Vulnerability Assessment and Penetration Testing

1. Vulnerability Scanning Techniques

Vulnerability scanning is the process of **identifying weaknesses in systems, networks, and applications** before attackers do. It involves automated tools (e.g., OpenVAS, Nessus, Nikto, Nmap scripts) that enumerate possible misconfigurations, outdated software, insecure services, and known CVEs.

Vulnerability scanning is a foundational component of cybersecurity assessment and focuses on identifying weaknesses in systems, networks, and web applications before attackers can exploit them. It relies on automated tools and scanning methodologies to evaluate the security posture of an environment. The goal is not only to detect vulnerabilities but also to understand their severity, potential impact, and likelihood of exploitation.

Network-based scanning involves using tools such as Nmap to enumerate live hosts, open ports, running services, and potential misconfigurations. Techniques such as SYN scanning, service detection, and OS fingerprinting help map out an attack surface. For example, a simple command like nmap -sV <IP> allows identification of versions of services running on a target machine, which is crucial for checking known CVEs. Application-level scanning focuses on web servers and applications, using tools like Nikto or Gobuster to identify outdated software, insecure configurations, hidden directories, and unpatched vulnerabilities. While network scans focus on ports and services, application scans reveal logic flaws, insecure endpoints, and exposure of sensitive files.

1.1 Core Concepts

a. Scan Types

- **Network Scanning**

  Used to identify open ports, services, and host information.

Scan Type	Description	Example Tool / Command
TCP SYN Scan	Fast and stealthy port scan using half-open connections	nmap -sS <IP>
Service Detection	Identifies service/version running on open ports	nmap -sV <IP>
OS Detection	Fingerprints OS using TCP/IP stack	nmap -O <IP>
Script Scanning	Runs NSE scripts to detect vulnerabilities	nmap --script vuln <IP>

- **Application Scanning**

  Used to identify web vulnerabilities, misconfigurations, outdated frameworks, etc.

Tool	Purpose
Nikto	Detects outdated software, XSS, insecure headers
Gobuster	Directory brute-forcing and hidden endpoint discovery
wpscan	WordPress vulnerability assessment

b. Vulnerability Scoring (CVSS v4.0)

Vulnerability severity must be understood through a standardized scoring system such as **CVSS v4.0**, which assigns numerical values to quantify risk. Scores are categorized from Low to Critical, where vulnerabilities like Remote Code Execution often fall into High or Critical severity. For instance, the Apache Struts RCE vulnerability (CVE-2017-5638) is rated Critical due to its ability to allow full system compromise over the network without authentication. Understanding severity helps organizations prioritize remediation.

CVSS (Common Vulnerability Scoring System) gives a standardized **0–10** severity score.

Score Range	Severity	Example
9.0 – 10.0	Critical	RCE vulnerabilities (e.g., Log4j – CVSS 10.0)
7.0 – 8.9	High	Remote Code Execution (RCE), privilege escalation
4.0 – 6.9	Medium	Information disclosure
0.1 – 3.9	Low	Misconfigurations, minor leaks

Example:

- *CVE-2017-5638 (Apache Struts RCE)* → **CVSS 10 (Critical)**
- *Open Docker API Exposure* → typically CVSS **8.8 (High)**

c. False Positives

Since scanners often flag issues automatically, they may produce **false positives**, meaning findings that appear dangerous but are not actually exploitable. This can happen when outdated banners are detected even though patches are applied, or when a service responds in a way that resembles a vulnerability but is not vulnerable in practice. Therefore, validating scanner results through manual testing is essential to maintain accuracy.

A vulnerability scanner may detect issues that are **not actually exploitable**.

Common causes:

- Banner grabbing shows outdated software, but patch is applied.
- Port appears open but filtered in the firewall.
- Web scanner misinterprets error messages as vulnerabilities.

Validation Required:

- Cross-verify using manual testing.
- Compare with system configuration.

Example:

If Nmap detects port **21 (FTP)** open but authentication is blocked → **false positive**.

d. Key Objectives of Vulnerability Scanning

- Identify exploitable weaknesses before attackers do.
- Generate accurate severity-based findings.
- Reduce false positives with manual validation.
- Provide actionable remediation guidelines.
- Measure system security maturity.

1.2 How to Learn

Recommended Resources:

Source	Why It's Useful
OWASP Testing Guide	Covers deep web scanning concepts
NIST SP 800-115	U.S. federal standard for security testing
WannaCry Case Study	Demonstrates real-world CVSS mapping (EternalBlue CVSS 8.1 → global RCE attacks)
OpenVAS Documentation	Learn enterprise-grade scanning workflows

2. Penetration Testing Techniques

Penetration testing is the simulated process of attacking a system to identify exploitable weaknesses before adversaries can misuse them. It goes beyond scanning by attempting to exploit discovered flaws and assess the full impact. The process follows a structured methodology that ensures ethical, safe, and actionable results.

Pentesting typically begins with reconnaissance, where testers gather publicly available information about a target using OSINT techniques or tools like Shodan. This phase helps identify external exposure and potential entry points. The next stage, scanning and

enumeration, involves using tools such as Nmap, OpenVAS, or Nessus to map out networks, services, and known vulnerabilities. Once opportunities are identified, testers move into the exploitation phase, where tools like Metasploit, SQLmap, or custom scripts are used to gain unauthorized access or extract sensitive data.

After successfully exploiting a system, the process shifts into post-exploitation, which focuses on exploring the system further, escalating privileges, maintaining access, or pivoting into other systems. Techniques may include running Meterpreter commands, dumping credentials, or exploiting kernel vulnerabilities for root-level access. The final stage of pentesting is reporting, which documents all findings, their impact, and recommended remediations in a clear and actionable format.

2.1 Core Concepts

a. Penetration Testing Phases

Phase	Description	Example Tools
1. Reconnaissance	Information gathering	OSINT, Shodan, WhatWeb
2. Scanning & Enumeration	Discover ports, services, vulnerabilities	Nmap, Nessus, OpenVAS
3. Exploitation	Exploit weaknesses to gain access	Metasploit, SQLmap
4. Post-Exploitation	Privilege escalation, persistence	Meterpreter, LinPEAS
5. Reporting	Document findings, risks, mitigation	Dradis, manual reports

Example:
Using **Shodan** to discover exposed services → verifying with **Nmap** → exploiting with **Metasploit**.

b. Pentesting Methodologies

1. PTES (Penetration Testing Execution Standard)

Used for **structured, professional pentests**.

PTES Steps:

1. Pre-engagement Interactions
2. Intelligence Gathering
3. Threat Modeling
4. Vulnerability Analysis
5. Exploitation
6. Post-Exploitation
7. Reporting

Example:
Using PTES for scoping a **web application assessment**.

2. OWASP WSTG (Web Security Testing Guide)

Industry standard for **web pentesting**.

Coverage includes:

- Authentication testing
- Session management
- Input validation (e.g., SQLi, XSS)
- Error handling
- API security

c. Ethical Considerations

- Obtain **signed authorization** before starting tests.
- Work within **defined scope** (IPs, systems, time window).
- Avoid damaging production systems.

- Maintain confidentiality of client data.
- Follow responsible disclosure practices.

d. Key Objectives of Pentesting

- Identify exploitable vulnerabilities.
- Evaluate security posture from an attacker perspective.
- Validate effectiveness of defensive controls.
- Provide actionable mitigation strategies.

2.2 How to Learn

Resource	What You Learn
PTES Official Documentation	Professional methodology for pentests
OWASP WSTG	Web application penetration testing
SANS Case Studies	Real-world pentests, techniques, and lessons
HackTheBox / TryHackMe	Hands-on exploitation practice

Ethics form a critical part of penetration testing. Testers must always operate under written authorization, work strictly within the agreed scope, avoid causing harm to production systems, and maintain confidentiality of any sensitive information discovered during testing. Ethical considerations ensure trust, legality, and professionalism throughout the engagement.

Learning pentesting effectively involves studying PTES and OWASP WSTG, analyzing case studies from SANS to understand real attack scenarios, and practicing hands-on exploitation through platforms like TryHackMe or HackTheBox. These combined resources help build a structured understanding of how real-world attacks unfold and how defenders can prevent them.

3. Exploit Development Basics

Exploit development is the process of understanding vulnerabilities at a deeper technical level and crafting custom payloads or scripts to trigger them. It requires knowledge of how software interacts with memory, how input is processed, and how security mechanisms operate. Unlike basic exploitation with tools, exploit development focuses on creating or modifying exploit code to demonstrate the risk posed by a vulnerability.

There are various types of exploits, including buffer overflows, SQL injection, and Cross-Site Scripting (XSS). Buffer overflows occur when applications fail to enforce memory boundaries, allowing attackers to overwrite critical sections of memory such as the instruction pointer. SQL injection exploits arise when user input is not sanitized, allowing attackers to manipulate database queries. XSS occurs when web applications render unescaped user input, leading to execution of malicious JavaScript in a victim's browser. These vulnerabilities demonstrate the importance of secure coding practices and input validation.

Writing an exploit involves multiple steps: identifying the vulnerable function, analyzing the program's behavior using debuggers such as GDB with extensions like pwndbg, determining the correct offset to overwrite memory safely, and crafting shellcode or payloads using tools like msfvenom. Many proof-of-concept exploits in Python or C can be found on Exploit-DB, providing practical reference points for learning how real exploits are structured.

3.1 Core Concepts

a. Types of Exploits

Exploit Type	Description	Example
Buffer Overflow	Overwriting memory to hijack execution flow	Classic stack overflow (e.g., vulnserver)
SQL Injection	Injecting SQL queries into input fields	' OR 1=1 --

Cross-Site Scripting (XSS)	Injecting malicious JavaScript	<script>alert(1)</script>
RCE (Remote Code Execution)	Executing remote commands	Log4Shell CVE-2021-44228

b. Exploit Writing Basics

Common languages used:

- **Python** → proof-of-concept exploits
- **C** → low-level binary exploits
- **Bash/PowerShell** → simple payload delivery

You learn to:

1. Identify vulnerable functions (e.g., strcpy, gets).
2. Calculate buffer size using fuzzing.
3. Overwrite EIP (Instruction Pointer).
4. Inject shellcode (payload).

Example tools:

- gdb
- pwndbg
- pattern_create & pattern_offset
- msfvenom

c. Understanding Mitigations

Systems use various protections against exploits:

Mitigation	Description
ASLR (Address Space Layout Randomization)	Randomizes memory locations

DEP / NX-bit	Prevents execution of injected code
Stack Canaries	Detects stack corruption
WAF (Web Application Firewall)	Filters malicious inputs

Exploit developers must learn **bypass techniques** for each.

d. Key Objectives of Exploit Development

- Understand how vulnerabilities work.
- Create safe exploit proof-of-concepts (PoCs).
- Test exploit behavior in a controlled environment.
- Understand patching and defense strategies.

3.2 How to Learn

Resource	Why It Helps
Exploit-DB	Thousands of public PoCs
TCM Security Exploit Development Series	Beginner-friendly walkthroughs
TryHackMe: Buffer Overflow Room	Hands-on exercises
Georgia Weidman's Penetration Testing Book	Deep introduction to exploit writing

The primary objectives of exploit development are to deepen understanding of how vulnerabilities work, learn how to construct safe and controlled proof-of-concept attacks, and develop the ability to analyze and remediate complex security flaws. It also trains professionals to think like attackers, which is invaluable for designing secure systems.

A strong learning path includes studying exploit code from Exploit-DB, following exploit development tutorials from TCM Security, and practicing buffer overflow challenges through TryHackMe's dedicated learning rooms.

4. Key Findings

The theoretical study of vulnerability scanning, penetration testing, and exploit development highlights several important aspects that form the foundation of a structured and effective security assessment. First, vulnerability scanning techniques particularly those involving network and web application analysis demonstrate the critical need for understanding different scan types and their appropriate use. Proper configuration of scanning tools directly influences the accuracy and reliability of results. The role of CVSS scoring in prioritizing vulnerabilities is reinforced through real-world examples, showing how high-severity issues like Remote Code Execution (RCE) significantly elevate risk. The study also highlights that false positives are common and must be validated manually, emphasizing the importance of analyst judgment rather than relying solely on automated scanners.

Pentesting techniques further reinforce the structured nature of offensive security. Each phase reconnaissance, scanning, exploitation, and post-exploitation plays a distinct role in identifying and validating weaknesses. The use of standards such as PTES and OWASP WSTG ensures that testing is both methodical and ethical, reducing the likelihood of oversight while helping maintain compliance with legal and professional requirements. Ethical considerations emerged as a consistent theme, emphasizing the need for authorization, adherence to scope, and responsible reporting. Through this structured approach, testers gain a clearer understanding of how attackers think and operate, enabling them to uncover systemic weaknesses that may not be visible through automated scanning alone.

Additionally, the exploration of exploit development fundamentals highlights the technical depth required to understand vulnerabilities at the code and memory level. Concepts such as buffer overflows, SQL injection, and XSS illustrate how insecure programming practices can lead to severe security flaws. The study demonstrates that exploit development is not only about writing attack code but also about understanding mitigation mechanisms like ASLR, stack canaries, and WAFs. This helps security professionals recognize both offensive techniques and defensive strategies, building a more comprehensive understanding of software security. Practical PoCs from platforms such as Exploit-DB and hands-on exercises strengthen the connection between theory and real-world exploitation scenarios.

In summary, the key findings demonstrate that:

- Vulnerability scanning requires careful configuration and interpretation to avoid inaccurate results.
- CVSS scoring frameworks are essential for prioritizing remediation based on real-world risk.
- Penetration testing must follow structured methodologies to ensure ethical, repeatable, and comprehensive assessment.
- Exploit development deepens understanding of how vulnerabilities function and how modern defenses impact exploitation feasibility.
- Manual validation, analyst judgment, and structured documentation remain indispensable components of the security assessment lifecycle.

5. Conclusion

The theoretical knowledge acquired through the study of vulnerability scanning, penetration testing techniques, and exploit development establishes a strong foundation for practical VAPT operations. Understanding how scanners work, how vulnerabilities are scored, and how false positives occur enables more accurate and meaningful assessments. Similarly, mastering structured pentesting methodologies ensures that assessments are conducted professionally, ethically, and with measurable outcomes. These frameworks help transform raw findings into actionable insights that organizations can use to strengthen their security posture.

Furthermore, the introduction to exploit development concepts provides valuable insight into the deeper mechanisms of vulnerabilities and the logic behind exploitation. This fosters a more analytical mindset, allowing security professionals to understand not only how attacks work but also how to effectively mitigate them. As organizations face an increasing number of sophisticated threats, having this grounded knowledge is essential for building resilient defenses.

Overall, the combined study reinforces that cybersecurity is both a technical and analytical discipline requiring continuous learning, hands-on practice, and adherence to industry best

practices. By integrating vulnerability scanning, pentesting methodologies, and exploit development fundamentals, a practitioner becomes well-equipped to identify, analyze, and remediate security gaps in real-world environments. This theoretical understanding will directly support the practical execution of VAPT tasks, improving the depth, accuracy, and impact of future assessments.

Document Information:

Report Generated: December 12, 2025

Author: Soumendu Manna - CyArt VAPT Intern