whois → netcraft → dnsdumpster → theHarvester

↓

subfinder → amass → knockpy → fierce

↓

dnsx

↓

nslookup → dig → dnsroute → traceroute

↓

katana → whatweb

↓

naabu → nmap → netcat

↓

httpx → wafw00f → nuclei

🟢 whois – Domain Registration Lookup

🔹 **Purpose:**

`whois` is a command-line tool (and also a protocol) used to query domain registration records. It reveals information about **who owns a domain**, where it's registered, when it was created/updated, and sometimes contact details.

🔑 Key Information Provided:

- **Registrant Details:** Name, email, organization (if not hidden with privacy protection).
- **Registrar Info:** The company that registered the domain (e.g., GoDaddy, Namecheap).
- **Creation & Expiration Dates:** Helps identify new or soon-to-expire domains.
- **Nameservers:** DNS servers hosting the domain.
- **Status Flags:** Such as `clientTransferProhibited` or `serverHold`.

⚡ Typical Usage:

Command-line (Linux/macOS):

```
whois example.com
```

Windows (PowerShell):

```
whois example.com
```

Some Windows systems may need a `whois` package (e.g., from Sysinternals).

🎯 Use Cases in Recon:

- ✅ Identify **registrant email addresses** to find related domains or employees.
- ✅ Spot **newly registered domains** (possible phishing sites).
- ✅ Discover **historical ownership changes** for attack surface mapping.
- ✅ Check if a company uses **privacy protection services**.

🧠 Tips:

- If the domain uses **privacy services** (like WhoisGuard), real owner details might be hidden.
- Use **historical WHOIS** (services like whoisxmlapi or SecurityTrails) to view old records before privacy was enabled.
- Pair with `theHarvester` or `amass` to cross-reference emails/names found in WHOIS.

🟢 Netcraft (Website & Service)

🔹 **Purpose:**
Netcraft is a **web-based reconnaissance platform** that provides detailed information about a domain or IP address, including **hosting history, SSL/TLS details, site technology stack, and network infrastructure**.
It's entirely **passive**, meaning you gather intelligence without touching the target directly.

🔑 Key Information Provided:

- 🌐 **Hosting History:** Previous hosting providers, IP addresses, and infrastructure changes.
- 🔒 **SSL/TLS Certificates:** Issuer, expiration date, and certificate transparency logs.
- ⚙️ **Technology Stack:** Web server type (e.g., Nginx, Apache), operating system, CMS (WordPress, Drupal, etc.).
- 🕵️ **Network Mapping:** IP ranges, ASNs (Autonomous System Numbers), and DNS data.
- 📜 **Site Reports:** Site rank, phishing/malware history, and uptime monitoring.

⚡ Typical Usage:

Since Netcraft is a web platform, you don't need a terminal command.
Simply visit:

```
https://sitereport.netcraft.com/?url=example.com
```

Enter your target domain, and Netcraft will generate a full report.

🎯 Use Cases in Recon:

- ✅ Track **infrastructure changes** over time (e.g., when a company switches hosts).
- ✅ Identify **outdated servers** or operating systems.
- ✅ Find **subdomains or IPs** hidden behind CDNs (like Cloudflare).
- ✅ Check **phishing or blacklist status**.

🧠 Tips:

- Use the **"Hosting History"** feature to discover old IPs and domains that might still be active.
- Cross-reference Netcraft results with `whois` to confirm ownership details.
- Combine with `dnsdumpster` or `amass` to expand subdomain and DNS findings.

🟢 DNSdumpster (Website)

🔹 **Purpose:**
DNSdumpster is a **free, passive DNS recon tool** that maps a target domain's DNS records and subdomains.
It helps visualize an organization's **external attack surface** by showing servers, mail exchanges, and hidden assets.

🔑 Key Information Provided:

- 🌐 **Subdomains:** Lists discovered subdomains and their IPs.
- 🗺️ **DNS Records:** A, MX, TXT, and NS records.
- 🕸️ **Network Mapping:** Provides a downloadable graph of the target's DNS infrastructure.
- 🖼️ **Reverse DNS:** Identifies related domains hosted on the same IP.

⚡ Typical Usage:

No installation required—just visit:

```
https://dnsdumpster.com/
```

Enter the target domain (e.g., `example.com`) and start the search.
It generates:

- A **table of results** (subdomains, records, IPs).
- A **visual network map** of DNS relationships.

🎯 Use Cases in Recon:

- ✅ Identify **hidden subdomains** that may host internal apps, staging servers, or dev environments.
- ✅ Find **mail servers (MX)** to test for email spoofing or phishing vectors.
- ✅ Detect **TXT/SPF/DMARC records** for security misconfigurations.
- ✅ Export results to CSV for integration with tools like `dnsx` or `amass`.

🧠 Tips:

- Combine results with `subfinder` and `amass` for deeper subdomain enumeration.
- Use discovered IPs for further scanning with `nmap` or `naabu`.
- Cross-check MX/TXT records with **SPF/DKIM validation** to find email security gaps.

🟢 theHarvester

🔹 **Purpose:**
`theHarvester` is a **passive information-gathering tool** used to collect **emails, subdomains, IPs, employee names, and URLs** from publicly available sources.
It is extremely useful in the **early stages of reconnaissance** because it doesn't directly interact with the target's infrastructure.

🔑 Key Information Collected:

- 📧 **Emails:** Employee and admin emails leaked on search engines, PGP servers, or GitHub.
- 🌐 **Subdomains:** Discovered through search engines (Google, Bing, Yahoo, Baidu, etc.).
- 🏷️ **Hosts/IPs:** Associated IP addresses of discovered subdomains.
- 🔗 **Links/URLs:** Publicly indexed URLs that reveal hidden directories or files.

⚡ Typical Usage:

Command-line example:

```
theharvester -d example.com -b google
```

Options:

- `-d` → Domain name to search (e.g., `example.com`).
- `-b` → Data source (Google, Bing, DuckDuckGo, Yahoo, LinkedIn, GitHub, etc.).
- `-l` → Limit number of results.
- `-f` → Export results to an HTML/XML file.

Example with multiple sources:

```
theharvester -d example.com -b google,bing,linkedin -l 500
```

🎯 Use Cases in Recon:

- ✅ Harvest **corporate emails** to use in phishing or credential stuffing campaigns (for testing in legal pentests).
- ✅ Find **hidden subdomains** exposed through search engines but not publicly linked.
- ✅ Discover **related infrastructure** (e.g., dev/staging servers) by checking URLs.
- ✅ Gather **OSINT data** before moving to active enumeration.

🧠 Tips:

- Combine with `whois` or **Netcraft** to verify email ownership.
- Use harvested emails in **breach databases** (like HaveIBeenPwned) to check for leaked passwords.
- Pair with `subfinder` or `amass` for deeper subdomain discovery.

🟢 subfinder

🔹 **Purpose:**
`subfinder` is a **fast and reliable subdomain discovery tool** developed by ProjectDiscovery.
It focuses on **passive enumeration**, meaning it collects subdomains from **public sources and APIs** rather than brute-forcing, making it **stealthy** and ideal for the early recon phase.

🔑 Key Features:

- 🌐 **Passive Sources:** Gathers data from certificate transparency logs, DNS databases, search engines, and APIs (like VirusTotal, SecurityTrails, Shodan).
- ⚡ **High Speed:** Optimized for quick discovery of thousands of subdomains.
- 🔒 **Stealthy:** No direct interaction with the target's servers.
- 🔁 **Integration:** Works seamlessly with tools like `dnsx`, `httpx`, and `nuclei` for automation.

⚡ Typical Usage:

Basic scan:

```
subfinder -d example.com
```

Save output to a file:

```
subfinder -d example.com -o subdomains.txt
```

Use multiple threads for speed:

```
subfinder -d example.com -t 50
```

Use all available passive sources:

```
subfinder -d example.com -all
```

🎯 Use Cases in Recon:

- ✅ Discover **live subdomains** (e.g., `api.example.com`, `dev.example.com`).
- ✅ Feed results into `dnsx` for DNS resolution and IP verification.
- ✅ Identify forgotten or **shadow infrastructure** (e.g., staging/test servers).
- ✅ Chain with `nuclei` for automated vulnerability scanning of discovered hosts.

🧠 Tips:

- Always use the `-silent` flag when integrating into automation pipelines:

  ```
  subfinder -d example.com -silent | dnsx -silent
  ```

- Combine with `amass` for maximum coverage (subfinder is faster, amass is deeper).
- Use an API key configuration (`~/.config/subfinder/config.yaml`) to unlock more data sources.

🟢 Amass

🔹 **Purpose:**
`Amass` is one of the **most powerful subdomain enumeration and network mapping tools** available.
It performs **passive, active, and brute-force discovery** to uncover hidden subdomains, IPs, ASN (Autonomous System Numbers), and related domains.

🔑 Key Features:

- 🌐 **Multiple Modes:**
  - **Passive** → Uses public sources (like SSL/TLS certs, APIs, search engines).
  - **Active** → Performs DNS resolution, zone transfers (if possible).
  - **Brute-force** → Uses wordlists to guess subdomains.
  - **Reverse DNS** → Finds domains hosted on discovered IPs.
- 📊 **Network Mapping:** Identifies ASN, CIDR ranges, and infrastructure relationships.
- 🔁 **Recursive Discovery:** Finds subdomains of subdomains.
- ⚡ **Integration:** Works with other ProjectDiscovery tools (`dnsx`, `httpx`, `nuclei`).

⚡ Typical Usage:

Passive scan (stealthy):

```
amass enum -passive -d example.com
```

Active scan with brute-force:

```
amass enum -brute -d example.com
```

Use a wordlist for deeper enumeration:

```
amass enum -brute -d example.com -w /path/to/wordlist.txt
```

Save results to a file:

```
amass enum -d example.com -o amass_results.txt
```

🎯 Use Cases in Recon:

- ✅ Find **deep and hard-to-detect subdomains**.
- ✅ Identify **entire network ranges** by mapping ASN and CIDRs.
- ✅ Correlate **related domains** belonging to the same organization.
- ✅ Feed discovered IPs to `nmap` or `naabu` for further service scanning.

🧠 Tips:

- Combine with `subfinder` for speed and coverage:

  ```
  subfinder -d example.com -silent | amass enum -active -d example.com
  ```

- Use `-config` with API keys to access premium data sources (VirusTotal, SecurityTrails, etc.).
- Run Amass in **passive mode first** to avoid detection, then use active scans if allowed.

🟢 Knockpy

🔹 **Purpose:**
`Knockpy` is a **subdomain enumeration tool** that uses **dictionary-based brute-forcing** to discover hidden or unlisted subdomains.
Unlike passive tools (like `subfinder` or `amass` passive mode), Knockpy **actively queries DNS records** to check which subdomains exist.

🔑 Key Features:

- 📜 **Wordlist-based Enumeration:** Uses a list of potential subdomains (e.g., `admin`, `api`, `dev`) to guess valid ones.
- 🌐 **DNS Resolution:** Verifies subdomains in real time to ensure accuracy.
- ⚡ **Custom Wordlists:** Allows you to supply your own dictionary for targeted brute-force attacks.
- 🗂️ **Output Files:** Stores discovered subdomains for later use.

⚡ Typical Usage:

Basic scan:

```
knockpy example.com
```

Use a custom wordlist:

```
knockpy example.com -w /path/to/wordlist.txt
```

Save output to a file:

```
knockpy example.com -o knockpy_results.txt
```

🎯 Use Cases in Recon:

- ✅ Discover **hidden subdomains** missed by passive tools.
- ✅ Identify **internal or test environments** (e.g., `dev.example.com`, `staging.example.com`).
- ✅ Feed results into tools like `dnsx` to validate and resolve IPs.
- ✅ Combine with `nuclei` for vulnerability testing of newly found subdomains.

🧠 Tips:

- Start with **small wordlists** for quick results, then move to **large wordlists** (like SecLists) for deeper discovery.
- Use `dnsx` after Knockpy to check which discovered subdomains are alive:

  ```
  knockpy example.com -w list.txt | dnsx -silent
  ```

- Brute-forcing may generate **a lot of DNS requests**, so use responsibly to avoid rate limiting or detection.

🟢 Fierce

🔹 **Purpose:**
`Fierce` is a **DNS reconnaissance tool** used to discover subdomains, misconfigured DNS records, and internal network information.
It's designed to identify **non-contiguous IP space** (IPs owned by the same company but not listed together) and hidden hosts.

🔑 Key Features:

- 🌐 **Subdomain Discovery:** Attempts zone transfers and brute-forces common subdomains.
- 🧩 **DNS Sweeping:** Detects wildcard DNS records and unusual configurations.
- 📡 **Network Mapping:** Helps locate different IP ranges owned by the target.
- 🕵️ **Internal Network Exposure:** Finds RFC1918 (private) IP leaks if misconfigurations exist.

⚡ Typical Usage:

Basic scan:

```
fierce --domain example.com
```

Custom DNS server:

```
fierce --domain example.com --dns-servers ns1.example.com
```

Use a specific wordlist:

```
fierce --domain example.com --subdomain-file subdomains.txt
```

🎯 Use Cases in Recon:

- ✅ Identify **misconfigured DNS servers** that allow zone transfers (AXFR).
- ✅ Locate **hidden subdomains** missed by purely passive tools.
- ✅ Map **IP ranges** belonging to the target organization.
- ✅ Check for **wildcard DNS records** that can hide legitimate services.

🧠 Tips:

- Run **passive tools first** (`subfinder`, `amass`) to avoid unnecessary noise, then use Fierce for deeper active probing.
- If a zone transfer is successful, you may discover **all subdomains in a single query**—a big win for recon.
- Combine results with `dnsx` for DNS resolution and `nmap` for port/service scanning.

🟢 **dnsx** – DNS Subdomain Resolver & Validator

Purpose:
dnsx resolves discovered subdomains to IP addresses and filters out inactive hosts. It's primarily used to validate subdomains gathered from tools like **subfinder**, **amass**, **Knockpy**, or **Fierce** before moving to scanning or web probing.

Key Features:

- Resolves domains to IPs.
- Checks which subdomains are alive (responsive).
- Supports reverse DNS lookups.
- Fast and can process large lists.

Usage Examples:

```
# Resolve subdomains from a file and show IPs
dnsx -l subdomains.txt -resp

# Check which hosts are alive silently
dnsx -l subdomains.txt -silent

# Resolve A records and perform reverse DNS lookup
dnsx -l subdomains.txt -a -ptr
```

Use Cases:
✅ Validate discovered subdomains.
✅ Filter live hosts for scanning with **nmap**, **naabu**, or **httpx**.
✅ Integrate into automation pipelines for recon workflows.

Tips:

- Combine with `-o` to save results to a file for further processing:

```
dnsx -l subdomains.txt -o live_hosts.txt
```

- Chain with **httpx** to check live web services:

```
dnsx -l subdomains.txt | httpx -silent
```

🟢 **nslookup** – Basic DNS Lookup Tool

Purpose:
`nslookup` is a command-line utility used to query DNS servers and retrieve DNS records of a domain. It's simple but effective for quickly verifying A, MX, CNAME, and TXT records.

Key Features:

- Query specific DNS record types (A, MX, NS, TXT, CNAME).
- Specify which DNS server to use for queries.
- Works on almost all operating systems.

Usage Examples:

```
# Basic query for A record
nslookup example.com

# Query MX records (mail servers)
nslookup -type=MX example.com

# Query using a specific DNS server
nslookup example.com 8.8.8.8
```

Use Cases:
✅ Verify if a domain resolves correctly.
✅ Check mail servers and DNS configurations.
✅ Troubleshoot DNS resolution issues.
✅ Validate subdomains discovered in recon.

Tips:

- Use different DNS servers (Google 8.8.8.8, Cloudflare 1.1.1.1) to check propagation.
- Pair with **dig** for more advanced or detailed DNS queries.

🟢 **dig** – Advanced DNS Query Tool

Purpose:
`dig` (Domain Information Groper) is a flexible command-line tool for querying DNS records. It's more advanced than `nslookup` and widely used for detailed DNS troubleshooting and reconnaissance.

Key Features:

- Query any DNS record type (A, MX, TXT, NS, CNAME, SOA, etc.).
- Supports batch queries and custom DNS servers.
- Displays detailed query responses including TTL, record type, and authoritative servers.
- Can be used to check propagation or diagnose DNS issues.

Usage Examples:

```
# Query all DNS records for a domain
dig example.com ANY

# Query a specific record type (e.g., MX for mail servers)
dig example.com MX

# Use a specific DNS server
dig @8.8.8.8 example.com A

# Reverse lookup for an IP address
dig -x 192.0.2.1
```

Use Cases:
✅ Verify DNS configurations and propagation.
✅ Validate subdomains discovered via recon tools.
✅ Troubleshoot DNS issues in networks.
✅ Collect DNS data for mapping the target's infrastructure.

Tips:

- Combine with **dnsx** to cross-verify which discovered subdomains are live.
- Use `+short` to get a concise output (just IP addresses or values).

```
dig example.com A +short
```

🟢 **dnsroute** – DNS Route/Propagation Mapper

Purpose:
`dnsroute` is used to visualize and analyze the path DNS queries take from your location to the target domain. It helps identify how DNS records propagate across different servers and networks.

Key Features:

- Maps DNS resolution path.
- Detects intermediate DNS servers between your system and the target.
- Useful for spotting misconfigurations or anomalies in DNS routing.

Usage Examples:

```
# Basic usage to trace DNS route for a domain
dnsroute example.com

# Specify a DNS server to start the trace from
dnsroute -server 8.8.8.8 example.com
```

Use Cases:
✅ Understand which DNS servers handle queries for the domain.
✅ Detect potential propagation delays or inconsistencies.
✅ Identify weak or misconfigured DNS infrastructure.
✅ Support deeper reconnaissance by combining with traceroute and dig results.

Tips:

- Run multiple traces from different locations to compare DNS propagation globally.
- Combine with **dig** or **nslookup** to validate the final resolved IP addresses.

🟢 **traceroute** – Network Path Mapping Tool

Purpose:
`traceroute` is used to map the route that packets take from your system to a target host. It shows all intermediate routers and network hops, helping you understand the network path and potential points of failure or filtering.

Key Features:

- Lists all hops between your machine and the target.
- Shows response times for each hop.
- Can help identify firewalls, bottlenecks, or unusual routing.

- Works on most operating systems (Linux/macOS) – Windows uses `tracert`.

Usage Examples:

```
# Basic traceroute to a domain
traceroute example.com

# Use ICMP packets (common on some networks)
traceroute -I example.com

# Set maximum hops
traceroute -m 30 example.com
```

Use Cases:
✅ Map network infrastructure to the target.
✅ Identify firewalls, proxies, or network filters.
✅ Complement DNS mapping for a full picture of network layout.
✅ Verify connectivity and latency to subdomains or servers discovered earlier.

Tips:

- On Windows, use `tracert example.com` instead.
- Combine with **dnsroute** to correlate DNS propagation with network path.
- Useful for planning further scanning (e.g., nmap or naabu) based on reachable hosts.

🟢 **Katana** – Web Discovery & Crawling Tool

Purpose:
Katana is a fast web crawler designed to discover hidden URLs, endpoints, and files on a target website. It's useful for mapping the web surface before scanning or vulnerability testing.

Key Features:

- Discovers hidden paths, directories, and JavaScript files.
- Supports multi-threading for high-speed crawling.
- Can output results in multiple formats for integration with other tools.
- Works well with automation pipelines (dnsx → katana → httpx → nuclei).

Usage Examples:

```
# Crawl a target website
katana -u https://example.com

# Save discovered URLs to a file
katana -u https://example.com -o urls.txt

# Multi-threaded crawling for speed
katana -u https://example.com -t 50
```

Use Cases:
✅ Discover hidden web pages, directories, and JS endpoints.
✅ Prepare a list of live URLs for vulnerability scanning with nuclei.
✅ Map the structure of web applications before testing.
✅ Feed discovered URLs into tools like httpx to check for live hosts.

Tips:

- Combine with **dnsx** to target only live subdomains.
- Use output files for chaining with **httpx** or **nuclei**.
- Avoid excessive crawling to prevent triggering WAFs or rate limits.

🟢 **WhatWeb** – Website Fingerprinting Tool

Purpose:
WhatWeb identifies the technologies used by a website, including CMS platforms, web servers, frameworks, analytics tools, and plugins. This helps in understanding the target's tech stack before further testing.

Key Features:

- Detects CMS (WordPress, Drupal, Joomla, etc.) and server software.
- Identifies JavaScript libraries, analytics tools, and web frameworks.
- Can detect CDN and WAF usage.
- Supports custom plugins and verbose output for detailed analysis.

Usage Examples:

```
# Basic scan of a website
whatweb https://example.com

# Verbose output for detailed info
whatweb -v https://example.com

# Scan multiple URLs from a file
whatweb -i urls.txt
```

Use Cases:
✅ Identify CMS and web server versions for potential vulnerabilities.
✅ Detect security layers like WAFs or CDNs.
✅ Gather intelligence for targeted web vulnerability testing.
✅ Combine with katana results to fingerprint discovered endpoints.

Tips:

- Use in passive mode first to avoid triggering WAF alerts.
- Cross-reference results with httpx to verify which hosts are live.
- Output results to a file for further automated analysis.

🟢 **Naabu** – Fast Port Scanning Tool

Purpose:
Naabu is a high-speed port scanner used to identify open TCP/UDP ports on a target host. It's designed for speed and can be integrated into automated recon workflows.

Key Features:

- Scans TCP and UDP ports quickly.
- Supports single IP, ranges, or subdomains.
- Can output results in multiple formats for automation.

- Integrates seamlessly with tools like **nmap**, **httpx**, and **nuclei**.

Usage Examples:

```
# Scan a single host for top 1000 TCP ports
naabu -host example.com

# Scan multiple hosts from a file
naabu -l hosts.txt

# Save results to a file
naabu -host example.com -o open_ports.txt
```

Use Cases:
✅ Identify which ports are open on target hosts.
✅ Feed open ports into nmap for detailed service and version detection.
✅ Combine with httpx for live web service checking.

Tips:

- Use `-top-ports 1000` for a quick scan, or `-p` to specify custom ports.
- Integrate with automation pipelines:

```
naabu -list hosts.txt -silent | nmap -sV -p- -oN nmap_results.txt
```

🟢 **Nmap** – Advanced Network & Port Scanner

Purpose:
Nmap (Network Mapper) is a powerful and flexible tool used for detailed network discovery, port scanning, service enumeration, and OS fingerprinting. It's essential for both recon and penetration testing.

Key Features:

- **Port Scanning:** TCP, UDP, and custom ports.
- **Service Detection:** Identifies running services and versions (`-sV`).
- **OS Fingerprinting:** Detects the operating system and device type (`-O`).
- **Scriptable Engine (NSE):** Automates vulnerability scanning, brute-forcing, and advanced checks.
- **Flexible Output:** Normal, XML, grepable, and JSON formats.

Best Usage Examples:

1️⃣ Basic Host Discovery & Port Scan

```
nmap example.com
```

2️⃣ Service Version Detection

```
nmap -sV example.com
```

3️⃣ Aggressive Scan (OS, Service, Scripts)

```
nmap -A example.com
```

4️⃣ Scan Specific Ports or Ranges

```
nmap -p 22,80,443 example.com       # specific ports
nmap -p 1-65535 example.com          # all ports
```

5️⃣ Scan Multiple Hosts from a File

```
nmap -iL hosts.txt -sV -O -oN nmap_results.txt
```

6️⃣ Run Vulnerability Scripts (NSE)

```
nmap --script vuln example.com
```

Use Cases:
✅ Detect open ports and running services.
✅ Identify operating systems and devices.
✅ Find vulnerabilities using NSE scripts.
✅ Map network infrastructure and security posture.

Tips for Best Results:

- Start with `-sS` (TCP SYN scan) for stealth and speed.
- Combine `-Pn` if the host ignores pings.
- Use NSE scripts for reconnaissance automation:

```
nmap -sV --script=http-title,http-enum example.com
```

- Save outputs for automation and chaining with other tools like **httpx** or **nuclei**.

🟢 **Netcat (nc)** – Swiss Army Knife for Network Interaction

Purpose:
Netcat is a versatile networking utility used for reading from and writing to network connections. In recon and pentesting, it's commonly used for manual port scanning, banner grabbing, and testing open ports or services.

Key Features:

- Connects to TCP or UDP ports.
- Sends/receives data manually for testing services.
- Can be used as a lightweight backdoor or listener in advanced pentesting.
- Useful for banner grabbing to identify service versions.

Usage Examples:

```
# Connect to a specific port
nc example.com 80

# Grab a banner from a service (e.g., HTTP)
echo -e "HEAD / HTTP/1.0\n\n" | nc example.com 80

# Scan a range of ports
nc -zv example.com 20-100

# Listen on a local port (for testing or reverse shells)
nc -lvp 4444
```

Use Cases:
✅ Manually check services running on open ports.
✅ Grab banners to identify software versions.
✅ Test connectivity to specific hosts/ports before automated scans.
✅ Useful for debugging network issues during recon.

Tips:

- Use `-z` for scanning without sending data.
- Combine with Nmap results to manually verify services.
- Great for learning and manual verification of automated scan results.

🟢 **httpx** – Web Probing & Live Host Detection

Purpose:
httpx is a fast and flexible tool to check which hosts are alive, determine HTTP/HTTPS status, and collect basic web information. It's often used after subdomain enumeration and port scanning to filter live web servers for further testing.

Key Features:

- Checks HTTP/HTTPS response status for hosts.
- Retrieves titles, status codes, and server headers.
- Detects technologies (basic fingerprinting) and SSL info.
- Supports multiple input sources and outputs in various formats.
- Integrates easily with other tools like **nuclei** or **katana**.

Usage Examples:

```
# Check a single host
httpx -u https://example.com

# Check multiple hosts from a file
httpx -l hosts.txt

# Retrieve titles and status codes
httpx -l hosts.txt -title -status-code

# Use multiple threads for speed
httpx -l hosts.txt -threads 50
```

```
# Output results to a file
httpx -l hosts.txt -o live_hosts.txt
```

Use Cases:
✅ Filter live web hosts from a list of discovered subdomains.
✅ Collect HTTP response headers for reconnaissance.
✅ Identify hosts for vulnerability scanning with **nuclei**.
✅ Quickly map the web surface of a target.

Tips:

- Combine with **dnsx** results to only probe alive subdomains.
- Use `-threads` for faster processing of large lists.
- Output to files for chaining with automated scanning pipelines.

🟢 **Wafw00f** – Web Application Firewall (WAF) Detection

Purpose:
Wafw00f identifies whether a target website is protected by a Web Application Firewall (WAF) or security filtering mechanism. This helps in understanding potential restrictions before performing vulnerability scans or attacks.

Key Features:

- Detects popular WAFs (Cloudflare, Akamai, F5, Sucuri, etc.).
- Provides WAF vendor information and detection confidence.
- Works passively without triggering alarms in most cases.
- Useful for planning bypass strategies or assessing security posture.

Usage Examples:

```
# Scan a single website
wafw00f https://example.com

# Scan multiple hosts from a file
wafw00f -i hosts.txt
```

Use Cases:
✅ Identify if a WAF is active on a web application.
✅ Determine the type of WAF for vulnerability testing or bypass techniques.
✅ Plan automated scans with tools like **nuclei** by knowing which hosts have protection.
✅ Support security assessments and pentesting reports.

Tips:

- Use after confirming live web hosts with **httpx**.
- Combine with **WhatWeb** to correlate WAF presence with other web technologies.
- Detection may not always be accurate; manual verification can be useful.

🟢 **Nuclei** – Automated Vulnerability Scanning

Purpose:
Nuclei is a template-based vulnerability scanner that automates the detection of security issues in web applications, APIs, and network services. It uses pre-defined templates to check for common vulnerabilities, misconfigurations, and CVEs.

Key Features:

- Template-driven: Supports thousands of ready-made templates for vulnerabilities like XSS, SQLi, SSRF, and more.
- Fast and scalable for large targets or multiple hosts.
- Can scan both web applications and network services.
- Integrates seamlessly with tools like **dnsx**, **httpx**, and **katana** for automated workflows.

Usage Examples:

```
# Scan a single URL with default templates
nuclei -u https://example.com

# Scan multiple hosts from a file
nuclei -l live_hosts.txt

# Use specific template
nuclei -u https://example.com -t cves/

# Output results to a file
nuclei -l live_hosts.txt -o vulnerabilities.txt
```

Use Cases:
✅ Automated detection of web vulnerabilities and misconfigurations.
✅ Scan multiple subdomains or URLs efficiently.
✅ Integrate into recon pipelines after subdomain enumeration and live host filtering.
✅ Quick identification of exploitable vulnerabilities for pentesting or security assessments.

Tips:

- Regularly update templates for the latest vulnerabilities:

```
nuclei -update-templates
```

- Combine with **httpx** results to scan only live hosts.
- Use custom templates for organization-specific scanning needs.

✅ **At this point, your full recon workflow is complete**:

Flow Summary:
`whois → Netcraft → DNSdumpster → theHarvester → subfinder → amass → Knockpy → Fierce → dnsx → nslookup → dig → dnsroute → traceroute → katana → WhatWeb → naabu → Nmap → Netcat → httpx → wafw00f → nuclei`