PRACTICAL FIELD GUIDE · OFFENSIVE & DEFENSIVE SECURITY

Security Testing & Assessment Toolkit

A comprehensive reference to the tools, methodology and workflow used to probe a web platform for access flaws — from reconnaissance and network scanning to web exploitation, cloud, containers and source-code analysis. Tailored for self-hosted infrastructure (FastAPI, systemd, reverse proxy, AWS).

AUTHOR Guillaume Oneill

COMPANY Ideo-Lab

CONTACT contact@ideo-lab.com

VERSION 1.0 · 16 June 2026

RECON SCAN ENUM EXPLOIT ANALYSE REPORT

Contents

1 Scope, ethics and the assessment mindset

Security testing your own infrastructure is the single most effective way to find an access flaw before someone else does. This guide treats the exercise as a **grey-box assessment run by the system owner**: you know your architecture, you probe what an attacker would see from the outside, you verify what an attacker would reach from the inside, and then you remediate. It catalogues the tools used at each stage, how to install them, the commands that matter, and how to read the results.

> **Read before you scan — legal boundary**
> Only run these tools against systems you own or have explicit written authorisation to test. Port scanning, active web scanning, brute-forcing and exploitation against third-party systems can be a criminal offence in most jurisdictions, regardless of intent. Everything below assumes the target is your own platform. When in doubt, scope it in writing first.

Two principles run through the whole document. First, **progressive intrusiveness**: start passive (observe, fingerprint) and only escalate to active and exploitative techniques deliberately, ideally against a staging copy. Second, **network first**: on self-hosted infrastructure, the highest-value finding is almost always a port or service exposed by mistake, not an exotic application bug. Spend your first hour there.

How tools are categorised here

Tools are grouped by the phase of work they serve rather than alphabetically, because in practice you reach for them in sequence. Many tools straddle categories (sqlmap both detects and exploits; ffuf both discovers content and fuzzes parameters); they are listed where they are most often first used, with cross-references where useful.

2 The assessment methodology — six phases

Professional assessments follow a repeatable lifecycle. The names vary between frameworks (PTES, OWASP WSTG, OSSTMM, the MITRE ATT&CK; kill-chain), but the shape is consistent. Mapping each tool to a phase keeps the work structured and the report coherent.

Phase	Goal	Representative tools
1. Reconnaissance	Map the target's footprint without touching it directly	whois, dig, Amass, theHarvester, Shodan
2. Scanning	Identify live hosts, open ports and running services	nmap, masscan, RustScan
3. Enumeration	Extract detail from each service (versions, shares, users)	enum4linux, snmpwalk, nmap NSE
4. Vuln. analysis	Match findings to known weaknesses and misconfigs	Nuclei, OpenVAS, Nessus, ZAP
5. Exploitation	Confirm a flaw is real and reachable (proof, not damage)	sqlmap, Metasploit, Hydra
6. Reporting	Document, prioritise (CVSS), recommend, re-test	Dradis, Faraday, spreadsheets

Passive vs active vs exploitative
Passive techniques observe data that is already public (DNS records, certificates, search-engine results) and are essentially risk-free. Active techniques send traffic to the target (port scans, web crawls) and are detectable. Exploitative techniques attempt to trigger the flaw and can alter state or cause downtime — reserve them for staging or a backed-up environment.

Three families of testing

- **DAST (Dynamic):** tests the running application from the outside, with no source access — ZAP, Burp, Nikto, Nuclei. Finds what an external attacker finds.
- **SAST (Static):** analyses source code or binaries without running them — Semgrep, Bandit, CodeQL. Finds flaws earlier and deeper, including ones not yet reachable.
- **IAST / SCA:** instruments the app at runtime, or scans dependencies for known vulnerable components — Snyk, OWASP Dependency-Check, Trivy. Best run in CI.

3 Reconnaissance & OSINT

Reconnaissance builds a picture of the attack surface from publicly available data, without sending anything intrusive to the target. For a self-hosted site this means enumerating domains and subdomains, mail and name servers, IP ranges, exposed services indexed by third parties, and any credentials or hostnames leaked online.

whois / dig / host — the fundamentals

Before any tool, the built-in DNS and registration utilities answer the first questions: who owns the domain, which name servers are authoritative, and what records resolve.

DNS FUNDAMENTALS

```
whois your-domain.com                # registrar, dates, contacts
dig your-domain.com ANY +noall +answer
dig +short A your-domain.com         # IPv4 behind the name
dig +short MX your-domain.com        # mail servers
dig +short TXT your-domain.com       # SPF/DKIM/verification records
dig -x 203.0.113.10                  # reverse lookup on an IP
```

Subdomain enumeration — Amass, Subfinder, assetfinder

Forgotten subdomains (staging, old admin panels, internal tools accidentally made public) are a classic source of access flaws. OWASP Amass is the most thorough, combining passive sources with optional active resolution; Subfinder is faster and passive-only.

SUBDOMAIN DISCOVERY

```
# OWASP Amass — passive enumeration
amass enum -passive -d your-domain.com -o subs.txt

# Subfinder — fast passive discovery
subfinder -d your-domain.com -all -o subs.txt

# Resolve and check which ones are live
cat subs.txt | httpx -silent -status-code -title
```

theHarvester — emails, hosts and names

theHarvester aggregates email addresses, host names and employee names from search engines and public sources. Useful to gauge what an attacker could use for phishing or credential-stuffing against your platform.

```
theHarvester -d your-domain.com -b bing,duckduckgo,crtsh -l 500
```

Shodan & Censys — the internet's index

Shodan and Censys continuously scan and index internet-facing services. Querying your own IP or domain shows exactly what banners, ports and certificates the world already sees — often revealing an exposed service you forgot about. This is passive from the target's point of view because the scanning was already done.

EXTERNAL SERVICE INDEX

```
shodan host 203.0.113.10           # services, ports, banners for an IP
shodan search hostname:your-domain.com
# Censys web UI: search your domain or IP, inspect open ports + certs
```

SpiderFoot, recon-ng & Maltego — automation frameworks

When recon grows beyond a handful of commands, these frameworks orchestrate dozens of data sources and visualise the relationships. SpiderFoot runs as a web UI and automates OSINT end-to-end; recon-ng offers a Metasploit-style modular console; Maltego renders entity graphs (domains to IPs to people) that make leaks obvious at a glance.

> **Certificate transparency is a goldmine**
> Every TLS certificate issued for your domain is logged publicly. Searching crt.sh (crt.sh/?q=%25.your-domain.com) instantly reveals subdomains you may have forgotten — including internal-sounding ones that should never have had a public certificate. It costs nothing and touches nothing on your servers.

4 Network discovery & port scanning

This is the highest-value phase for self-hosted infrastructure. The question it answers — *which ports does the world actually reach?* — directly exposes the most common access flaw: a service bound to a public interface by mistake.

nmap — the reference scanner

nmap maps hosts, ports, service versions and, through its scripting engine (NSE), a wide range of vulnerabilities and misconfigurations. It is the backbone of the scanning phase.

NMAP — DISCOVERY & VERSIONING

```
sudo apt install -y nmap

# Full TCP port sweep + version detection + default scripts
nmap -sV -sC -p- your-domain.com

# Fast scan of the 1000 most common ports
nmap -sV your-domain.com

# UDP scan (slower, but DNS/SNMP/etc. live here)
sudo nmap -sU --top-ports 50 your-domain.com

# Run the vulnerability NSE category
nmap --script vuln your-domain.com

# Verify your API/DB ports are NOT reachable externally
nmap -p 8810,8820,3306,22 your-domain.com
```

Scan from outside, not from the box

Run nmap from a machine off your cloud (your home connection), never from the server itself. You then see exactly what a visitor sees — through your reverse proxy and your cloud firewall — rather than the stack from the inside. A port that answers locally but is filtered externally is correctly protected; the reverse is a finding.

Reading the STATE column is what matters. **open** means a service answered and must be justified; **filtered** means a firewall dropped the probe (the desired state for 8810/8820/3306); **closed** means the port is reachable but nothing listens. The target outcome for a typical web host is only 80 and 443 open, everything else filtered.

EXAMPLE OUTPUT — THE STATE YOU WANT

```
PORT     STATE    SERVICE   VERSION
22/tcp   filtered ssh
80/tcp   open     http      nginx 1.24.0
443/tcp  open     ssl/http  nginx 1.24.0
8810/tcp filtered http                  # NLLB engine — correctly hidden
8820/tcp filtered http                  # MADLAD engine — correctly hidden
3306/tcp filtered mysql                 # database not exposed
```

masscan & RustScan — speed at scale

When you need to sweep large ranges quickly, masscan scans the entire IPv4 port space at high packet rates, and RustScan rapidly finds open ports then hands them to nmap for deep inspection. For a single host these are overkill, but they shine across a fleet.

HIGH-SPEED SCANNING

```
# masscan — extremely fast, rate-limited to be polite
sudo masscan 203.0.113.0/24 -p1-65535 --rate 1000 -oG scan.gnmap

# RustScan — fast discovery, pipes into nmap for -sV -sC
rustscan -a your-domain.com -- -sV -sC
```

netcat & hping3 — manual probing

netcat (the 'TCP/IP swiss-army knife') opens raw connections to test a single port, grab a banner, or check connectivity. hping3 crafts custom packets for firewall testing and low-level diagnostics.

MANUAL CONNECTION TESTING

```
nc -vz your-domain.com 443          # is the port open?
nc your-domain.com 80               # then type: GET / HTTP/1.0
sudo hping3 -S -p 443 your-domain.com   # crafted SYN probe
```

5 Service enumeration

Once ports are mapped, enumeration squeezes detail out of each service: exact versions, shared resources, user lists, configuration. The richer the detail, the easier it is to match a service to a known vulnerability.

Service	Tool	What it extracts
SMB / Windows	enum4linux-ng, smbmap	Shares, users, password policy, OS detail
SNMP	snmpwalk, onesixtyone	Device config, interfaces, sometimes credentials
DNS	dnsenum, fierce	Zone transfers, records, host brute-forcing
LDAP	ldapsearch, windapsearch	Directory objects, users, group membership
NFS	showmount	Exported and mountable file shares
HTTP	whatweb, wafw00f	Tech stack fingerprint, WAF detection

For a typical web platform the most relevant of these is HTTP fingerprinting. whatweb identifies the server, framework and libraries in use; wafw00f detects whether a web application firewall sits in front and which one — useful to understand why some probes are being blocked.

HTTP FINGERPRINTING

```
whatweb -a 3 https://your-domain.com    # aggressive fingerprint
wafw00f https://your-domain.com         # detect and identify a WAF
# nmap NSE also enumerates HTTP detail:
nmap -p 443 --script http-enum,http-headers,http-methods your-domain.com
```

Relevance to your stack
SMB, SNMP and LDAP enumeration matter most on Windows/enterprise networks. On a Linux self-hosted box the practical enumeration is HTTP fingerprinting plus confirming that SSH (22), MySQL (3306) and your FastAPI ports never answer enumeration probes from outside.

6 TLS / SSL configuration testing

TLS testing audits how your site negotiates encryption: which protocol versions and cipher suites it accepts, the certificate chain and expiry, and exposure to historical flaws (Heartbleed, ROBOT, BEAST, POODLE). Misconfiguration here weakens every other control.

testssl.sh — thorough, dependency-light

A self-contained bash script that audits the full TLS posture. Cloning the repository (rather than copying the single file) also pulls the data files — cipher lists, CA bundles and vulnerability mappings — it needs for complete results.

TESTSSL.SH

```
git clone --depth 1 https://github.com/testssl/testssl.sh.git
cd testssl.sh
./testssl.sh your-domain.com

./testssl.sh --vulnerable your-domain.com        # known TLS vulns only
./testssl.sh --htmlfile report.html your-domain.com

# Or with zero install, via Docker:
docker run --rm -it drwetter/testssl.sh your-domain.com
```

The output rates each aspect of the configuration. You want modern protocols only (TLS 1.2 and 1.3), no offered weak ciphers, and a clean bill on the vulnerability section. Anything flagged in red — an enabled legacy protocol, a weak cipher, an expiring certificate — is a remediation item for your reverse-proxy config.

EXAMPLE OUTPUT — THE STATE YOU WANT

```
Testing protocols
 TLS 1.2   offered (OK)
 TLS 1.3   offered (OK)
 TLS 1.0   not offered (OK)
 SSLv3     not offered (OK)
Testing vulnerabilities
 Heartbleed   not vulnerable (OK)
 ROBOT        not vulnerable (OK)
```

sslscan & sslyze — fast cipher audits

sslscan gives a quick, colour-coded view of supported protocols and ciphers; sslyze is a Python-based scanner designed for automation and CI, emitting machine-readable JSON. Both complement testssl.sh when you want speed or scriptability.

CIPHER AND PROTOCOL SCANNING

```
sslscan your-domain.com
sslyze --regular your-domain.com
sslyze --json_out tls.json your-domain.com     # for pipelines
```

Qualys SSL Labs for an external grade
ssllabs.com/ssltest gives an A–F grade from outside your network with no installation, and explains every deduction. Aim for an A; it is the quickest way to validate that your reverse-proxy TLS config is sound and that weak protocols (TLS 1.0/1.1, RC2/RC4) are off.

7 Web application proxies & scanners

This is the core of DAST — testing the running application for injection, broken authentication, misconfiguration and the rest of the OWASP Top 10. The category splits into interception proxies (manual, surgical) and automated scanners (broad, fast).

Burp Suite — the manual testing standard

Burp Suite is the industry reference for hands-on web testing. It proxies your browser so you can inspect, modify and replay every request. The Community edition includes the proxy, repeater and decoder; the

Professional edition adds an active scanner and the Intruder for automated fuzzing. Workflow: route your browser through Burp, exercise the app, then pick interesting requests apart in Repeater.

BURP SUITE — MANUAL INTERCEPTION

```
# Burp listens on 127.0.0.1:8080 by default.
# Point your browser proxy there, install Burp's CA cert for HTTPS,
# then: Proxy > intercept, Repeater > replay, Intruder > fuzz params.
# Headless/CI scanning is available in the Professional edition.
```

OWASP ZAP — the open-source workhorse

ZAP is the free, OWASP-maintained counterpart to Burp and the central automated web scanner. It runs two ways: a **passive spider** that crawls and flags issues without sending anything aggressive (always safe to run), and an **active scan** that injects real payloads to confirm vulnerabilities (intrusive — staging only). Its headless mode fits neatly into a scripted pipeline.

OWASP ZAP — AUTOMATED SCANNING

```
# Baseline (passive) scan via Docker — safe, fast
docker run --rm -t ghcr.io/zaproxy/zaproxy:stable \
  zap-baseline.py -t https://your-domain.com

# Full active scan with an HTML report
docker run --rm -t -v $(pwd):/zap/wrk/:rw ghcr.io/zaproxy/zaproxy:stable \
  zap-full-scan.py -t https://your-domain.com -r report_zap.html

# API-aware scan from an OpenAPI/Swagger definition
docker run --rm -t -v $(pwd):/zap/wrk/:rw ghcr.io/zaproxy/zaproxy:stable \
  zap-api-scan.py -t https://your-domain.com/openapi.json -f openapi
```

A baseline scan summarises findings by risk level with the alert name and the rule that fired. WARN items are worth reviewing; FAIL items (in stricter configurations) should block a release. ZAP groups identical alerts, so a single line can represent many affected URLs.

EXAMPLE OUTPUT — BASELINE SCAN SUMMARY

```
WARN-NEW: Content Security Policy (CSP) Header Not Set [10038]
WARN-NEW: Missing Anti-clickjacking Header [10020]
WARN-NEW: Server Leaks Version Information via Server Header [10036]
PASS: Vulnerable JS Library [10003]
FAIL-NEW: 0   WARN-NEW: 3   PASS: 48
```

Active scans can write to your database

A full/active scan submits forms and calls endpoints with crafted data — it can create, modify or delete records and can cause load or downtime. Run it against a staging copy or after a verified backup, never blindly against live production. This applies to Burp's active scanner, ZAP's full scan, and sqlmap alike.

Nikto — fast server-level sweep

Nikto checks for thousands of known-risky files, outdated server software, default credentials and missing headers. It is noisy and prone to false positives, so treat its output as leads to verify rather than confirmed findings — but it is a quick first pass.

NIKTO

```
sudo apt install -y nikto
nikto -h https://your-domain.com
nikto -h https://your-domain.com -o nikto.html -Format htm
```

Wapiti, w3af, Arachni, Skipfish — alternative scanners

Several other open-source scanners cover similar ground and can be useful as a second opinion: each engine catches slightly different issues.

- **Wapiti** — black-box scanner that fuzzes for injection, XSS, file disclosure and more; lightweight and command-line friendly.
- **w3af** — extensible Python framework with a large plugin set spanning discovery, audit and exploitation.
- **Arachni** — high-performance Ruby scanner with a strong crawler, good for JavaScript-heavy apps (now community-maintained).
- **Skipfish** — very fast active reconnaissance crawler that produces an interactive site map and issue report.

8 Content & directory discovery

Forgotten endpoints — old admin panels, backup files, .git directories, debug routes — are a frequent access flaw. Content-discovery tools brute-force paths and parameters against a wordlist to surface what is not linked anywhere.

ffuf, feroxbuster & gobuster — the fast trio

ffuf is an extremely fast, flexible fuzzer (paths, parameters, virtual hosts, headers); feroxbuster recurses automatically into discovered directories; gobuster is the simple, reliable classic. All three rely on a good wordlist — SecLists is the standard collection.

CONTENT DISCOVERY

```
# Wordlists (SecLists)
sudo apt install -y seclists   # or git clone the SecLists repo

# ffuf — directory discovery; FUZZ is the injection point
ffuf -u https://your-domain.com/FUZZ \
  -w /usr/share/seclists/Discovery/Web-Content/raft-medium-directories.txt \
  -mc 200,301,302,403

# feroxbuster — recursive
feroxbuster -u https://your-domain.com -w /usr/share/seclists/Discovery/Web-Content/common.txt

# gobuster — directory mode
gobuster dir -u https://your-domain.com -w common.txt -x php,txt,bak
```

ffuf lists each discovered path with its HTTP status and response size. A 200 on an unexpected path, or a 403 (forbidden but present) on an admin route, is worth manual follow-up — the latter confirms the resource exists even though access is blocked.

EXAMPLE OUTPUT — INVESTIGATE 200S AND 403S

```
admin                   [Status: 403, Size: 162]   # exists, access denied
.git                    [Status: 200, Size: 92]    # exposed VCS!
api                     [Status: 200, Size: 1043]
backup                  [Status: 301, Size: 0]     # redirect, investigate
```

wfuzz, dirsearch & parameter discovery

wfuzz fuzzes any part of a request and is strong for parameter and login testing; dirsearch is a focused path brute-forcer popular for its sensible defaults. To find hidden GET/POST parameters specifically, Arjun is purpose-built.

PARAMETERS & LOGIN FUZZING

```
dirsearch -u https://your-domain.com -e php,html,json
arjun -u https://your-domain.com/api/endpoint     # hidden parameters
wfuzz -c -w wordlist.txt -d 'user=admin&pass=FUZZ' https://your-domain.com/login
```

Check for exposed VCS and backups first

Before a full brute-force, manually request /.git/HEAD, /.env, /config.php.bak and /backup.zip. An exposed .git directory or .env file is a direct, high-severity access flaw that leaks source code and secrets — and it is trivial to test for in seconds.

9 CMS-specific scanners

If any part of your platform runs a content-management system, dedicated scanners know its internals — plugin versions, default paths, known CVEs and user enumeration quirks — far better than a generic tool. Run them only against CMS instances you operate.

CMS	Scanner	Notes
WordPress	WPScan	Enumerates plugins/themes/users, checks a CVE database (token needed)
Joomla	JoomScan	OWASP project; components, versions, known issues
Drupal	droopescan	Modules, themes, version detection
Multiple	CMSeeK / CMSmap	Detects the CMS then runs targeted checks

CMS SCANNING

```
# WordPress: enumerate vulnerable plugins and users
wpscan --url https://your-blog.com --enumerate vp,u --api-token <TOKEN>

# Drupal
droopescan scan drupal -u https://your-site.com

# Detect-then-scan across CMS types
cmseek -u https://your-site.com
```

10 Injection & web exploitation

These tools confirm and exploit the highest-impact web vulnerability classes. They are intrusive by nature — use them only against your own application, ideally on staging, and understand that they can modify data.

sqlmap — SQL injection detection & exploitation

sqlmap automates finding and exploiting SQL injection. Point it at a parameter (or a saved request) and it fingerprints the database, extracts schemas and data, and can even gain a shell where the configuration allows. The first run should stay at a low risk/level to avoid heavy or destructive payloads.

SQLMAP

```
# Test a single parameter
sqlmap -u 'https://your-domain.com/item?id=1' --batch

# Use a request captured from Burp/ZAP (handles auth & POST cleanly)
sqlmap -r request.txt --batch

# Enumerate once injection is confirmed
sqlmap -r request.txt --dbs            # databases
sqlmap -r request.txt -D appdb --tables
sqlmap -r request.txt -D appdb -T users --dump
```

When a parameter is injectable, sqlmap reports the injection type and the back-end DBMS, then lets you drill into schemas and data. A clean run with no injectable parameter — the result you want for a well-built FastAPI app — simply reports that none were found.

EXAMPLE OUTPUT — INJECTABLE PARAMETER FOUND

```
[INFO] testing connection to the target URL
[INFO] GET parameter 'id' is 'MySQL >= 5.0 boolean-based blind' injectable
[INFO] the back-end DBMS is MySQL
available databases [2]:
[*] appdb
[*] information_schema
```

XSS, command & template injection

Beyond SQL, dedicated tools target other injection classes. dalfox and XSStrike find and verify cross-site scripting; commix automates command injection; tplmap and SSTImap target server-side template injection (relevant to any app rendering templates with user input).

OTHER INJECTION CLASSES

```
dalfox url 'https://your-domain.com/search?q=test'    # XSS
commix --url='https://your-domain.com/ping?host=127.0.0.1'  # command injection
python3 tplmap.py -u 'https://your-domain.com/page?name=test'  # SSTI
```

FastAPI relevance

A FastAPI service using parameterised queries (SQLAlchemy, the standard) is largely immune to classic SQL injection — but injection can still appear in raw queries, dynamic ORM filters built from user input, or template rendering. Test the endpoints that build queries or templates from request data, not the framework boilerplate.

11 Vulnerability scanners

Vulnerability scanners cross-reference detected services and versions against large databases of known flaws (CVEs) and misconfigurations, producing a prioritised list. They are the bridge between enumeration and exploitation, and the backbone of continuous assessment.

Nuclei — fast, template-driven scanning

Nuclei has become the go-to open-source scanner. It runs thousands of community-maintained YAML templates — each describing how to detect a specific CVE, exposure or misconfiguration — at high speed, and is trivial to run in CI. It is the easiest high-signal scanner to adopt for a self-hosted site.

NUCLEI

```
# Install and update the template set
nuclei -update-templates

# Scan a target with all templates
nuclei -u https://your-domain.com

# Focus on specific categories or severities
nuclei -u https://your-domain.com -tags cve,exposure -severity high,critical
```

Each line of output names the template that matched, its severity, and the affected URL — concise and immediately actionable. High and critical findings should be triaged first.

EXAMPLE OUTPUT — READ SEVERITY FIRST

```
[tls-version] [info] https://your-domain.com  [TLSv1.3]
[missing-hsts] [low] https://your-domain.com
[git-config] [medium] https://your-domain.com/.git/config   <-- exposed!
[CVE-2023-xxxxx] [high] https://your-domain.com/path
```

OpenVAS / Greenbone — full open-source scanner

OpenVAS (the engine inside the Greenbone Community Edition) is a comprehensive, authenticated-capable scanner comparable to commercial products. It runs network-wide scans against a continuously updated feed of network vulnerability tests and presents results in a web console. Heavier to set up, but free and thorough; the Docker compose deployment is the simplest route.

OPENVAS / GREENBONE

```
# Greenbone Community via Docker compose is the simplest install.
# Once running, browse to the web UI (default https://127.0.0.1:9392),
# create a Target (your IP/host) and a Task, then launch the scan.
```

Commercial scanners — Nessus, Qualys, Rapid7

Commercial platforms add polish, support and breadth. Tenable Nessus is the long-standing reference (a limited free 'Essentials' tier exists); Qualys and Rapid7 Nexpose/InsightVM are enterprise cloud platforms with asset management and compliance reporting. For a single self-hosted site they are usually more than you need — Nuclei plus OpenVAS covers the ground — but they are the standard in larger organisations.

Scanner	Type	Best for
Nuclei	Open-source	Fast, CI-friendly, CVE & exposure checks
OpenVAS / Greenbone	Open-source	Comprehensive authenticated network scans
Nessus	Commercial (free tier)	Reliable all-rounder, broad plugin set
Qualys	Commercial (cloud)	Enterprise asset & compliance management

Scanner	Type	Best for
Rapid7 InsightVM	Commercial (cloud)	Live monitoring, risk prioritisation

12 Exploitation frameworks

Exploitation frameworks turn a known vulnerability into a verified, demonstrable compromise. They are the most powerful — and most dangerous — tools in the kit; in an owner-run assessment they prove that a finding is genuinely reachable, not that you intend to cause harm.

Metasploit Framework — the exploitation platform

Metasploit is the de-facto standard. It bundles thousands of exploits, payloads and auxiliary modules (scanners, brute-forcers, fuzzers) behind a consistent console. The typical flow: search for a module matching a discovered service, set its options, and run it. The auxiliary scanner modules are useful even if you never fire an exploit.

METASPLOIT & EXPLOIT-DB

```
msfconsole
msf> search type:exploit nginx
msf> use auxiliary/scanner/http/http_version
msf> set RHOSTS your-domain.com
msf> run

# searchsploit queries the offline Exploit-DB copy
searchsploit nginx 1.24
```

BeEF & RouterSploit — specialised frameworks

BeEF (Browser Exploitation Framework) demonstrates the impact of client-side flaws by 'hooking' a browser via XSS and showing what an attacker could then do. RouterSploit mirrors Metasploit's model for embedded devices and routers. Both are niche but valuable when the relevant surface is in scope.

> **Exploitation is the staging-only zone**
> Running real exploits or payloads against live production risks downtime, data corruption and lateral effects you did not intend. Confine this phase to a staging replica or a snapshot you can roll back. The goal is a proof-of-concept, captured and documented — then stop. Never escalate beyond what is needed to demonstrate the flaw.

13 Password & authentication testing

Weak or reused credentials remain a leading cause of breach. These tools test authentication strength — online (against a live login) and offline (against captured hashes). Online brute-forcing is intrusive and easily locks accounts, so throttle it and prefer testing your own accounts.

Hydra, Medusa & Patator — online brute-forcing

Hydra is the best-known online password tool, supporting dozens of protocols (SSH, HTTP forms, FTP, databases). Medusa and Patator are parallel alternatives with slightly different strengths. Use them to confirm that your login enforces rate-limiting, lockouts and strong-password policy — not to actually crack accounts.

ONLINE BRUTE-FORCING

```
# SSH login policy test (against your own test account)
hydra -l testuser -P wordlist.txt ssh://your-domain.com

# HTTP POST login form — F marks the failure string
hydra -l admin -P wordlist.txt your-domain.com http-post-form \
  '/login:user=^USER^&pass=^PASS^:Invalid credentials'
```

John the Ripper & Hashcat — offline cracking

When you legitimately hold password hashes (your own dump, recovered during an assessment), John the Ripper and Hashcat test their strength offline. Hashcat is GPU-accelerated and extremely fast; John is flexible and great for mixed hash types. The exercise validates that your hashing (algorithm, salt, work factor) actually resists cracking.

OFFLINE HASH CRACKING

```
# John — autodetect and crack with a wordlist
john --wordlist=rockyou.txt hashes.txt
john --show hashes.txt

# Hashcat — GPU, mode 0 = MD5, 1800 = sha512crypt, 3200 = bcrypt
hashcat -m 3200 -a 0 hashes.txt rockyou.txt
```

Wordlists & rules matter more than speed

Cracking effectiveness comes from good wordlists (rockyou, SecLists) plus mutation rules and target-specific lists built with CeWL from your own site's content. If your bcrypt or Argon2 hashes resist a rockyou run with rules, your storage is doing its job.

14 API, JWT & modern application testing

APIs and token-based authentication are the backbone of modern apps — including a FastAPI backend like yours — and have their own testing tools and failure modes (broken object-level authorisation, excessive data exposure, weak JWT handling). The OWASP API Security Top 10 is the reference checklist.

Postman, ZAP & Burp for API flows

Because FastAPI auto-generates an OpenAPI/Swagger schema, you can feed it directly to a scanner. ZAP's api-scan and Burp both import OpenAPI definitions and exercise every endpoint; Postman is invaluable for crafting and replaying authenticated requests by hand.

API SCANNING

```
# Import your FastAPI schema (served at /openapi.json) into ZAP
docker run --rm -t -v $(pwd):/zap/wrk/:rw ghcr.io/zaproxy/zaproxy:stable \
  zap-api-scan.py -t https://your-domain.com/openapi.json -f openapi

# kiterunner — discover API routes from wordlists
kr scan https://your-domain.com -w routes-large.kite
```

jwt_tool — testing JSON Web Tokens

If your API issues JWTs, jwt_tool inspects and attacks them: the classic flaws are the 'alg:none' bypass, weak HMAC secrets crackable offline, and missing signature verification. Confirming your tokens reject tampering

is a quick, high-value check.

JWT_TOOL

```
python3 jwt_tool.py <token>                 # decode & inspect
python3 jwt_tool.py <token> -X a            # try the alg:none bypass
python3 jwt_tool.py <token> -C -d rockyou.txt   # crack a weak HMAC secret
```

The most common API flaw is authorisation, not injection
Broken Object-Level Authorisation (BOLA/IDOR) — fetching /api/users/124's data while logged in as user 123 — is the top API risk and is not caught by automated scanners. Test it by hand: authenticate as a low-privilege user and try to read or modify another user's objects by changing IDs in the request.

15 Cloud & container security (AWS-focused)

For infrastructure on AWS, the cloud control plane and any containers are part of the attack surface. The most common cloud access flaws are over-permissive security groups, public S3 buckets, and excessive IAM permissions — none of which a web scanner sees. Dedicated tools audit the configuration directly.

Prowler & ScoutSuite — cloud posture auditing

Prowler runs hundreds of checks against AWS (and other clouds) mapped to CIS benchmarks and best practice — security groups, IAM, S3, logging, encryption. ScoutSuite produces a browsable HTML report of the same posture. Both read your account through standard read-only credentials and are the fastest way to find a misconfigured security group or a world-readable bucket.

CLOUD POSTURE (READ-ONLY)

```
# Prowler — full AWS assessment against your account
prowler aws
prowler aws --services ec2 s3 iam      # focus on key services

# ScoutSuite — multi-cloud posture report
scout aws
```

Prowler prints a PASS/FAIL line per check with the affected resource and a severity. The findings to act on first are any FAIL on a security-group ingress rule open to 0.0.0.0/0, a public S3 bucket, or an over-privileged IAM policy.

EXAMPLE OUTPUT — FIX THE FAILS

```
FAIL ec2_securitygroup_allow_ingress_from_internet_to_any_port
     sg-0abc123  ->  port 3306 open to 0.0.0.0/0     # database exposed!
PASS ec2_securitygroup_default_restrict_traffic
FAIL s3_bucket_public_access  ->  my-backups  # publicly readable bucket
PASS iam_root_mfa_enabled
```

Pacu — AWS exploitation framework

Pacu is to AWS what Metasploit is to networks: a modular framework for testing whether a set of credentials can be escalated or abused — privilege escalation paths, persistence, data exfiltration. Use it to validate that a compromised low-privilege key on your account cannot pivot to admin.

Trivy, Grype & Docker Bench — container scanning

If your services ship as containers, image scanners find vulnerable OS packages and libraries baked into the image, plus misconfigurations and embedded secrets. Trivy is the most popular all-rounder (images, filesystems, IaC, secrets); Grype focuses on package vulnerabilities; Docker Bench audits the Docker host against the CIS benchmark.

CONTAINER & IMAGE SCANNING

```
# Trivy — scan a container image for CVEs, secrets, misconfig
trivy image your-registry/your-app:latest

# Trivy — scan a project directory (deps + IaC + secrets)
trivy fs .

# Grype — vulnerability scan of an image or directory
grype your-registry/your-app:latest
```

kube-hunter & kube-bench — Kubernetes

If you run Kubernetes, kube-bench checks the cluster against the CIS Kubernetes benchmark and kube-hunter actively probes for exploitable weaknesses. Not relevant for a plain EC2/systemd deployment, but essential the moment an orchestrator is involved.

> **Cloud posture is your highest-leverage cloud check**
> On AWS, run Prowler first. A single over-permissive security group rule (e.g. 3306 or 8810 open to 0.0.0.0/0) is both the most common cloud finding and the most direct access flaw — and Prowler flags it in one pass, with the exact rule to fix.

16 Static analysis, dependencies & secrets (SAST)

Static analysis examines source code without running it, catching flaws earlier and deeper than any external scan — including vulnerabilities not yet reachable through the UI. For a Python/FastAPI codebase this is some of the highest-value, lowest-risk testing available, and it belongs in CI.

Semgrep & Bandit — code analysis

Bandit is a Python-specific linter that flags common security issues (use of eval, weak crypto, hardcoded passwords, SQL built by string concatenation). Semgrep is a fast, multi-language pattern scanner with a large rule registry, including FastAPI- and framework-aware rules. Run both; they overlap little.

SAST FOR PYTHON

```
# Bandit — Python security linter
pip install bandit
bandit -r ./your_app -ll        # recurse, report medium+ severity

# Semgrep — multi-language, registry rulesets
pip install semgrep
semgrep --config=auto ./your_app
semgrep --config=p/security-audit ./your_app
```

Dependency scanning — pip-audit, Safety, Dependency-Check, Snyk

Most modern vulnerabilities arrive through third-party dependencies. These tools compare your installed packages against vulnerability databases. pip-audit and Safety are Python-native; OWASP

Dependency-Check is language-agnostic; Snyk is a commercial platform with a generous free tier and good CI integration.

DEPENDENCY VULNERABILITY SCANNING

```
pip install pip-audit
pip-audit                          # audit the current environment
pip-audit -r requirements.txt      # audit a requirements file

# Trivy also scans Python deps:
trivy fs --scanners vuln .
```

Gitleaks & TruffleHog — secret detection

Hardcoded API keys, database passwords and tokens committed to a repository are a frequent and severe leak. Gitleaks and TruffleHog scan code and git history for secrets — run them across your whole history, not just the current checkout, because a key committed once lives forever in the log unless rewritten.

SECRET SCANNING

```
gitleaks detect --source . -v          # scan repo + history
trufflehog git file://./your-repo      # deep history secret scan
```

17 Network traffic analysis

Traffic analysis inspects what actually flows over the wire — to confirm encryption, debug a protocol, or spot data leaking in the clear. It is defensive and diagnostic rather than offensive, but indispensable when a finding needs evidence.

Wireshark, tshark & tcpdump

tcpdump captures packets on the command line; tshark is the scriptable Wireshark engine; Wireshark itself provides the deep graphical analysis with protocol dissectors. Together they let you verify, for example, that an internal service really is talking over TLS and not leaking credentials between your reverse proxy and a backend.

PACKET CAPTURE & ANALYSIS

```
# Capture traffic to your API port for inspection
sudo tcpdump -i any -w capture.pcap port 8810

# Quick command-line analysis
sudo tshark -i any -f 'port 443' -Y http

# Then open capture.pcap in Wireshark for full dissection
```

mitmproxy — intercepting HTTPS

mitmproxy is an interactive HTTPS proxy ideal for inspecting and modifying API and mobile traffic on the command line. It complements Burp/ZAP when you want a scriptable, terminal-based interception point — for instance to watch exactly what a client sends to your API.

18 All-in-one security distributions

Rather than installing each tool individually, several Linux distributions ship the entire arsenal pre-configured. Running one in a dedicated virtual machine gives a clean, reproducible testing environment and keeps offensive tooling off your daily machine.

Distribution	Base	Character
Kali Linux	Debian	The reference; broadest tool set, best documented, huge community
Parrot Security OS	Debian	Lighter, privacy-focused, includes dev and anonymity tools
BlackArch	Arch	Enormous tool count (2000+); for advanced Arch users
Pentoo	Gentoo	Hardened, source-based; niche
Commando VM	Windows	Offensive tooling on a Windows base, for AD-heavy targets

You don't need a full distro for one site
For auditing a single self-hosted platform, installing the handful of tools you actually use (nmap, testssl.sh, ZAP, Nuclei, ffuf, sqlmap, Prowler) directly on your machine or in a small VM is simpler than maintaining a full Kali install. Reach for a distribution when you test regularly or across many systems and want everything pre-integrated.

19 Applied to your stack: FastAPI, nginx, AWS

Bringing it together for a self-hosted platform with FastAPI services behind a reverse proxy on AWS, two configuration checks are worth half an application audit on their own.

Bind services to localhost, not 0.0.0.0

Your FastAPI engines should listen on 127.0.0.1 so that only the local reverse proxy can reach them. A service bound to 0.0.0.0 with no firewall is directly exposed to the internet — exactly the access flaw the network phase looks for.

VERIFY THE BIND ADDRESS

```
# Check which interface your services listen on
sudo ss -tlnp | grep -E '8810|8820'

# Desired:  127.0.0.1:8810   (NOT 0.0.0.0:8810)
# In uvicorn:  --host 127.0.0.1   (never --host 0.0.0.0 when proxied)
```

Lock down the AWS security group

The instance security group should allow inbound only what is strictly necessary: 443 (and 80 for redirection), plus 22 restricted to your own IP. Everything else — 8810, 8820, 3306 — stays closed to the world, with communication happening locally on the machine.

Two independent barriers
The AWS security group and the 127.0.0.1 bind are two independent defences. Keep both: if one fails through a config error, the other still protects your translation engines and MySQL. Run Prowler to confirm the security group, and ss -tlnp to confirm the bind.

Harden the reverse proxy and headers

Add security headers at the nginx layer (HSTS, X-Content-Type-Options, X-Frame-Options, a tightened Content-Security-Policy), disable server tokens, and rate-limit authentication endpoints. Validate the result on

securityheaders.com — it is the fastest hardening win for a self-hosted site.

20 A recommended end-to-end workflow

A concrete order of operations, from least to most intrusive. Run the passive and discovery phases against production; reserve active and exploitative phases for staging or a backed-up snapshot.

#	Step	Tools	Phase
1	Map the footprint (passive)	whois, dig, Amass, crt.sh, Shodan	Recon
2	Scan ports from outside AWS	nmap, RustScan	Scan
3	Fingerprint the web stack	whatweb, wafw00f, nmap NSE	Enum
4	Audit TLS	testssl.sh, SSL Labs	Scan
5	Check security headers	securityheaders.com, curl	Scan
6	Fast vuln & exposure scan	Nuclei	Vuln
7	Content & endpoint discovery	ffuf, feroxbuster	Enum
8	Web app scan (passive then active)	ZAP, Burp, Nikto	Vuln
9	Targeted injection/auth tests	sqlmap, dalfox, Hydra, jwt_tool	Exploit
10	Cloud & container posture	Prowler, Trivy	Vuln
11	Source, deps & secrets	Semgrep, Bandit, pip-audit, Gitleaks	SAST
12	Infra config checks	ss -tlnp, security-group review	Enum
13	Remediate, then re-test	(the relevant tool from above)	Report

The guiding principle, once more

On self-hosted infrastructure, start with the network. The typical access flaw is a port or service exposed by mistake, not an exotic application bug. Steps 2 and 10 — external port scan and AWS posture — are your best return on time invested.

21 Reporting & remediation

Findings are only useful once written down, prioritised and fixed. A good report records, for each issue: what it is, where it is, how to reproduce it, the impact, a severity score, and a concrete remediation. CVSS provides a standard 0–10 severity scale; tools like Dradis and Faraday help aggregate findings, but a structured spreadsheet works for a single site.

Prioritise by severity

- **Critical / High** — exposed service or database, RCE, authentication bypass, leaked secrets or .env/.git. Fix immediately, before anything else.
- **Medium** — missing security headers, weak TLS ciphers, verbose error messages, outdated-but-not-exploited components. Schedule promptly.

- **Low / Informational** — banner disclosure, minor misconfigurations, defence-in-depth improvements. Address as part of routine hardening.

Crucially, the loop closes with a **re-test**: after remediation, re-run the tool that found each issue to confirm it is genuinely resolved and that the fix introduced no regression. Security testing is continuous, not a one-off — fold the fast checks (Nuclei, pip-audit, Semgrep, testssl.sh) into CI so regressions surface automatically.

> **Make it repeatable**
> Script the passive and discovery steps so a full baseline takes one command and can run on a schedule. The first assessment is the slowest; every subsequent one should be mostly automated, leaving your attention for the manual, judgement-heavy tests (authorisation logic, business rules) that scanners cannot perform.

22 Building a safe testing lab

Active and exploitative testing should never be rehearsed against live production. A dedicated lab gives you a place to learn tools, validate techniques and stage a copy of your own application before pointing anything intrusive at the real thing. A virtual machine on your workstation is enough.

Virtualisation and isolation

Run your offensive tooling and any vulnerable targets inside isolated VMs (VirtualBox, VMware, or KVM/QEMU on Linux), connected on a host-only or internal network so traffic never leaves the lab. Snapshots let you roll back instantly after a destructive test.

LAB TOPOLOGY

```
# Typical lab layout (host-only network):
#   VM1: Kali / Parrot          -> your attacking machine
#   VM2: staging copy of app    -> the real target, safely cloned
#   VM3: intentionally weak box  -> for practising tools
# Take a snapshot of each VM BEFORE any active or exploit test.
```

Intentionally vulnerable targets

To practise tools without risk, the security community maintains deliberately vulnerable applications. They let you see exactly what a positive finding looks like in each tool before you interpret results on your own platform.

Target	Focus	Run as
OWASP Juice Shop	Modern web/API (OWASP Top 10)	Docker / npm
DVWA	Classic web vulns, adjustable difficulty	Docker / PHP
WebGoat	OWASP teaching app with lessons	Docker / Java
Metasploitable 2/3	Vulnerable Linux host (network & services)	VM image
VulnHub / HTB	Full boot-to-root challenge machines	VM / cloud lab

PRACTICE TARGETS

```
# Spin up OWASP Juice Shop in seconds
docker run --rm -p 3000:3000 bkimminich/juice-shop
# Then point ZAP, ffuf, sqlmap etc. at http://localhost:3000

# DVWA
docker run --rm -p 8080:80 vulnerables/web-dvwa
```

Mirror production, then test the mirror

The safest way to exercise active scans against your real app is to clone it into the lab — same code, a copy of the schema, dummy data — and run sqlmap, ZAP's active scan and Hydra there. You get findings that apply to production without any risk to it.

23 Fuzzing

Fuzzing feeds malformed, unexpected or random input into a program to trigger crashes, memory errors or unhandled states that reveal vulnerabilities. It complements the signature-based scanners by finding unknown (zero-day) bugs rather than catalogued ones. Web parameter fuzzing (ffuf, wfuzz) was covered earlier; this section concerns deeper protocol and application fuzzing.

Coverage-guided fuzzers — AFL++ & libFuzzer

AFL++ and libFuzzer instrument a target binary to maximise code coverage, mutating inputs intelligently to reach deep code paths. They are the standard for fuzzing native code and parsers — relevant if any part of your stack handles untrusted binary input (file uploads, custom protocols, C extensions).

AFL++ — COVERAGE-GUIDED FUZZING

```
# Compile the target with AFL++ instrumentation
afl-cc -o target target.c
# Run the fuzzer with seed inputs
afl-fuzz -i seeds/ -o findings/ -- ./target @@
```

Protocol & application fuzzers

boofuzz is a Python framework for fuzzing network protocols (define the message structure, let it mutate fields and monitor for failures). radamsa is a simple, powerful mutation engine that takes valid samples and produces malformed variants. For APIs, RESTler generates stateful test sequences from an OpenAPI specification — directly applicable to a FastAPI service.

PROTOCOL & API FUZZING

```
# radamsa — mutate a valid sample into malformed variants
radamsa sample.json > fuzzed.json
echo 'GET /api/x' | radamsa

# RESTler consumes your OpenAPI schema to fuzz the API statefully
# (compile from the spec, then run fuzz mode)
```

Where fuzzing pays off for a web platform
For a typical FastAPI app, the highest-value fuzzing targets are file-upload handlers, any custom parser, and the API itself via RESTler. Pydantic validation absorbs a lot of malformed input, so focus fuzzing on the places where data leaves the validated path.

24 Reverse engineering & binary analysis

Reverse engineering inspects compiled software to understand its behaviour, find vulnerabilities, or analyse malware. It is a specialised discipline, less relevant to a pure web platform, but part of the complete expert toolkit — and useful whenever you deploy a closed-source component and need to understand what it really does.

Tool	Role	Licence
Ghidra	Full disassembler + decompiler (NSA-released)	Open-source
radare2 / Cutter	CLI/GUI reverse-engineering framework	Open-source
IDA Pro	Industry-standard disassembler	Commercial
Binary Ninja	Modern disassembler with strong API	Commercial
GDB + pwndbg/GEF	Dynamic debugging, exploit development	Open-source
objdump / strings / file	Quick static triage of a binary	Built-in

Quick static triage

Before reaching for a disassembler, the built-in utilities answer a lot: file identifies the format, strings pulls readable text (often revealing URLs, paths and even secrets), and objdump disassembles. These cost nothing and frequently surface the answer immediately.

BINARY TRIAGE

```
file ./mystery_binary           # format, architecture, linkage
strings -n 8 ./mystery_binary   # readable strings (URLs, paths, keys)
objdump -d ./mystery_binary     # disassembly
checksec --file=./mystery_binary  # which hardening protections are on
```

Start with Ghidra if you are new to RE
Ghidra is free, cross-platform and includes a capable decompiler that turns assembly back into readable C-like pseudocode. For occasional analysis it covers most of what the expensive commercial tools do, with extensive documentation and an active community.

25 Social engineering & phishing simulation

Most real breaches begin with a person, not a port. Phishing-simulation tools let an organisation test and train its own staff safely. Used responsibly — against your own team, with management sign-off, for awareness — they measure how resilient your people are to the techniques attackers actually use.

GoPhish & the Social-Engineer Toolkit

GoPhish is an open-source phishing framework that runs campaigns end-to-end: build a template, send to a controlled recipient list, and track who opened, clicked and submitted credentials — all in a dashboard. The Social-Engineer Toolkit (SET) is a broader framework for crafting attack scenarios. Both must only ever target

people who have consented to be tested as part of a formal awareness programme.

> **Strict consent and scope**
> Phishing simulation against anyone without explicit, documented authorisation is unethical and frequently illegal. Run it only on your own organisation, with written management approval, clear scope, and a debrief that educates rather than punishes. Never harvest or store real credentials beyond the metric of 'submitted: yes/no'.

For a small self-hosted operation, the practical takeaway is less about running campaigns and more about the controls they test: multi-factor authentication everywhere, strong anti-phishing email configuration (SPF, DKIM, DMARC), and staff who know to verify unusual requests out of band.

26 Wireless & mobile testing

Two further domains round out the expert toolkit. They are unlikely to apply to a cloud-hosted web platform, but are core to a complete assessment practice and worth knowing exist.

Wireless — the Aircrack-ng suite & Kismet

Wireless testing assesses Wi-Fi security: the Aircrack-ng suite captures and analyses traffic and tests WPA/WPA2 key strength; Kismet is a wireless detector and sniffer; Wifite automates common attacks. Relevant for physical/office network assessments, not for a remote server.

Mobile — MobSF, Frida & objection

If your platform has a mobile client, mobile testing applies. MobSF (Mobile Security Framework) performs automated static and dynamic analysis of Android/iOS apps; Frida instruments a running app to inspect and modify behaviour; objection builds on Frida for runtime exploration without rooting. These reveal hardcoded secrets, insecure storage and weak certificate pinning in the app itself.

MOBILE APP ANALYSIS

```
# MobSF via Docker — upload an APK/IPA to the web UI for analysis
docker run --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf

# Frida — list processes / hook a running app
frida-ps -U
objection -g com.your.app explore
```

27 Continuous security & CI/CD integration

A one-off assessment ages the moment it finishes: new dependencies, new code and new endpoints arrive constantly. The mature approach folds the fast, non-intrusive checks into your build pipeline so that regressions surface automatically, on every change, before they reach production. This is sometimes called DevSecOps — shifting security left, into development.

Which checks belong in CI

Pick the tools that are fast, deterministic and non-destructive: static analysis, dependency and secret scanning, and lightweight dynamic checks. Heavy or intrusive scans (full ZAP active scan, sqlmap, brute-forcing) belong in a scheduled job against staging, not on every commit.

Stage	Check	Tool	Speed
Pre-commit	Secret detection	Gitleaks	Instant

Stage	Check	Tool	Speed
On push	Static analysis	Semgrep, Bandit	Seconds
On push	Dependency audit	pip-audit, Trivy fs	Seconds
On build	Container image scan	Trivy image, Grype	Seconds
Nightly	Exposure & CVE scan	Nuclei	Minutes
Nightly	TLS posture	testssl.sh	Minutes
Weekly (staging)	Full web scan	ZAP full-scan	Longer
Weekly	Cloud posture	Prowler	Minutes

Example CI step

A minimal pipeline stage that fails the build on a new high-severity dependency or code finding. The exact syntax depends on your CI system, but the shape is universal: install the scanner, run it, let a non-zero exit fail the job.

SECURITY CHECKS IN THE PIPELINE

```
# Example CI stage (GitHub Actions / GitLab CI style)
steps:
  - run: pip install bandit semgrep pip-audit
  - run: bandit -r ./app -ll          # fail on medium+ findings
  - run: semgrep --config=p/security-audit --error ./app
  - run: pip-audit -r requirements.txt # fail on vulnerable deps
  - run: gitleaks detect --source . --no-banner
```

Start with two checks, then grow

Don't try to wire in everything at once. Begin with secret scanning (Gitleaks) and dependency auditing (pip-audit) — they have near-zero false positives and catch the most common real incidents. Add SAST and nightly Nuclei once the team is comfortable with the signal, so security feedback becomes routine rather than a special event.

28 Glossary of key terms

Term	Meaning
CVE	Common Vulnerabilities and Exposures — a unique ID for a known public flaw
CVSS	Common Vulnerability Scoring System — a 0–10 severity score
DAST	Dynamic Application Security Testing — testing the running app from outside
SAST	Static Application Security Testing — analysing source code without running it
SCA	Software Composition Analysis — scanning third-party dependencies for known flaws
OSINT	Open-Source Intelligence — data gathered from public sources
NSE	Nmap Scripting Engine — nmap's library of detection/exploitation scripts
WAF	Web Application Firewall — filters/blocks malicious HTTP traffic
IDOR / BOLA	Insecure Direct Object Reference / Broken Object-Level Authorisation
RCE	Remote Code Execution — running arbitrary code on the target

Term	Meaning
SSTI	Server-Side Template Injection — injecting into a server template engine
XSS	Cross-Site Scripting — injecting script that runs in a victim's browser
PoC	Proof of Concept — minimal demonstration that a vulnerability is real
Grey-box	Testing with partial knowledge of the system (between black- and white-box)

A Master tool reference table

A consolidated index of the tools in this guide, by phase, with their licence and typical intrusiveness against a target.

Tool	Category	Licence	Intrusiveness
whois / dig	Recon	Built-in	None (passive)
Amass / Subfinder	Recon — subdomains	Open-source	None (passive)
theHarvester	Recon — OSINT	Open-source	None (passive)
Shodan / Censys	Recon — index	Freemium	None (passive)
SpiderFoot / recon-ng	Recon — framework	Open-source	Low
nmap	Network scanning	Open-source	Low–Medium
masscan / RustScan	Network scanning	Open-source	Medium
enum4linux / snmpwalk	Enumeration	Open-source	Low
whatweb / wafw00f	Enumeration — HTTP	Open-source	Low
testssl.sh / sslscan	TLS testing	Open-source	None–Low
Burp Suite	Web — proxy	Freemium	Manual / High
OWASP ZAP	Web — scanner	Open-source	Low–High
Nikto	Web — server scan	Open-source	Medium
ffuf / feroxbuster	Content discovery	Open-source	Medium
WPScan / droopescan	CMS scanning	Open-source	Medium
sqlmap	Injection	Open-source	High
dalfox / commix	Injection	Open-source	High
Nuclei	Vuln scanning	Open-source	Low–Medium
OpenVAS / Nessus	Vuln scanning	OSS / Commercial	Medium
Metasploit	Exploitation	Open-source	High
Hydra / Medusa	Auth — online	Open-source	High
John / Hashcat	Auth — offline	Open-source	None (offline)
jwt_tool / kiterunner	API testing	Open-source	Low–Medium
Prowler / ScoutSuite	Cloud posture	Open-source	None (read-only)
Pacu	Cloud exploitation	Open-source	High
Trivy / Grype	Container scanning	Open-source	None (offline)
Semgrep / Bandit	SAST	Open-source	None (static)
pip-audit / Snyk	Dependency scan	OSS / Freemium	None (static)
Gitleaks / TruffleHog	Secret scanning	Open-source	None (static)

Tool	Category	Licence	Intrusiveness
Wireshark / tcpdump	Traffic analysis	Open-source	None (passive)

B Command cheat-sheet

A quick-reference of the most-used commands, ready to adapt.

Reconnaissance

```
whois your-domain.com
dig +short A your-domain.com
subfinder -d your-domain.com -all | httpx -silent -title -status-code
amass enum -passive -d your-domain.com
# crt.sh/?q=%25.your-domain.com   (certificate transparency)
```

Scanning & enumeration

```
nmap -sV -sC -p- your-domain.com
nmap -p 8810,8820,3306,22 your-domain.com    # confirm filtered
whatweb -a 3 https://your-domain.com
wafw00f https://your-domain.com
```

TLS & headers

```
./testssl.sh your-domain.com
sslscan your-domain.com
curl -sI https://your-domain.com | grep -iE 'strict-transport|content-security|x-frame'
```

Web app & discovery

```
nuclei -u https://your-domain.com -severity high,critical
ffuf -u https://your-domain.com/FUZZ -w common.txt -mc 200,301,403
nikto -h https://your-domain.com
docker run --rm -t ghcr.io/zaproxy/zaproxy:stable zap-baseline.py -t https://your-domain.com
```

Targeted tests (staging only)

```
sqlmap -r request.txt --batch --dbs
dalfox url 'https://your-domain.com/search?q=test'
hydra -l testuser -P wordlist.txt ssh://your-domain.com
python3 jwt_tool.py <token> -X a
```

Cloud, containers & source

```
prowler aws --services ec2 s3 iam
trivy image your-registry/your-app:latest
trivy fs .
semgrep --config=auto ./your_app
bandit -r ./your_app -ll
pip-audit -r requirements.txt
gitleaks detect --source . -v
```

Your-stack checks

```
sudo ss -tlnp | grep -E '8810|8820|3306'    # expect 127.0.0.1, not 0.0.0.0
# AWS: inbound 443 + 80 (+ 22 from your IP only); everything else closed
```

This document is a self-assessment reference. It does not replace a professional security audit or penetration test, and every technique described must be used exclusively against infrastructure you own or are explicitly authorised to test. Tool names, options and availability evolve over time — always consult each tool's current documentation before relying on a specific command.