanced scans rely on matching the target's responses to its big database of known service and OS fingerprints, but firewalls or filtering can change those responses and make the fingerprinting less accurate.
- Attackers put all these small findings together such as open ports, banners, versions, and any leaked details, in-order to build a clear picture of where the system is vulnerable.