# Penetration Testing Methodology

## A Complete 10-Chapter Security Course

**Topics covered:**

Authorization, scoping & standard methodology frameworks (PTES/OWASP/NIST)

Reconnaissance, scanning & enumeration

Vulnerability analysis using this site's own XSS/CSRF/SQLi/Auth/DB Security toolkit

Exploitation, post-exploitation & professional reporting

**Exercises:** 30 hands-on scenarios with worked solutions

**Format:** A4 · Dark-theme code examples

**Single standalone course** · a pentest is a repeatable, disciplined process

Philip Osztromok · Generated with Claude

# Table of Contents

CHAPTER 1 OF 10

# What Penetration Testing Actually Is (and Requires)

## Penetration Testing Methodology

Chapter 1 · What Penetration Testing Actually Is (and Requires)

This site already has deep, dedicated courses on specific vulnerability classes — `xss1`, `csrf1`, `sqli1`, `bc1`, `dbsec1`, and the `owasp1` survey course tying several of them together. This course is different: it's not about any one vulnerability. It's about the **process** a professional actually follows to find and prove real security weaknesses in a target system — and before a single technique gets named, there's one precondition that has to be true first.

### Penetration Testing vs. Related Disciplines

"Security testing" covers several genuinely different activities, often confused with each other:

| DISCIPLINE | WHAT IT ACTUALLY DOES | TYPICAL SCOPE/DURATION |
|---|---|---|
| Vulnerability Scanning | Automated, tool-driven identification of known vulnerability signatures — largely non-invasive, doesn't attempt exploitation | Continuous or scheduled, broad |
| Penetration Testing | Manual, expert-driven testing that *attempts real exploitation* to prove actual, demonstrable impact — this course's own subject | Time-boxed, formally scoped engagement |
| Red Teaming | A broader, more adversarial simulation of a real determined attacker's full campaign — often combining technical, physical, and social engineering vectors, and often unannounced to the target's own defenders, specifically to test detection and response | Longer, goal-oriented rather than checklist-oriented |
| Bug Bounty Hunting | Crowdsourced, ongoing testing against a publicly published scope/policy — anyone meeting that policy's terms can participate and get paid for valid findings | Open-ended, no single formal engagement window |

This course focuses specifically on penetration testing — but the precondition covered next applies, in one form or another, to every discipline in that table.

### The Single Non-Negotiable Precondition — Written Authorization

Before any reconnaissance, any scanning, any attempt at exploitation — before *anything* this course covers from Chapter 4 onward — one thing has to already be true: **explicit, written authorization from someone with actual legal authority over the target system**. Not a verbal "sure, go ahead." Not an assumption that testing is welcome because a company has a bug bounty page somewhere. A real, specific, written agreement, in place before testing starts.

> **UNAUTHORIZED TESTING IS A REAL CRIME, EVEN WITH GOOD INTENTIONS**
>
> In the United States, the Computer Fraud and Abuse Act (CFAA) criminalizes accessing a computer system without authorization — and most other jurisdictions have comparable computer-crime statutes. Genuinely good intentions (wanting to help, planning to report a bug responsibly, believing a system "looked" insecure enough to justify a look) provide no legal protection at all without real, prior authorization. People with entirely good motives have been investigated and prosecuted for exactly this. This isn't a footnote to penetration testing methodology — it's the actual foundation everything else in this course sits on top of.

## What Authorization Actually Looks Like

In practice, authorization takes the form of a written **scope agreement** or **Rules of Engagement (RoE)** document — sometimes informally called a "get out of jail free" letter — signed by someone who genuinely has the authority to grant it for the specific systems being tested. It defines, in writing, exactly what's in scope, what testing windows are permitted, what techniques are and aren't allowed, and who to contact if something unexpected happens during testing. `pentest1-2` covers the full structure of this document in depth.

Bug bounty programs work slightly differently: authorization comes from the program's own published policy rather than an individually signed document — but the same underlying principle holds. Testing anything outside that policy's own stated scope is exactly as unauthorized as testing a company with no bug bounty program at all.

> **EVERYTHING IN THIS COURSE ASSUMES AUTHORIZATION ALREADY EXISTS**
>
> From here forward, every technique, framework, and phase this course covers is written for the situation where a real, written authorization is already in place. This chapter's own emphasis isn't excessive caution — it's the actual professional standard the rest of this course is built on top of.

## What This Course Covers

| CHAPTER | TOPIC |
|---|---|
| 2 | Engagement Types & Scoping |
| 3 | Standard Methodology Frameworks |
| 4 | Reconnaissance & OSINT |

| CHAPTER | TOPIC |
|---|---|
| 5 | Scanning & Enumeration |
| 6 | Vulnerability Analysis — The Toolkit This Site Already Built |
| 7 | Exploitation — Proving Impact Without Causing Harm |
| 8 | Post-Exploitation & Privilege Escalation |
| 9 | Reporting — The Actual Deliverable |
| 10 | Capstone: Scoping and Planning a Real Engagement |

## Hands-On Exercises

**EXERCISE 1**

Using this chapter's own comparison table, explain the specific difference between vulnerability scanning and penetration testing — what does a pentest do that a scan deliberately does not?

View solution

**EXERCISE 2**

A well-meaning developer notices what looks like an exposed admin panel on a company's public website and decides to "test" whether it's actually vulnerable, intending to report it responsibly if so. Using this chapter's own warn-box, explain what's legally wrong with this plan, regardless of the developer's intentions.

View solution

**EXERCISE 3**

Explain how authorization works differently for a formally scoped penetration test versus a bug bounty program, and explain why testing something outside a bug bounty program's own published scope is still unauthorized, even though the company clearly welcomes security research in general.

View solution

**CHAPTER 1 QUICK REFERENCE**

- **Vulnerability scanning** — automated, non-invasive · **Penetration testing** — manual, attempts real exploitation · **Red teaming** — broader adversary simulation, often tests detection/response · **Bug bounty** — crowdsourced, ongoing,

scoped by published policy

- Written authorization from someone with real legal authority over the target is the single non-negotiable precondition — before any technique in this course is used
- Unauthorized access is a real crime (the CFAA in the US, comparable laws elsewhere) — good intentions provide no legal protection
- Authorization takes the form of a **scope agreement / Rules of Engagement (RoE)** document, signed before testing begins — full structure covered in pentest1-2
- Bug bounty authorization comes from a program's own published policy — testing outside that scope is exactly as unauthorized as testing a company with no program at all
- Every later chapter in this course assumes authorization already exists — it's the foundation, not a footnote
- **Next chapter:** Engagement Types & Scoping — Black Box/White Box/Gray Box, and why scope must be precisely defined in writing

CHAPTER 2 OF 10

# Engagement Types & Scoping

## Penetration Testing Methodology

Chapter 2 · Engagement Types & Scoping

`pentest1-1` established that written authorization has to exist before anything else happens. This chapter covers what that authorization actually has to specify: what *kind* of test is being run, and — with real, sobering stakes — exactly what is and isn't in bounds.

### Black Box, White Box, and Gray Box Testing

How much internal knowledge the tester starts with is itself a deliberate, negotiated choice:

| | STARTING KNOWLEDGE | REALISM | THOROUGHNESS IN A FIXED TIMEFRAME |
|---|---|---|---|
| Black Box | None — no source, no architecture, no credentials | Highest — matches a real outside attacker | Lower — significant time spent just on recon |
| White Box | Full — source code, architecture docs, often credentials | Lowest — real attackers rarely start this informed | Highest — depth possible in the same time window |
| Gray Box | Partial — e.g. a standard user account, no source/admin access | Moderate — a genuinely common real starting position (a leaked credential, an insider) | Moderate-to-high — the most common practical compromise |

Gray Box is often the practical default for real engagements — it balances realism against making efficient use of a genuinely limited testing budget and timeframe.

### Internal vs. External Engagements

**External** engagements test from outside the organization's own network, simulating an internet-based attacker — this is the threat model most of this site's own web-application-focused courses (`xss1`, `csrf1`, `sqli1`) already assume. **Internal** engagements test from inside the network instead, simulating a compromised internal machine, a phishing victim's own foothold, or a malicious insider — a genuinely different threat model, and a direct real-world instance of `dbsec1-1`'s own insider-threat material: someone who already has legitimate network access, tested from that exact starting position.

## Scope Categories — What's Actually Being Tested

- **Network** — infrastructure-level testing: servers, firewalls, network segmentation.
- **Web Application** — the specific vulnerability classes this site already teaches in depth (`xss1`, `csrf1`, `sqli1`, `bc1`) — the direct technical grounding `pentest1-6` formally names as this course's own "toolkit."
- **Wireless** — Wi-Fi security testing, a genuinely distinct technical domain this course names but doesn't cover in depth.
- **Social Engineering** — phishing simulations, pretexting: testing *people*, not systems — a fundamentally different attack surface than anything else in this list.
- **Physical** — testing whether someone can physically walk into a facility or server room — easy to overlook, and a real, named category in almost every professional methodology.

## Why Scope Must Be Precise, in Writing

Vague scope isn't just an inefficiency — it's a real legal risk, directly extending `pentest1-1`'s own authorization material.

**SHARED INFRASTRUCTURE CAN QUIETLY BREAK AUTHORIZATION ENTIRELY**

A client can genuinely, sincerely authorize testing of "our website" without realizing that website sits on shared hosting, behind a third-party CDN, or inside a cloud environment where other systems — belonging to *other* organizations entirely — share the same infrastructure. Testing that goes even slightly beyond what the client actually owns and controls can mean the tester has unknowingly committed genuinely unauthorized access against a completely different organization, even though the client believed, in good faith, that they were authorizing it. This is exactly why scope has to specify precise IP ranges, exact domains, and explicit exclusions ("the payment processing system, though reachable from the same network, is out of scope") — not general descriptions like "our website" or "our network."

**WEB APPLICATION SCOPE IS WHERE THIS COURSE RECONNECTS WITH THE REST OF THE SITE**

`pentest1-6` comes back to the Web Application scope category specifically, naming the site's own existing XSS/CSRF/SQLi/Auth courses as the concrete vulnerability-class knowledge a real web-app-scoped engagement actually draws on.

## Hands-On Exercises

**EXERCISE 1**

A client wants the most realistic possible simulation of an external attacker and has a generous multi-week testing window. Using this chapter's own comparison table, which of Black Box, White Box, or Gray Box best fits, and why?

View solution

**EXERCISE 2**

Explain the real difference between an external and an internal engagement, and explain specifically how an internal engagement connects to dbsec1-1's own insider-threat material.

View solution

**EXERCISE 3**

Using this chapter's own warn-box, explain how a client sincerely authorizing "testing of our website" could still result in genuinely unauthorized access to a different organization's systems, and explain what specific scoping detail would have prevented it.

View solution

### CHAPTER 2 QUICK REFERENCE

- **Black Box** — no starting knowledge, most realistic, least thorough per unit time · **White Box** — full knowledge, most thorough, least realistic · **Gray Box** — partial knowledge, the common practical compromise
- **External** — tested from outside the network, matching xss1/csrf1/sqli1's own threat model · **Internal** — tested from inside, a real instance of dbsec1-1's own insider-threat material
- Five scope categories: **Network, Web Application, Wireless, Social Engineering, Physical** — testing infrastructure, code, radio, people, and buildings respectively
- Vague scope is a real legal risk, not just an inefficiency — shared/third-party infrastructure can turn sincerely-authorized testing into genuinely unauthorized access
- Scope must specify precise IP ranges, exact domains, and explicit exclusions — not general descriptions
- **Next chapter:** Standard Methodology Frameworks — PTES, the OWASP Testing Guide, and NIST SP 800-115

CHAPTER 3 OF 10

# Standard Methodology Frameworks

## Penetration Testing Methodology

Chapter 3 · Standard Methodology Frameworks

`pentest1-1` and `pentest1-2` covered authorization and scope — the preconditions a real engagement needs before it starts. This chapter covers what "methodology" actually means once testing does start: a documented, standardized structure, not an individual tester's own improvised approach.

### Why Standardized Frameworks Exist

Three concrete reasons a real engagement follows a published framework rather than one tester's own personal process:

- **Consistency** — the same engagement produces comparable results whoever runs it, and results from different engagements over time can be meaningfully compared.
- **Completeness** — a published framework was built and refined by many practitioners over years specifically to avoid skipped steps; an individual's own improvised checklist has no such guarantee.
- **Legal defensibility** — if an engagement is ever scrutinized after the fact (a dispute, an incident, an audit), "we followed a recognized, published standard" is a genuinely different, stronger position than "we did what seemed reasonable at the time."

### PTES — The Penetration Testing Execution Standard

PTES defines seven phases, in order: **Pre-engagement Interactions**, **Intelligence Gathering**, **Threat Modeling**, **Vulnerability Analysis**, **Exploitation**, **Post Exploitation**, and **Reporting**. It's broad and technique-agnostic — it applies to network, web application, wireless, and social engineering engagements alike, rather than being scoped to any one category from `pentest1-2`'s own list.

**THIS COURSE'S OWN CHAPTER ORDER LOOSELY MIRRORS PTES**

`pentest1-1` / `pentest1-2` map to Pre-engagement Interactions; `pentest1-4` is Intelligence Gathering; `pentest1-5` extends that into active Vulnerability Analysis groundwork; `pentest1-6` is Vulnerability Analysis proper; `pentest1-7` is Exploitation; `pentest1-8` is Post Exploitation; `pentest1-9` is Reporting. This wasn't a coincidence — this course's own structure is deliberately built around PTES's own phase order.

## The OWASP Testing Guide

Where PTES is broad, the OWASP Testing Guide is deliberately narrow and deep: it's the field manual specifically for the **Web Application** scope category named in `pentest1-2`. It's maintained by the same organization behind `owasp1`'s own Top 10 survey, and its testing categories map closely onto vulnerability classes this site already teaches in real depth — `xss1`, `csrf1`, and `sqli1` each correspond to entire testing categories inside the Guide.

## NIST SP 800-115

NIST's *Technical Guide to Information Security Testing and Assessment* is a US government publication, organized around four higher-level phases: **Planning**, **Discovery**, **Attack**, and **Reporting**. It's the framework most likely to be explicitly, contractually required — government contractors and organizations in regulated industries are often required to demonstrate compliance with a named NIST standard specifically, not just "a reasonable methodology."

| FRAMEWORK | MAINTAINED BY | SCOPE/FOCUS | STRUCTURE | BEST FIT |
|---|---|---|---|---|
| PTES | Community/industry practitioners | Broad — any engagement type | 7 phases | General-purpose engagements of any scope category |
| OWASP Testing Guide | OWASP | Narrow — web applications specifically | Category-based (mirrors owasp1's own Top 10) | Web Application-scoped engagements |
| NIST SP 800-115 | US NIST (government) | Broad, compliance-oriented | 4 phases | Government contracts, regulated industries requiring named-standard compliance |

**NO FRAMEWORK IS UNIVERSALLY LEGALLY MANDATORY — BUT CONTRACTS AND REGULATIONS OFTEN NAME ONE SPECIFICALLY**

Following a published framework isn't a general legal requirement in itself. But specific contracts, industry regulations, or compliance frameworks (PCI DSS is a common real example) can explicitly require testing to follow a named standard. Confirming which framework — if any — an engagement is contractually or regulatorily required to follow is itself part of scoping, not an afterthought.

## Hands-On Exercises

**EXERCISE 1**

Using this chapter's own tip-box, map each of PTES's seven phases to the specific chapter of this course that corresponds to it.

View solution

**EXERCISE 2**

A security firm has been hired to test a government contractor's systems, and the contract explicitly requires NIST-standard compliance. Using this chapter's own comparison table, explain why PTES or the OWASP Testing Guide wouldn't satisfy this requirement even if either was followed thoroughly.

View solution

**EXERCISE 3**

Using this chapter's own three reasons (consistency, completeness, legal defensibility), explain specifically why "we followed PTES" is a stronger position after an incident than "we did what seemed reasonable at the time," even if the actual steps taken were nearly identical.

View solution

**CHAPTER 3 QUICK REFERENCE**

- Standardized frameworks exist for three reasons: **consistency, completeness, legal defensibility** — not because improvisation necessarily produces worse individual results
- **PTES** — 7 phases (Pre-engagement, Intelligence Gathering, Threat Modeling, Vulnerability Analysis, Exploitation, Post Exploitation, Reporting), broad/technique-agnostic; this course's own structure mirrors it
- **OWASP Testing Guide** — narrow, web-application-specific, maps onto xss1/csrf1/sqli1/owasp1's own vulnerability categories
- **NIST SP 800-115** — 4 phases (Planning, Discovery, Attack, Reporting), government/compliance-oriented, often contractually required by name
- No framework is universally mandatory, but specific contracts/regulations (e.g. PCI DSS) can require a named one — confirming this is part of scoping
- **Next chapter:** Reconnaissance & OSINT — passive vs. active recon, extending net1-9's and host1's own DNS material

CHAPTER 4 OF 10

# Reconnaissance & OSINT

## Penetration Testing Methodology

Chapter 4 · Reconnaissance & OSINT

`pentest1-3` named this phase Intelligence Gathering in PTES's own terms. This chapter covers what that actually means in practice: building a real picture of a target before touching it, using techniques that range from completely invisible to directly observable by the target.

### Passive vs. Active Reconnaissance

**Passive recon** gathers information without directly interacting with the target's own systems — nothing sent reaches the target, so nothing can appear in their logs. Public records, search engines, social media, cached pages, and WHOIS lookups against a third-party registry all count as passive. **Active recon** means direct interaction with the target's own infrastructure — even something as simple as a DNS query sent to the target's own authoritative nameserver, or a single ping, counts as active, since it touches systems the target actually controls and could show up in their own logs.

The distinction matters because it maps directly onto risk: passive recon carries essentially zero risk of detection or disruption, while active recon — however innocuous it seems — can be logged, can trigger alerts, and some scope agreements explicitly restrict it to specific time windows or exclude particular techniques entirely.

### WHOIS & DNS Enumeration

WHOIS lookups reveal registrant information, registrar, and creation/expiration dates — though privacy services now redact much of what used to be public. DNS enumeration goes further, extending `net1-9`'s own DNS resolution hierarchy material directly into an information-gathering technique: subdomain enumeration, MX records (revealing mail infrastructure and often which third-party provider handles it), TXT records (SPF/DKIM entries and verification tokens that leak which third-party services — analytics, marketing tools, cloud providers — an organization actually uses), and NS records.

A classic (often already-fixed) misconfiguration worth checking is an open zone transfer (AXFR) — a nameserver that will hand over its entire zone file to anyone who asks, rather than only to its own designated secondary servers.

THIS DIRECTLY EXTENDS TWO CHAPTERS ALREADY ON THE SITE

`net1-9` covered the DNS resolution hierarchy itself; `host1` covered managing DNS at a registrar and through Cloudflare. Understanding how DNS is actually managed and configured — the subject of both those chapters — is exactly what makes it possible to know what to look for and what looks unusual during DNS-based recon.

## OSINT — Public Records & Social Media

- **Job postings** — a listing for "a Senior React Developer with AWS and PostgreSQL experience" reveals real technology-stack details for free.
- **Employee enumeration via LinkedIn** — relevant specifically to the Social Engineering scope category from `pentest1-2`.
- **Public code repositories** — accidentally committed credentials, internal hostnames, or revealing comments.
- **Public device search engines** (Shodan/Censys-style) — searching pre-existing, already-collected scan data rather than scanning the target directly, keeping this technique passive.
- **Breach databases** — checking whether an organization's own email addresses have appeared in known public breaches, directly relevant to `bc1`'s own password-security material and previewing `pentest1-7`'s credential-based exploitation.

## Why Recon Quality Determines Everything Downstream

`pentest1-5`'s scanning depends on already knowing what to scan — the IP ranges and subdomains this phase discovers. `pentest1-6`'s vulnerability analysis depends on knowing what's actually running, using the technology-stack clues this phase surfaces. `pentest1-7`'s exploitation sometimes depends directly on valid targets or credentials this phase turns up. A shallow recon phase doesn't just produce a smaller result on its own — it produces a shallow, incomplete attack surface map that every later phase inherits without ever knowing what was missed.

**OSINT still has to respect scope, even though the information itself is public**

Most OSINT is passive and deals only with publicly available information — but `pentest1-2`'s own scope-precision lesson still applies here directly. Some organizations explicitly exclude social-engineering-relevant recon (like enumerating specific employees) from an engagement, or don't want personal social media research conducted on their staff at all, even using only public information. Whether OSINT of this kind is included has to be settled explicitly in the written scope, the same way `pentest1-2` required for every other category — "it's public information" doesn't substitute for it being within the agreed scope.

## Hands-On Exercises

**EXERCISE 1**

Classify each of the following as passive or active recon, and explain why: (a) a WHOIS lookup, (b) a port scan against the target's own server, (c) a LinkedIn search for the target's employees, (d) a DNS query sent directly to the target's own authoritative nameserver.

View solution

**EXERCISE 2**

Explain, using this chapter's own "why recon quality determines everything downstream" section, specifically how a shallow reconnaissance phase would cause concrete problems in pentest1-5, pentest1-6, and pentest1-7.

View solution

**EXERCISE 3**

Explain why OSINT research on an organization's employees needs to be explicitly addressed in the written scope agreement, even though the information being gathered is entirely public — tying your answer to pentest1-2's own scope-precision material.

View solution

### CHAPTER 4 QUICK REFERENCE

- **Passive** — no interaction with target systems, essentially undetectable · **Active** — direct interaction, loggable/detectable even when innocuous
- WHOIS + DNS enumeration (subdomains, MX, TXT, NS, zone transfer checks) extends net1-9's resolution hierarchy and host1's DNS-management material
- OSINT sources: job postings, employee enumeration, public code repos, Shodan/Censys-style device search, breach databases (ties to bc1)
- Recon quality bounds every later phase — pentest1-5/6/7 all inherit whatever this phase missed
- Public information ≠ automatically in scope — OSINT scope must still be addressed explicitly, per pentest1-2
- **Next chapter:** Scanning & Enumeration — building directly on net1-10's own diagnostic tooling (ping, traceroute, ss, tcpdump)

CHAPTER 5 OF 10

# Scanning & Enumeration

## Penetration Testing Methodology

Chapter 5 · Scanning & Enumeration

`pentest1-4` produced a candidate list of assets — IP ranges, subdomains, hints about technology in use. This phase confirms which of those candidates are actually alive, what ports are actually open on them, and what's actually running behind each one — turning a speculative list into hard, verified evidence.

### Host Discovery

Before scanning individual ports, a real engagement first confirms which hosts in a range are actually reachable — most simply, via a ping sweep, extending `net1-10`'s own diagnostic-tooling chapter directly. But this needs an important caveat: many firewalls block ICMP by default, so a host that doesn't respond to a ping isn't necessarily down — it may simply be filtering the exact protocol being used to check.

### Port Scanning

A port scan sends probes across a range of ports and observes the response to classify each one as **open**, **closed**, or **filtered** — filtered meaning no response was returned at all, which is itself ambiguous (it could mean a firewall is silently dropping the probe, or that nothing is listening and the host simply doesn't reply).

| SCAN TYPE | MECHANISM | HANDSHAKE COMPLETED? | TRADE-OFF |
|---|---|---|---|
| TCP Connect Scan | Completes a full connection via the OS's own networking stack | Yes — full three-way handshake | Reliable, but easily logged — a completed connection is the most visible kind of event |
| SYN Scan (half-open) | Sends SYN, reads the SYN-ACK response, then sends RST instead of completing with ACK | No — deliberately incomplete | Faster and stealthier — many logging systems only record fully-established connections |

**SYN SCANNING IS NET1-7'S OWN THREE-WAY HANDSHAKE, DELIBERATELY INTERRUPTED**

`net1-7` covered TCP's three-way handshake — SYN, SYN-ACK, ACK — in full. A SYN scan uses exactly that same exchange, but stops after step two: it sends the SYN, reads whether a SYN-ACK comes back (which alone reveals whether the port is open), and then sends an RST instead of the final ACK, rather than ever completing the handshake

and establishing a real connection. This is precisely why it's called "half-open," and precisely why it tends to leave less of a trace than a full TCP Connect scan.

## Service & Version Detection, Banner Grabbing

Confirming a port is open only answers "is something listening here" — the next question is *what*. Many services announce their own software name and version on connection — an SSH server, for instance, commonly announces a banner like `OpenSSH_8.9` before any authentication happens at all. Tools like nmap automate this with dedicated version-detection modes (`-sV`), sending protocol-specific probes and comparing the responses against a database of known service signatures.

A specific, identified version is directly useful — it immediately narrows down which known vulnerabilities are even worth considering in `pentest1-6`, rather than starting from a generic "this port is open" with no further information.

## Building a Real Attack Surface Map

Combining every piece from this phase — `pentest1-4`'s discovered candidate assets, confirmed-alive hosts, confirmed-open ports, and identified service versions — produces a genuine, evidence-based map of the target's actual attack surface, replacing `pentest1-4`'s own broader, more speculative candidate list. This map is the direct, concrete input `pentest1-6`'s vulnerability analysis phase works from.

**AGGRESSIVE SCANNING CAN CAUSE REAL DISRUPTION, NOT JUST NOISE**

A fast, aggressive scan isn't just "loud" in a logging sense — against fragile or legacy systems (older embedded devices, industrial control systems, some IoT hardware), an aggressive scan has genuinely crashed real production equipment in documented real-world cases. This is exactly why scan intensity and timing sometimes need to be addressed explicitly in the written scope agreement from `pentest1-2` — not just which hosts are in bounds, but how hard and how fast they can safely be probed.

## Hands-On Exercises

**EXERCISE 1**

Explain the difference between a TCP Connect scan and a SYN scan, tying your answer explicitly to net1-7's own three-way handshake material. Why is a SYN scan considered "stealthier"?

View solution

**EXERCISE 2**

A ping sweep against a target's IP range gets no response from a specific address. Explain why this doesn't necessarily mean the host is down, and what the more careful conclusion is.

View solution

**EXERCISE 3**

Using this chapter's own warn-box, explain why scan intensity and timing might need to be explicitly addressed in the written scope agreement from pentest1-2, with a concrete example of what could go wrong without that.

View solution

### CHAPTER 5 QUICK REFERENCE

- Host discovery builds directly on net1-10's diagnostic tools — but a lack of ping response ≠ confirmed down (ICMP is often filtered)
- Port states: **open / closed / filtered** — filtered is ambiguous (could be a firewall, could be nothing listening)
- **TCP Connect scan** — full handshake, reliable, easily logged · **SYN scan** — stops after SYN-ACK, deliberately never completes net1-7's own three-way handshake, faster/stealthier
- Version detection/banner grabbing narrows the field for pentest1-6's vulnerability analysis directly
- Recon (pentest1-4) + confirmed hosts/ports/versions (this chapter) = the real attack surface map
- Aggressive scanning can cause real disruption on fragile systems — scan intensity/timing belongs in written scope, per pentest1-2
- **Next chapter:** Vulnerability Analysis — the site's own existing xss1/csrf1/sqli1/bc1/dbsec1/owasp1 "toolkit," formally named

CHAPTER 6 OF 10

# Vulnerability Analysis — The Toolkit This Site Already Built

## Penetration Testing Methodology

Chapter 6 · Vulnerability Analysis — The Toolkit This Site Already Built

`pentest1-1` opened this course with a claim worth restating in full now that it's time to prove it: a pentest is a repeatable, disciplined process, and every vulnerability class this site already teaches in depth is a *tool* this process calls on during one specific phase — not the process itself. This is that phase.

### From Attack Surface Map to Candidate Vulnerabilities

`pentest1-5` produced a real, evidence-based map: which hosts are alive, which ports are open, what service versions are running. This phase's job is to take that concrete map and match it against *already-documented* categories of weakness — not to invent new attack techniques, but to recognize which known vulnerability classes plausibly apply to what was actually found.

### The Toolkit This Site Already Built

This site already teaches the vulnerability-class knowledge this phase draws on, in real depth, across six existing Security courses:

| ATTACK-SURFACE SIGNAL FROM PENTEST1-5 | RELEVANT COURSE | WHAT IT TEACHES |
|---|---|---|
| A page reflecting user input back into its own output | xss1 | Cross-Site Scripting — the three types, injection contexts, defenses |
| A form performing a state-changing action | csrf1 | CSRF — the confused-deputy pattern, tokens, SameSite cookies |
| A database-backed feature accepting user input (search, filters, login) | sqli1 | SQL Injection — data-vs-code, UNION/blind techniques, parameterization |
| A login form or session-based feature | bc1 | Authentication & Session Security — password storage, sessions, MFA, JWTs |

| ATTACK-SURFACE SIGNAL FROM PENTEST1-5 | RELEVANT COURSE | WHAT IT TEACHES |
|---|---|---|
| An exposed or reachable database port | dbsec1 | Database Security — access control, network exposure, encryption |
| Everything else — a systematic survey pass | owasp1 | The full OWASP Top 10, including categories with no dedicated course (Insecure Design, Misconfiguration, Outdated Components, Integrity Failures, Logging Failures, SSRF) |

Each of those courses taught how a specific vulnerability class works, how to recognize it, and how to defend against it. This phase of a real engagement is the moment that same knowledge gets applied in the other direction — using it to recognize a discovered piece of attack surface as a plausible candidate worth investigating further, rather than devising something novel from scratch.

## CVE Databases & Known-Vulnerability Matching

`pentest1-5`'s own banner-grabbing and version-detection output enables a genuinely different kind of matching: once a specific service and version is known, it can be checked against public vulnerability databases (CVE/NVD) for already-documented, publicly disclosed flaws affecting that *exact* version.

This is a meaningfully different tool than the vulnerability-class matching above. Vulnerability-class knowledge (from `xss1` / `csrf1` / `sqli1` / `bc1`) is the right tool for an organization's own **custom application code** — there's no CVE for someone's own bespoke bug, since it was never publicly disclosed anywhere. CVE matching is the right tool for **known third-party software** — an outdated web server, a specific library version — where the exact flaw has already been found and documented by someone else.

## Matching, Not Inventing

It's worth restating the chapter's own core discipline directly: this phase's job is to recognize known patterns in what was actually discovered, not to devise brand-new attack techniques. Discovering genuinely novel vulnerabilities is a distinct, much rarer skill. The overwhelming majority of real, professional penetration tests find real, exploitable issues through exactly this kind of methodical, thorough matching against an already-documented body of knowledge — not by stumbling onto something never seen before.

**A PATTERN MATCH ISN'T PROOF OF A REAL VULNERABILITY**

Finding an attack-surface element that superficially resembles a known vulnerability class doesn't confirm the vulnerability is actually real or exploitable — a login form might already be protected by rate limiting and MFA; a reflected input might already be properly encoded. This phase produces *candidates*, not confirmed findings. `pentest1-7` is specifically about actually proving impact for a candidate, rather than assuming impact from a surface-level resemblance alone.

**THIS IS THE CHAPTER PENTEST1-1 WAS BUILDING TOWARD**

`pentest1-1` named the site's own vulnerability courses as tools this process calls on, not the process itself. This chapter is where that claim becomes concrete and literal — every one of those six courses just became a direct input to a specific phase of a real methodology, exactly as promised.

## Hands-On Exercises

**EXERCISE 1**

pentest1-5's scan found: a login form, a product search feature querying a database, and a directly reachable database port on 3306. Using this chapter's own table, identify which earlier course's vulnerability-class material is most directly relevant to each of the three findings, and why.

View solution

**EXERCISE 2**

Explain the difference between vulnerability-class matching (the xss1/csrf1/sqli1/bc1 style) and CVE-based known-vulnerability matching, and give one example of when each is the correct tool to use.

View solution

**EXERCISE 3**

Using this chapter's own warn-box, explain why finding a surface-level pattern match to a known vulnerability class isn't the same as confirming a real, exploitable vulnerability — and name the specific chapter where that gap actually gets closed.

View solution

**CHAPTER 6 QUICK REFERENCE**

- This phase matches pentest1-5's own attack surface map against KNOWN vulnerability classes — it doesn't invent new attack techniques
- The site's own toolkit: **xss1, csrf1, sqli1, bc1, dbsec1** for specific vulnerability classes; **owasp1** as the systematic survey pass covering everything else
- **Vulnerability-class matching** — the right tool for custom application code (no CVE exists for a bespoke bug) · **CVE/NVD matching** — the right tool for known third-party software/services with a specific, already-documented version
- A surface-level pattern match produces a candidate, not a confirmed finding — pentest1-7 proves actual impact

- This chapter is the direct, literal payoff of pentest1-1's own opening claim about this site's vulnerability courses being "tools this process calls on"
- **Next chapter:** Exploitation — Proving Impact Without Causing Harm

CHAPTER 7 OF 10

# Exploitation — Proving Impact Without Causing Harm

## Penetration Testing Methodology

Chapter 7 · Exploitation — Proving Impact Without Causing Harm

`pentest1-6` closed with an open question: a candidate produced by pattern matching isn't a confirmed finding. This chapter closes that gap — but draws a real, professional line around exactly how far confirming it should go.

### From Candidate to Confirmed — What Exploitation Actually Proves

`pentest1-6`'s candidates are attempted here, hands-on, to determine whether they're actually real, actually exploitable, and what genuine impact they enable in this specific target's own context — resolving the exact gap `pentest1-6`'s own warn-box left open, where an existing mitigation (rate limiting, output encoding) could mean a superficial pattern match isn't actually exploitable at all.

### Proof-of-Concept vs. Full Exploitation

A **proof-of-concept (PoC)** is the minimum action needed to demonstrate a vulnerability is real, without going further than necessary. **Full exploitation** continues past that point — and is very often not necessary to prove anything additional.

| CANDIDATE (FROM PENTEST1-6) | PROOF-OF-CONCEPT | UNNECESSARY FULL EXPLOITATION |
|---|---|---|
| SQLi candidate (sqli1) | A UNION-based query extracting one non-sensitive row (e.g. a database version string) | Extracting an entire customer table, or modifying/deleting real data |
| Stored XSS candidate (xss1) | A harmless payload proving arbitrary script execution in another user's browser context | Actually stealing a real user's live session cookie |
| Weak authentication candidate (bc1) | Demonstrating a successful login using a guessed/default credential once | Logging into every account the credential pattern might affect |

### Why "I Could Have" Is Often the Correct Stopping Point

Once a vulnerability is confirmed exploitable in principle — the UNION query worked, the script executed — extracting every row or every session isn't "being more thorough." It adds no additional evidentiary value: the PoC already proved everything the client needs to know. "I could have extracted the entire customers table, but stopped after confirming access to a single non-sensitive record" is both scientifically sufficient and the professionally correct choice — not a missed opportunity.

## When the Target Is a Live Production System

Most real engagements test systems actual customers are actively using, not an isolated lab — so the risk of "going further than necessary" isn't hypothetical, it's a real, current risk to real people and real data. This is exactly why scope, per `pentest1-2`, sometimes explicitly restricts testing to low-traffic windows, or arranges for a cloned staging environment specifically to remove this tension entirely rather than relying on tester discipline alone.

**AUTHORIZATION IS NOT A BLANK CHECK**

Even with written authorization from `pentest1-1`, causing unnecessary damage beyond what's needed to prove a vulnerability can still expose a tester to real liability. A Rules of Engagement document typically authorizes **testing** — not unrestricted destruction, data exfiltration, or service disruption — and often explicitly limits what's permitted (no data destruction, no denial-of-service, no touching real customer data beyond a single proof record). Going beyond a proof-of-concept isn't automatically covered just because the underlying system was in scope.

**THE SAME DISCIPLINE CARRIES FORWARD INTO POST-EXPLOITATION**

`pentest1-8` applies this exact "prove it, don't cause unnecessary harm" principle to lateral movement and privilege escalation specifically — the same PoC-not-full-exploitation logic, one layer deeper into the target's environment.

## Hands-On Exercises

**EXERCISE 1**

A pentest1-6 candidate confirmed a SQL injection point in a login form. Describe what a proper proof-of-concept would look like, what would cross into unnecessary full exploitation, and explain why the PoC alone is sufficient.

View solution

**EXERCISE 2**

Explain why "I could have extracted the entire database" is often the professionally correct stopping point, even though the tester technically could go further — tying your answer to this chapter's own reasoning about evidentiary value vs. added risk.

View solution

**EXERCISE 3**

Explain why written authorization from pentest1-1 doesn't act as a blanket permission to cause any damage during exploitation — what does authorization typically cover, and what does it typically not cover?

View solution

### CHAPTER 7 QUICK REFERENCE

- Exploitation confirms whether a pentest1-6 candidate is actually real and exploitable — closing that chapter's own "candidate, not confirmed finding" gap
- **Proof-of-concept** — the minimum action needed to prove exploitability · **full exploitation** — going further, usually with no added evidentiary value
- "I could have" is often the professionally correct stopping point — extracting more data adds risk without adding proof
- Live production systems mean the risk of going too far is real, not hypothetical — scope sometimes uses low-traffic windows or a staging environment to remove the tension entirely
- Authorization covers **testing**, not unrestricted damage — a signed RoE is not a blank check
- **Next chapter:** Post-Exploitation & Privilege Escalation — the same discipline applied one layer deeper

CHAPTER 8 OF 10

# Post-Exploitation & Privilege Escalation

## Penetration Testing Methodology

Chapter 8 · Post-Exploitation & Privilege Escalation

`pentest1-7` proved a candidate was real, stopping deliberately at proof-of-concept. But a single confirmed foothold rarely answers the full question a client actually cares about: if a real attacker got exactly this same foothold, what could they genuinely reach from there? This chapter asks that question directly — with `pentest1-7`'s own discipline still fully in force.

### Privilege Escalation

Privilege escalation means gaining a higher level of access than the initial foothold provided — moving from a low-privilege web application user to administrative access on the underlying server, or from a standard database account to a database administrator. This is usually called **vertical** escalation, since it moves upward within the same general context. A related but distinct concept, **horizontal** movement, means reaching a different account at roughly the same privilege level (another user's own data, rather than a genuinely higher-privileged role) — worth distinguishing by name even though both matter to a real assessment.

**A GENUINE, IF DISTANT, ECHO OF ASSEMBLY2-8'S OWN PRIVILEGE RINGS**

`assembly2-8` formalized privilege levels in hardware terms — ring 0 (full, kernel-level access) versus ring 3 (restricted, user-mode access) — and the real, hardware-enforced page-fault boundary between them. Privilege escalation in a real assessment is a genuine conceptual echo of exactly that boundary: an application typically runs in a deliberately restricted context, and successful privilege escalation means finding a way across into a genuinely more privileged one. Most real-world privilege escalation happens at the OS-account level (a regular user account reaching root/administrator) rather than literally crossing a hardware protection ring — but in the most severe cases, an OS-level exploit can achieve genuine kernel-level access, which is the literal crossing assembly2-8 described in hardware terms. The parallel is real, not just a name reused — a lower-privileged context is structurally unable to do certain things until it crosses a boundary into a higher one — but it's a conceptual echo, not a claim that every privilege escalation is literally a ring transition.

### Lateral Movement

Lateral movement means using an established foothold on one system to reach other systems on the same network that weren't directly reachable or vulnerable from outside — directly relevant to the Internal engagement category from `pentest1-2`. A single compromised low-value system can become a stepping

stone deeper into a network that was otherwise well-defended at its own perimeter. Credential reuse is a common enabler here: a credential surfaced during `pentest1-4`'s own breach-database OSINT, or captured live on one system, can turn out to work on entirely different systems — a real, practical extension of `bc1`'s own password-security material.

## The "Assumed Breach" Mentality

Assumed breach engagements deliberately start from a different position: rather than asking "can an attacker get in at all," the tester is given an already-compromised low-privilege account or workstation access from the start, and the engagement asks "given that a foothold already exists, how much access can genuinely be reached from here." This connects directly to `pentest1-2`'s own Gray Box (partial starting access) and Internal engagement categories.

This mentality matters because real breaches rarely start with a sophisticated zero-day — they usually start with something mundane, like a phishing email or a single leaked credential. Understanding what happens *after* that ordinary, unglamorous initial foothold is very often more operationally valuable to a real organization than one more confirmation that a perimeter can theoretically be breached at all.

**PENTEST1-7'S OWN DISCIPLINE APPLIES HERE WITH EVEN HIGHER STAKES**

Post-exploitation activity happens deeper inside a network, frequently touching systems and data that were never the original vulnerable target at all. The proof-of-concept-not-full-exploitation line from `pentest1-7` matters just as much here — arguably more, since lateral movement risks touching far more systems than the single original vulnerability ever did. Confirming that lateral movement to another system is *possible* is the goal; actually rummaging through that system's real data is not.

## Hands-On Exercises

**EXERCISE 1**

Explain the difference between vertical privilege escalation and lateral movement, using this chapter's own definitions, with one concrete example of each.

View solution

**EXERCISE 2**

Explain the conceptual connection between privilege escalation and assembly2-8's own ring 0/ring 3 material — and explain the honest limit this chapter places on that connection.

View solution

**EXERCISE 3**

Explain the "assumed breach" testing philosophy and why it's often more operationally valuable to a real organization than a traditional test starting entirely from the outside, tying your answer to pentest1-2's own Gray Box and Internal engagement categories.

View solution

## CHAPTER 8 QUICK REFERENCE

- **Vertical privilege escalation** — reaching a genuinely higher privilege level (user → admin/root) · **Lateral movement** — using one foothold to reach other systems on the same network
- Privilege escalation is a genuine, honest, distant echo of assembly2-8's own ring 0/ring 3 boundary — most real cases are OS-account-level, not literal hardware ring transitions
- Lateral movement often relies on credential reuse — ties directly to bc1's password material and pentest1-4's own breach-database OSINT
- **Assumed breach** — starting from an already-compromised foothold (a Gray Box/Internal variant per pentest1-2) rather than testing the perimeter itself; often more operationally valuable since real breaches usually start with something mundane
- pentest1-7's PoC-not-full-exploitation discipline applies here with even higher stakes — more systems are at risk of unnecessary exposure
- **Next chapter:** Reporting — The Actual Deliverable

CHAPTER 9 OF 10

# Reporting — The Actual Deliverable

## Penetration Testing Methodology

Chapter 9 · Reporting — The Actual Deliverable

`pentest1-4` through `pentest1-8` produced real, confirmed findings. None of that has any value to the client until it's written down clearly, prioritized honestly, and communicated to the right audience. This chapter is about that final, essential step.

### The Report Is the Product

The exploitation itself leaves no lasting artifact — once testing ends, everything that happened during `pentest1-4` through `pentest1-8` only continues to exist for the client in one form: the written report. A pentest that finds real, serious vulnerabilities but produces a poor report has, in a real and practical sense, failed to deliver the thing the client actually paid for.

This closes a loop opened all the way back in `pentest1-2`: the engagement began with a written document — the scope agreement and Rules of Engagement — and it ends with a written document, the report. Both are the tangible, contractual artifacts of the entire engagement; the actual testing happens in between them, but the client's lasting deliverable is written on both ends.

### Two Audiences, Two Sections

| SECTION | AUDIENCE | CONTENT |
|---|---|---|
| Executive Summary | Non-technical decision-makers, budget-holders | Overall risk posture, business impact, high-level statistics (e.g. "3 Critical, 6 High findings"), no deep technical detail |
| Technical Findings | Engineers and IT staff who will actually fix each issue | Full technical detail, exact reproduction steps, affected systems, evidence (screenshots, request/response pairs) |

Both sections are necessary, for symmetric reasons: a report with only technical detail rarely secures the budget or priority needed from people who don't have time to parse it; a report with only an executive summary gives the engineers who need to fix each issue nothing concrete to act on. A good report serves both audiences without forcing either one to wade through content meant for the other.

## CVSS — Common Vulnerability Scoring System

Not every finding deserves equal urgency, and CVSS is the standardized way to communicate that consistently. It scores a vulnerability across several metrics — attack vector, attack complexity, privileges required, user interaction needed, and impact on confidentiality/integrity/availability — combining them into a 0–10 score and a Low/Medium/High/Critical rating. Standardization matters here for the same reason `pentest1-3`'s own frameworks mattered: a "Critical" finding should mean roughly the same thing regardless of which tester or firm wrote the report.

## Remediation Guidance

A finding without a fix isn't actionable — the report needs concrete remediation guidance, not just a description of the weakness. This is where this course's own "toolkit" chapters (`pentest1-6`) pay off a second time, now applied to the defensive side rather than the offensive matching side: a SQL injection finding's remediation should point specifically at `sqli1`'s own parameterized-query material; an XSS finding at `xss1`'s own output-encoding material; a weak-authentication finding at `bc1`'s own material on password hashing and MFA. Remediation guidance should generally be ordered by CVSS severity — though a realistic, useful report is honest that both severity *and* ease of fix genuinely matter to a client's actual remediation roadmap, not severity alone.

**ONLY CONFIRMED FINDINGS BELONG IN THE REPORT**

A report that dumps raw scanner output or vague "could potentially be vulnerable" language undermines trust and actionability. Everything in the report should trace back to something actually confirmed via proof-of-concept in `pentest1-7` — not an unverified candidate from `pentest1-6`'s own pattern-matching phase that was never actually tested. Reporting an unconfirmed candidate as though it were a proven finding risks sending a client to fix something that may not actually be exploitable, while genuine findings compete for the same limited attention.

**THE REPORT CLOSES THE LOOP PENTEST1-2 OPENED**

A written scope document opened this engagement; a written report closes it. Everything between the two — recon, scanning, matching, exploitation, post-exploitation — only has lasting value to the client because it's captured here.

## Hands-On Exercises

**EXERCISE 1**

Explain why the report, not the exploitation itself, is the actual deliverable a client is paying for. What happens to the value of a genuinely confirmed finding that's never properly documented?

View solution

**EXERCISE 2**

Explain why a report needs both an executive summary and a technical findings section, using this chapter's own reasoning about the two different audiences.

View solution

**EXERCISE 3**

Explain why a SQL injection finding's remediation guidance should point back at sqli1 specifically, and explain this chapter's own warn-box point about why only pentest1-7-confirmed findings — not raw pentest1-6 candidates — belong in the report.

View solution

### CHAPTER 9 QUICK REFERENCE

- The report is the actual deliverable — exploitation leaves no lasting artifact once testing ends
- Closes the loop pentest1-2 opened: a written scope document opened the engagement, a written report closes it
- **Executive Summary** — business impact, high-level stats, non-technical · **Technical Findings** — reproduction steps, evidence, for the engineers fixing it
- **CVSS** — a standardized 0–10 severity score, for the same consistency reason pentest1-3's own frameworks mattered
- Remediation guidance reuses pentest1-6's own toolkit courses (sqli1/xss1/bc1/etc.) on the defensive side
- Only pentest1-7-confirmed findings belong in the report — never an unverified pentest1-6 candidate
- **Next chapter:** Capstone — Scoping and Planning a Real Engagement

CHAPTER 10 OF 10

# Capstone: Scoping and Planning a Real Engagement

## Penetration Testing Methodology

Chapter 10 · Capstone: Scoping and Planning a Real Engagement

Nine chapters built one continuous process: authorization, scope, framework, recon, scanning, matching, exploitation, post-exploitation, reporting. This capstone walks that entire process end to end for a single fictional target — Meridian Retail Co., a fictional online retailer — combining every prior chapter into one coherent engagement.

### The Scenario

Meridian Retail Co. is preparing for a major holiday sales push and wants its online store and supporting infrastructure tested before traffic peaks. They've requested a Gray Box engagement (per `pentest1-2`) — testers will be given a standard customer account, but no source code or administrative access — scoped to their Web Application and Network categories.

### Sample Rules of Engagement Document

```
MERIDIAN RETAIL CO. — RULES OF ENGAGEMENT (SAMPLE, FICTIONAL)
================================================================
Client:                Meridian Retail Co.
Authorized signatory:  [Name], VP of Engineering
Testing firm:          [Firm Name]
Engagement type:       Gray Box (pentest1-2) — one standard
                        customer account provided, no source
                        code or admin access
Scope — IN:
  - shop.meridianretail.example (production web application)
  - api.meridianretail.example
  - IP range 203.0.113.0/24 (Meridian's own hosted infrastructure)
Scope — OUT:
  - payments.meridianretail.example (third-party payment
    processor infrastructure, NOT owned by Meridian — see
    pentest1-2's own shared-infrastructure warning)
  - Any Meridian employee's personal social media (OSINT on
    named individuals excluded per pentest1-4's own scope note)
Testing window:        Weekdays, 22:00-06:00 local time only
                        (low-traffic hours, per pentest1-5's own
                        scan-intensity guidance)
```

```
Methodology:            PTES (pentest1-3)
Escalation contact:     [Name], available 24/7 during the window
Emergency stop:         Testing halts immediately on request from
                        either party; no penalty for invoking it
Explicitly authorized:  Testing up to proof-of-concept level
                        (pentest1-7) — NOT data destruction,
                        NOT denial-of-service, NOT extraction
                        beyond a single non-sensitive record
Signatures:             _____  _____
                        Client          Testing Firm
================================================================
```

## Walking Through the Methodology

1. **Pre-engagement (pentest1-1 / pentest1-2):** the RoE above is signed by someone with real authority over Meridian's own systems, before any technical activity begins.
2. **Framework (pentest1-3):** the engagement follows PTES's seven phases, matching this course's own structure directly.
3. **Reconnaissance (pentest1-4):** passive WHOIS/DNS enumeration reveals Meridian's own hosting provider and mail infrastructure; a public job posting reveals a React/Node.js stack; employee LinkedIn presence is noted but not investigated further, per the RoE's own exclusion.
4. **Scanning & Enumeration (pentest1-5):** confirms `shop.meridianretail.example` and `api.meridianretail.example` are alive, with ports 443 and — unexpectedly — 3306 (MySQL) directly reachable; a SYN scan run only during the authorized window identifies service versions.
5. **Vulnerability Analysis (pentest1-6):** the reachable 3306 port is flagged as a `dbsec1`-style candidate (a database that should never be directly internet-facing); the checkout page's search field is flagged as a `sqli1` candidate; the login form is flagged as a `bc1` candidate.
6. **Exploitation (pentest1-7):** a single non-sensitive proof-of-concept row (the database version string) is extracted via the checkout search field, confirming the SQLi candidate; the login form's session cookie is confirmed to be missing the `Secure` flag. Testing stops at proof-of-concept for both — no customer data is touched.
7. **Post-Exploitation (pentest1-8):** using the Gray Box starting account, the tester confirms the compromised checkout server can also reach an internal admin panel not exposed externally — a real lateral-movement path — and stops there, without logging into the panel itself.
8. **Reporting (pentest1-9):** the final report includes an executive summary (three findings, one Critical, two High, business impact framed around the upcoming sales push) and technical findings with CVSS scores and reproduction steps, with remediation pointing directly at `dbsec1` (network isolation for the database), `sqli1` (parameterized queries), and `bc1` (proper cookie flags).

## Chapter Attribution

| CAPSTONE ELEMENT | CHAPTER |
|---|---|
| Signed RoE, authorization precondition | pentest1-1 |
| Gray Box selection, in/out scope, shared-infrastructure exclusion | pentest1-2 |
| PTES as the adopted framework | pentest1-3 |
| WHOIS/DNS/job-posting recon, OSINT scope exclusion | pentest1-4 |
| Host/port confirmation, SYN scan, restricted testing window | pentest1-5 |
| Matching findings to dbsec1/sqli1/bc1 | pentest1-6 |
| Single-row PoC, stopping short of full exploitation | pentest1-7 |
| Lateral movement confirmation without full access | pentest1-8 |
| Executive summary, CVSS, remediation pointing at earlier courses | pentest1-9 |

**HONEST SCOPE NOTE**

This capstone is a walkthrough against a fictional target, not a live-fire exercise against real infrastructure — no actual system named here exists. It does not attempt to teach specific tool usage beyond the conceptual scanning coverage already grounded in `net1-10`; real engagements require real, hands-on familiarity with tools like nmap that this course deliberately doesn't attempt to substitute for. And nothing in this course constitutes legal advice — before conducting any real engagement, consult a qualified lawyer and your organization's own compliance function regarding authorization, scope, and applicable law.

**THE THROUGHLINE, CLOSED**

`pentest1-1` opened this course by claiming that a pentest is a repeatable, disciplined process, and that the site's own vulnerability courses are tools this process calls on — not the process itself. This capstone is the proof: nine distinct chapters, each with its own job, produced one coherent, professional engagement for a single fictional target, with `dbsec1` / `sqli1` / `bc1` called on at exactly the moment the process needed them.

## Hands-On Exercises

**EXERCISE 1**

Using this chapter's own sample RoE, explain specifically why payments.meridianretail.example is listed as out of scope, tying your answer to pentest1-2's own shared-infrastructure warning.

View solution

**EXERCISE 2**

Using the chapter's own methodology walkthrough, explain what would go wrong if the exploitation step (step 6) had been attempted before the pre-engagement step (step 1) actually completed.

View solution

**EXERCISE 3**

Using the chapter-attribution table, explain how this capstone embodies pentest1-1's own opening claim that this site's vulnerability courses are "tools this process calls on, not the process itself."

View solution

### CHAPTER 10 QUICK REFERENCE — COURSE COMPLETE

- A full sample RoE combines pentest1-1's authorization precondition and pentest1-2's precise scope, testing window, and out-of-scope exclusions
- The nine-phase walkthrough maps directly onto pentest1-1 through pentest1-9, in order
- Remediation in the final report calls directly on dbsec1, sqli1, and bc1 — the exact "toolkit" pentest1-1 and pentest1-6 named
- Honest scope note: fictional target only, no tool-usage tutorial beyond net1-10's own conceptual coverage, no legal advice
- This closes the full 10-chapter Penetration Testing Methodology course