

Part 1.1 – Subfinder (Fast Subdomain Discovery)

🎯 Goal

Find as many **live subdomains** of a target domain as quickly as possible.

✅ Why to Use It

- Collects **subdomains from 30+ online sources** (certificates, DNS records, APIs).
 - Extremely **fast & lightweight**, great for daily scans.
 - Ideal first step to map your target's attack surface.
-

⌚ When to Use It

- **First step** of recon on any new target.
 - Whenever you need a **fresh list of subdomains** (targets change over time).
 - Before running scanners (Nmap, Nuclei) to focus only on valid domains.
-

⚡ Best Tip (Basic Command)

```
subfinder -d target.com -all -recursive -silent -o subdomains.txt
```

- `-all` → Use all available sources.
 - `-recursive` → Find subs of subs (deep recon).
 - `-silent` → Clean output (no banners).
 - `-o` → Save results to a file.
-

🔑 Filtering for the Best Results

After collecting, immediately check **which subdomains are live**:

```
subfinder -d target.com -silent | httpx -silent -status-code -title -tech-detect  
-o live_subs.txt
```

- `httpx` filters **only active hosts** and shows:
 - **Status Code** (e.g., 200, 403, 500)
 - **Page Title** (e.g., "Admin Login")
 - **Technology** (e.g., WordPress, Nginx)

Pro Tip:

Focus on:

- **403/401** (Forbidden/Unauthorized) → Often hide juicy admin panels.
 - **Unique tech stacks** (e.g., outdated WordPress, custom apps).
-

Workflow Example

1. Initial Recon

```
subfinder -d example.com -all -recursive -silent | tee subs.txt
```

2. Check Which Are Alive

```
cat subs.txt | httpx -status-code -title -tech-detect -o live.txt
```

3. Prioritize Interesting Hosts

```
grep -E "200|403" live.txt > priority_targets.txt
```

This gives you a **clean list of live, high-value subdomains** ready for deeper scanning.

Part 1.2 – Amass (Deep Subdomain & Asset Enumeration)

Goal

Discover **hidden subdomains and related assets** using a combination of **OSINT, brute force, and active probing**.

Why to Use It

- Goes **deeper than Subfinder** by combining:
 - DNS enumeration
 - Certificate transparency
 - Web archives
 - Brute forcing with custom wordlists
 - Great for finding **internal, old, or forgotten** assets.
-

When to Use It

- After your initial **Subfinder** scan to expand scope.
 - When hunting **large organizations** with many subdomains.
 - During **intel gathering** to discover **related domains** owned by the same company.
-

⚡ Best Tip (Basic Command)

```
amass enum -d target.com -passive -o amass_passive.txt
```

- `enum` → Run the enumeration mode.
- `-passive` → Use only passive sources (safe, no active probing).
- `-o` → Save results to a file.

➡ Active + Brute Force Combo (more powerful but louder)

```
amass enum -d target.com -brute -w wordlist.txt -o amass_brute.txt
```

- `-brute` → Brute force with a wordlist.
- `-w` → Specify a custom subdomain wordlist.

🔑 Finding Related Domains (Intel Mode)

To discover **other domains owned by the same org**:

```
amass intel -org "Target Company" -whois -o related_domains.txt
```

- `-org` → Search by organization name.
- `-whois` → Include WHOIS records for pivoting.

💡 Use Case:

This can reveal **sister brands, dev domains, or staging environments**.

💎 Filtering for High-Value Results

Amass output can be huge. Use these commands to isolate the *best* findings:

1. Check Live Hosts Only

```
cat amass_passive.txt | httpx -silent -status-code -title -tech-detect -o amass_live.txt
```

1. Highlight Admin/Interesting Assets

```
grep -iE 'admin|login|api|dev|staging' amass_live.txt
```

1. Find New Assets vs Old Results

```
diff <(sort old_amass.txt) <(sort amass_passive.txt)
```

Shows only **new subdomains** since your last scan.

Workflow Example

```
# 1 Deep enumeration  
amass enum -d target.com -brute -w ~/wordLists/big.txt -o amass_all.txt  
  
# 2 Live check & filter  
cat amass_all.txt | httpx -silent -status-code -title -tech-detect | tee live_amass.txt  
  
# 3 Prioritize new/hidden  
grep -iE '403|401|admin|internal|staging' live_amass.txt > priority.txt
```

This workflow gives a **refined list of unique, high-value subdomains** that Subfinder might have missed—perfect for next-stage testing.

Part 1.3 – Chaos Dataset (Pre-Collected Bug Bounty Assets) **(optional)**

Goal

Instantly grab **domains and subdomains** that are **already known** to be in scope for bug bounty programs.

Why to Use It

- Saves time by giving you **ready-made targets** collected from public programs.
- Helps you **avoid out-of-scope domains** because most entries come from official bounty disclosures.
- Great for **continuous recon**—new assets are added frequently.

When to Use It

- **Before brute-forcing or heavy scans:** start with known assets first.
- **Daily or weekly** to catch **newly added** targets for active programs.
- When joining a **new bug bounty platform** (HackerOne, Bugcrowd, etc.).

Best Tip (Basic Command)

```
chaos -d target.com -key CHAOS_API_KEY -silent -o chaos_results.txt
```

- `-d` → Specify the domain scope.
- `-key` → Your free ProjectDiscovery Chaos API key.
- `-silent` → Clean output (only domains).
- `-o` → Save results.

 Get an API key: <https://chaos.projectdiscovery.io/> (free signup).

🔑 Filtering for the Best Results

1. Check Live Subdomains Only

```
cat chaos_results.txt | httpx -silent -status-code -title -tech-detect -o  
chaos_live.txt
```

- Filters out dead subs and shows:
 - HTTP **status code** (200/403 are juicy)
 - **Page title** (e.g., “Admin Panel”)
 - **Tech stack** (e.g., WordPress, Nginx)

1. Find High-Value Keywords

```
grep -iE 'admin|api|login|internal|dev' chaos_live.txt
```

- Quickly surfaces **admin dashboards, APIs, and internal hosts**.

1. Sort by Size to Spot Interesting Pages

```
cat chaos_live.txt | sort -k3 -n
```

- Larger response sizes often mean **richer functionality**.

🚀 Workflow Example

```
# 1 Fetch known bounty assets  
chaos -d example.com -key CHAOS_API_KEY -silent -o chaos_raw.txt  
  
# 2 Verify which are live and interesting  
cat chaos_raw.txt | httpx -silent -status-code -title -tech-detect | tee  
chaos_live.txt  
  
# 3 Highlight priority targets  
grep -iE "200|403|api|admin" chaos_live.txt > chaos_priority.txt
```

This gives you a **curated list of live, bounty-approved domains**—the perfect starting point before spending time on deeper recon.

Part 1.4 – crt.sh (Certificate Transparency Search)

Goal

Discover **hidden subdomains** by analyzing SSL/TLS certificates issued for the target domain.

Why to Use It

- SSL certificates must publicly list every hostname they protect.
- Catch **internal**, **testing**, or **forgotten** subdomains that other tools miss.
- Works even if the subdomain **has no DNS record yet** but has a certificate.

When to Use It

- Right after **Subfinder/Amass**, to grab extra subdomains quickly.
- When you need **fresh, real-time data** (new certs appear instantly).
- Before heavy scans—cheap way to grow your asset list fast.

Best Tip (Basic Command)

You can use **curl + grep** to pull all subdomains directly from crt.sh:

```
curl -s "https://crt.sh/?q=%25.target.com&output=json" \
| jq -r '.[].name_value' \
| sed 's/>\*\.\//g' | sort -u > crtsh_subs.txt
```

- `%25.target.com` → `%25` is URL-encoded `%` (wildcard search).
- `jq` → Extract only the subdomain names.
- `sed 's/>*\.\//g'` → Remove wildcard entries like `*.example.com`.
- `sort -u` → Remove duplicates.

 If you don't want to code, just visit <https://crt.sh>, search for `%.target.com`, and copy results.

Filtering for the Best Results

1. Verify Live Hosts Only

```
cat crtsh_subs.txt | httpx -silent -status-code -title -tech-detect -o
crtsh_live.txt
```

- Keeps only live hosts and reveals page titles & technologies.

1. Focus on Juicy Keywords

```
grep -iE 'admin|api|dev|internal|staging|test' crtsh_live.txt
```

- Surfaces subdomains likely to contain **admin panels**, **APIs**, or **dev environments**.

1. Sort by Response Size

```
cat crtsh_live.txt | sort -k3 -n
```

- Large pages often have **richer functionality or hidden inputs**.

🚀 Workflow Example

```
# 1 Extract subdomains from crt.sh
curl -s "https://crt.sh/?q=%25.example.com&output=json" \
| jq -r '.[].name_value' \
| sed 's/\/*\.\//g' | sort -u > crt_raw.txt

# 2 Check which are live and interesting
cat crt_raw.txt | httpx -silent -status-code -title -tech-detect | tee
crt_live.txt

# 3 Prioritize valuable targets
grep -iE "200|403|admin|api" crt_live.txt > crt_priority.txt
```

This quickly delivers a **fresh, high-value list of subdomains** discovered from real SSL certificates —perfect for immediate scanning.

Part 1.5 – Gau (GetAllURLs) – Harvest Historical Endpoints

🎯 Goal

Collect **historical URLs** (endpoints, parameters, APIs) of a target domain from sources like the **Wayback Machine**, Common Crawl, and AlienVault.

✅ Why to Use It

- Finds **old but still live endpoints** that modern crawlers often miss.
- Great for discovering **hidden parameters**, **admin paths**, and **retired APIs** that may still respond.
- Perfect for feeding into **fuzzers**, **parameter scanners**, or **Nuclei**.

⌚ When to Use It

- After you have a list of **domains/subdomains** (from Subfinder/Amass/crt.sh).
- Before **parameter testing** (XSS, SQLi, LFI, etc.).
- When you need **quick payload targets** for vulnerability scanners.

⚡ Best Tip (Basic Command)

```
gau target.com | tee gau_raw.txt
```

- Fetches **all historical URLs** for the given domain.

➡ To process multiple subdomains:

```
cat live_subs.txt | gau | tee gau_all.txt
```

- `live_subs.txt` → list of subdomains you verified earlier.

🔑 Filtering for the Best Results

1. Only Live URLs

```
cat gau_all.txt | httpx -silent -status-code -mc 200,403 -o gau_live.txt
```

- `-mc 200,403` → Keep only URLs that return OK (200) or Forbidden (403).

1. Extract URLs with Parameters (for XSS/LFI testing)

```
cat gau_live.txt | grep "=" | uro > gau_params.txt
```

- `uro` → Cleans duplicates and normalizes URLs.

1. Focus on Interesting File Types

```
cat gau_live.txt | grep -E "\.php|\.aspx|\.jsp|/api/" > gau_important.txt
```

- Prioritizes backend-heavy files with higher bug potential.

🚀 Workflow Example

```
# 1 Harvest URLs from all subdomains
cat subdomains.txt | gau | tee gau_raw.txt

# 2 Clean & keep only live parameterized URLs
cat gau_raw.txt | grep "=" | uro | httpx -silent -status-code -mc 200,403 -o
gau_params_live.txt

# 3 Highlight high-value endpoints
grep -iE "admin|login|debug|api|internal" gau_params_live.txt > gau_priority.txt
```

Pro Tip

Combine **Gau** output with tools like:

- **ParamSpider** → for hidden parameters
- **Nuclei** → for automated vulnerability checks
- **Burp Suite** → for deeper manual testing

This ensures you get a **clean, high-value list of historical URLs and parameters** ready for exploitation.

Part 1.6 – Hakrawler (Live Crawling for Hidden Paths & JS Files)

Goal

Crawl **live web applications** to uncover hidden **links, JavaScript files, APIs, and directories** directly from the site's HTML and JavaScript.

Why to Use It

- Finds **real-time URLs** that historical tools (like Gau) may miss.
 - Quickly grabs **JavaScript files**, which often contain API endpoints and secrets.
 - Useful for building **custom wordlists** for further fuzzing.
-

When to Use It

- After verifying **live subdomains** with `httpx`.
 - Before running **JS Secret Finder**, **ParamSpider**, or vulnerability scanners.
 - When you need **fresh links** that aren't stored in archives.
-

Best Tip (Basic Command)

```
hakrawler -url https://target.com -depth 3 -plain | tee hak_raw.txt
```

- `-url` → Target URL (can be a single domain or subdomain).
- `-depth 3` → Crawl up to 3 levels deep for more links.
- `-plain` → Clean output with URLs only.

Crawl Multiple Domains at Once

```
cat live_subs.txt | hakrawler -depth 3 -plain | tee hak_all.txt
```

- `live_subs.txt` → List of live subdomains from Subfinder/Amass.
-

🔑 Filtering for the Best Results

1. Extract JavaScript Files (for secret hunting)

```
cat hak_all.txt | grep "\.js$" | sort -u > hak_js.txt
```

1. Grab API Endpoints Only

```
cat hak_all.txt | grep -i "/api" | sort -u > hak_api.txt
```

1. Clean and Keep Unique URLs

```
cat hak_all.txt | uro > hak_clean.txt
```

- `uro` removes duplicates and normalizes URLs.

1. Check Which JS Files Are Live

```
cat hak_js.txt | httpx -silent -status-code -mc 200 -o live_js.txt
```

- Only keeps JavaScript files that return a 200 OK.

🚀 Workflow Example

```
# 1 Crawl all live subdomains for URLs & JS files
cat live_subs.txt | hakrawler -depth 3 -plain | tee hak_all.txt

# 2 Extract parameterized URLs for testing
cat hak_all.txt | grep "=" | uro > hak_params.txt

# 3 Filter high-value endpoints
grep -iE "admin|login|api|internal|debug" hak_params.txt > hak_priority.txt
```

💡 Pro Tip

Feed the JavaScript file list (`hak_js.txt`) into **SecretFinder** or **LinkFinder** to automatically extract:

- API keys
- Hidden endpoints
- Developer comments

This workflow ensures you capture **live, up-to-date attack surface data** that static tools can't provide.

Part 1.7 – ParamSpider (Hidden Parameter Discovery)

🎯 Goal

Discover **GET/POST parameters** on a target's URLs for vulnerability testing (e.g., **XSS**, **SQLi**, **LFI**).

✓ Why to Use It

- Automatically extracts **parameterized URLs** from multiple sources (Wayback Machine, commoncrawl, etc.).
 - Finds parameters that tools like Gau may miss.
 - Saves hours of manual hunting for **input points**.
-

⌚ When to Use It

- After collecting live URLs from **Gau** or **Hakrawler**.
 - Before running automated scanners or fuzzing for:
 - **XSS** (cross-site scripting)
 - **SQL injection**
 - **Open Redirects**
 - **SSRF** (server-side request forgery)
-

⚡ Best Tip (Basic Command)

```
paramspider -d target.com -o paramspider_raw.txt
```

- `-d` → Domain name.
- `-o` → Output file for results.

➡ Exclude Noisy Parameters (to reduce junk)

```
paramspider -d target.com --exclude "utm*,sessionid,fbcclid" -o paramspider_clean.txt
```

- `--exclude` → Ignore analytics or tracking parameters that usually aren't exploitable.
-

🔑 Filtering for the Best Results

1. Check Which URLs Are Live

```
cat paramspider_clean.txt | httpx -silent -status-code -mc 200,403 -o param_live.txt
```

- Keeps only URLs returning **200 (OK)** or **403 (Forbidden)**.

1. Focus on Juicy Parameters

```
grep -iE 'id=|redirect=|q=|file=|page=' param_live.txt > param_priority.txt
```

- Parameters like `id=`, `q=`, `file=`, or `redirect=` often lead to XSS/LFI/Redirect bugs.

1. Remove Duplicate URLs

```
cat param_priority.txt | uro > param_final.txt
```

- `uro` cleans duplicates and normalizes URLs for easy testing.

🚀 Workflow Example

```
# 1 Extract parameters from target  
paramspider -d example.com --exclude "utm*,sessionid" -o params_raw.txt  
  
# 2 Verify which parameterized URLs are live  
cat params_raw.txt | httpx -silent -status-code -mc 200,403 -o params_live.txt  
  
# 3 Highlight high-risk parameters  
grep -iE "id=|redirect=|q=|file=|page=" params_live.txt > params_priority.txt
```

💡 Pro Tip

Pipe `param_final.txt` into **Nuclei** or **Burp Suite** to quickly test for:

- Reflected XSS** → `id=` or `q=`
- Open Redirect** → `redirect=` or `next=`
- LFI/RFI** → `file=` or `path=`

This gives you a **clean, ready-to-attack list of exploitable parameters** with minimal noise.

Part 1.8 – Git Dorks (Hunting Secrets in Public Git Repositories)(optional but can be powerful)

🎯 Goal

Find **sensitive data** (API keys, credentials, endpoints, internal URLs) leaked in **public GitHub repositories** related to your target.

Why to Use It

- Developers often accidentally push **tokens, passwords, or internal configs** to public repos.
 - Exposed secrets can lead directly to:
 - **Account takeovers**
 - **API abuse**
 - **Infrastructure access**
 - One of the **highest impact recon steps**—single secret = massive payout.
-

When to Use It

- After confirming the **organization name, email domain, or developer usernames**.
 - During **intel gathering** before active exploitation.
 - Regularly for ongoing programs (new commits appear daily).
-

Best Tip (Search Dorks in GitHub)

GitHub dorking uses special queries in the GitHub search bar:

```
org:targetcompany "api_key"  
org:targetcompany "password"  
org:targetcompany "secret"  
org:targetcompany "Authorization"
```

- `org:targetcompany` → Searches all repos of the target org.
- `"api_key"`, `"password"` → Common leak keywords.

Search Across All GitHub (Not Just Org)

```
"target.com" "secret"  
"target.com" "apikey"  
"target.com" "bearer"
```

- Finds references to the target even in **third-party repos**.
-

Filtering for the Best Results

1. Search by Email Domain (*great for private orgs*)

```
"@target.com" password  
"@target.com" api
```

- Finds commits mentioning **company emails** and secrets.

1. Combine with File Types (*narrow to config files*)

```
org:targetcompany filename:.env  
org:targetcompany filename:config.json
```

- `.env`, `config.json`, `settings.py` often contain **API keys**.

1. Sort by Recency

- Use GitHub's **Sort → Recently Indexed** to catch **fresh leaks** before they're removed.

🔑 CLI Automation (Optional)

If you have a GitHub token, use [GitHub-Search](#) or [truffleHog](#):

```
python3 github-subdomains.py -d target.com -t GITHUB_TOKEN -o git_results.txt
```

- Pulls subdomains & secrets directly from GitHub commits.

🚀 Workflow Example

```
# 1 Search manually using GitHub dorks:  
org:examplecorp "api_key"  
org:examplecorp filename:.env  
"example.com" "Authorization"  
  
# 2 Save interesting repos or commits for follow-up.  
# 3 Clone suspicious repos to inspect commit history:  
git clone https://github.com/user/leaky-repo.git  
cd leaky-repo && git log -p
```

💡 Pro Tip

- Always **take screenshots and store commit hashes** before reporting—owners often delete repos quickly.
- Use `trufflehog` locally on cloned repos to catch **hidden historical secrets**:

```
trufflehog --regex --entropy=False ./leaky-repo/
```

🔥 Quick Win Checklist

- API keys in `.env`
- Database credentials in `config.json`
- Hardcoded tokens in `JavaScript`
- AWS/Google Cloud keys in commit history

This step often produces **directly exploitable bugs**—sometimes even before touching the target's live infrastructure.

Part 2.1 – Naabu (Fast Live Host & Port Discovery)

🎯 Goal

Quickly identify **live hosts** and **open ports** on your target subdomains for further testing.

✓ Why to Use It

- Extremely **fast** compared to traditional port scanners.
 - Identifies **which hosts are alive** and **which ports are open**.
 - Reduces scanning time before running **Nmap, HTTPX, or vulnerability scans**.
-

⌚ When to Use It

- After compiling a **list of live subdomains** (Subfinder/Amass/crt.sh).
 - Before running **Nmap NSE scripts** or **HTTP service scans**.
 - Useful for **large scopes** with hundreds or thousands of hosts.
-

⚡ Best Tip (Basic Command)

```
naabu -list live_subs.txt -top-ports 1000 -o naabu_ports.txt
```

- `-list` → File containing live subdomains.
- `-top-ports 1000` → Scan the 1000 most common ports.
- `-o` → Save results.

➡ Scan Custom Ports

```
naabu -list live_subs.txt -p 21,22,80,443,8080,8443 -o naabu_custom.txt
```

- Focuses on ports **most likely to host web or critical services**.
-

🔑 Filtering for the Best Results

1. Keep only hosts with open HTTP/HTTPS ports

```
grep -E ":80|:443|:8080|:8443" naabu_ports.txt > web_hosts.txt
```

- Quickly isolates web targets for HTTP scanning.

1. Prioritize unusual open ports

```
grep -v -E ":80|:443" naabu_ports.txt > unusual_ports.txt
```

- Finds **hidden services**, like admin panels on non-standard ports.

1. Check for alive hosts only

```
cat web_hosts.txt | httpx -silent -status-code -title -tech-detect -o  
live_web.txt
```

- Filters hosts that respond over HTTP(S), showing status, title, and tech stack.

Workflow Example

```
# 1 Scan top 1000 ports on live subdomains  
naabu -list live_subs.txt -top-ports 1000 -o naabu_ports.txt  
  
# 2 Extract web services for further scanning  
grep -E ":80|:443|:8080|:8443" naabu_ports.txt > web_hosts.txt  
  
# 3 Verify live web hosts and capture basic info  
cat web_hosts.txt | httpx -silent -status-code -title -tech-detect -o  
live_web.txt
```

Pro Tip

- Run Naabu **before Nmap** to **reduce scan time**—Nmap can focus on **hosts with open ports only**.
- Use `-exclude-cdn` if you want to skip common CDN IPs to avoid scanning irrelevant hosts.

This step ensures you have a **concise list of live, exploitable hosts** ready for service scanning and vulnerability testing.

Part 2.2 – Nmap NSE (Service & Vulnerability Scanning)

Goal

Identify **services, versions, and known vulnerabilities** on open ports of your target hosts.

Why to Use It

- Nmap is the **industry standard** for port/service scanning.
- NSE (Nmap Scripting Engine) allows **automated vulnerability detection**.
- Helps **prioritize targets** before deeper exploitation.

When to Use It

- After **Naabu** identifies open ports.
 - To map **services** (HTTP, SSH, FTP, databases, etc.) and detect potential **CVEs**.
 - Before running targeted **Nuclei templates** or manual tests.
-

Best Tip (Basic Command)

```
nmap -iL open_hosts.txt -sv --script vuln -oN nmap_vulns.txt
```

- `-iL` → Input file with hosts/IPs.
- `-sv` → Detect service versions.
- `--script vuln` → Run vulnerability detection scripts.
- `-oN` → Save output in readable format.

Faster Scan for Specific Ports

```
nmap -iL open_hosts.txt -p 80,443,8080,8443 -sv --script vuln -oN nmap_web.txt
```

- Focuses on **web ports** only, saving time on large scopes.
-

Filtering for the Best Results

1. Only show hosts with vulnerabilities

```
grep -i "VULNERABLE" nmap_vulns.txt > nmap_high_risk.txt
```

- Quickly highlights hosts with **known CVEs**.

1. Extract web service info

```
grep -E "80/tcp|443/tcp|8080/tcp|8443/tcp" nmap_vulns.txt > web_services.txt
```

- Focuses on **HTTP/S services** for further testing.

1. Check for outdated versions

```
grep -E "Apache|Nginx|OpenSSH" nmap_vulns.txt | grep -E "[0-9]{1,2}\.[0-9]" > outdated_services.txt
```

- Flags targets running **old versions** likely vulnerable.
-

Workflow Example

```
# 1 Scan open hosts discovered by Naabu  
nmap -iL open_hosts.txt -sV --script vuln -oN nmap_full.txt  
  
# 2 Highlight hosts with vulnerabilities  
grep -i "VULNERABLE" nmap_full.txt > nmap_high_risk.txt  
  
# 3 Focus on web services for HTTP scanning  
grep -E "80/tcp|443/tcp|8080/tcp|8443/tcp" nmap_full.txt > web_services.txt
```

Pro Tip

- Use Nmap's `--script vuln` for automated CVE checks, but follow up manually—some findings may be false positives.
- Combine **Nmap + HTTPX + WhatWeb** to prioritize live hosts with exploitable services.

This step ensures you know **which hosts and services are the most promising** for further testing.

Part 2.3 – HTTPX (Verify Live HTTP Services & Collect Basic Info)

Goal

Quickly verify which subdomains or hosts **serve web content** and gather **status codes, titles, and technologies**.

Why to Use It

- Confirms **live HTTP/S services** before running deeper scans.
- Provides **page title, status code, and tech stack** for prioritization.
- Works **faster than full Nmap web scans**, ideal for large scopes.

When to Use It

- After **Naabu or Nmap** identifies hosts with open web ports.
- Before **WhatWeb, Nuclei, or fuzzing** to filter out dead or irrelevant targets.
- Useful when you need a **quick snapshot** of the target's web infrastructure.

⚡ Best Tip (Basic Command)

```
cat web_hosts.txt | httpx -silent -status-code -title -tech-detect -o httpx_live.txt
```

- `-silent` → Cleaner output.
- `-status-code` → HTTP response codes (200, 403, etc.).
- `-title` → Captures the page title for prioritization.
- `-tech-detect` → Detect underlying technologies (WordPress, Nginx, etc.).
- `-o` → Save output to a file.

➡ Scan Multiple Domains

```
httpx -l live_subdomains.txt -status-code -title -tech-detect -o httpx_all.txt
```

- `-l` → File with subdomains instead of piping input.

🔑 Filtering for the Best Results

1. Keep only live/interesting pages

```
grep -E "200|403|401" httpx_live.txt > httpx_priority.txt
```

- Focuses on pages that respond **successfully or restrict access**, often valuable.

1. Highlight admin, login, or dev pages

```
grep -iE "admin|login|dashboard|dev|staging" httpx_priority.txt > httpx_highvalue.txt
```

1. Sort by Technology

```
cat httpx_highvalue.txt | sort -u -k3
```

- Helps **prioritize targets by tech stack**, e.g., outdated WordPress or Apache servers.

🚀 Workflow Example

```
# 1 Verify live HTTP(S) hosts
cat web_hosts.txt | httpx -silent -status-code -title -tech-detect -o httpx_live.txt

# 2 Highlight high-value pages
grep -iE "200|403|401" httpx_live.txt | grep -iE "admin|login|dashboard" > httpx_priority.txt

# 3 Optionally sort by detected tech stack
cat httpx_priority.txt | sort -u -k3 > httpx_sorted.txt
```

Pro Tip

- Combine **HTTPX + Nuclei**: Only scan hosts with valid HTTP responses to **avoid wasting templates on dead hosts**.
- Use status codes and page titles to **quickly find admin panels, staging sites, and potentially misconfigured endpoints**.

This step produces a **filtered, actionable list of live web assets** ready for vulnerability scanning or fuzzing.

Part 2.4 – WhatWeb (Technology Detection)

Goal

Identify the **technologies and software** running on a web application, such as **CMS, frameworks, web servers, and plugins**.

Why to Use It

- Helps **prioritize targets** based on known vulnerabilities in specific software versions.
 - Provides insight for **Nuclei templates, manual exploits, or fuzzer customization**.
 - Quick way to detect **WordPress, Joomla, Drupal, Apache, Nginx**, etc.
-

When to Use It

- After verifying **live web hosts** with HTTPX.
 - Before running **targeted vulnerability scans**.
 - When deciding **which targets are easier or more valuable** to test first.
-

Best Tip (Basic Command)

```
cat live_web.txt | whatweb -v --log-verbose whatweb_results.txt
```

- `-v` → Verbose output.
- `--log-verbose` → Save full details including plugins, frameworks, and server info.
- `whatweb_results.txt` → Output file for later filtering.

Scan a Single Domain

```
whatweb https://target.com -v --log-verbose whatweb_target.txt
```

Filtering for the Best Results

1. Focus on CMS or Outdated Software

```
grep -iE "WordPress|Joomla|Drupal|Apache|Nginx" whatweb_results.txt > tech_priority.txt
```

- Quickly identifies **targets with exploitable tech stacks**.

1. Highlight Version Numbers

```
grep -iE "[0-9]{1,2}\.[0-9]{1,2}" tech_priority.txt > outdated_tech.txt
```

- Flags **old versions** potentially vulnerable to CVEs.

1. Sort by Target Importance

```
cat outdated_tech.txt | sort -u > final_tech_priority.txt
```

- Keeps a **clean, deduplicated list of exploitable tech stacks**.

Workflow Example

```
# 1 Scan all live web hosts  
cat live_web.txt | whatweb -v --log-verbose > whatweb_results.txt  
  
# 2 Filter for CMS or web servers  
grep -iE "WordPress|Joomla|Drupal|Apache|Nginx" whatweb_results.txt > tech_priority.txt  
  
# 3 Focus on outdated versions  
grep -iE "[0-9]{1,2}\.[0-9]{1,2}" tech_priority.txt > outdated_tech.txt
```

Pro Tip

- Combine **WhatWeb + Nuclei**: Use detected CMS versions to **run only relevant templates**, avoiding unnecessary scans.
- Keep an eye on **plugins**—they often have publicly known vulnerabilities even if the main CMS is updated.

This step ensures you **know exactly what software and versions your target is running**, helping you **prioritize exploitable hosts efficiently**.

Part 2.5 – Nuclei (Automated Vulnerability Scanning)

Goal

Run **predefined vulnerability templates** against live targets to quickly identify **security issues** like misconfigurations, CVEs, and exposure risks.

Why to Use It

- Automates **thousands of checks** in seconds using ProjectDiscovery's template library.
 - Covers a wide range of vulnerabilities: **XSS, SQLi, LFI, open ports, misconfigurations**.
 - Saves time compared to **manual scanning**, especially for large scopes.
-

When to Use It

- After confirming **live hosts** with HTTPX.
 - After identifying **tech stacks** with WhatWeb to focus on relevant templates.
 - Daily or ongoing scans to **catch new vulnerabilities**.
-

Best Tip (Basic Command)

```
nuclei -l live_web.txt -t ~/nuclei-templates/ -o nuclei_results.txt
```

- `-l` → List of live hosts or URLs.
- `-t` → Path to the template directory.
- `-o` → Output file with results.

Focus on High-Severity Vulnerabilities

```
nuclei -l live_web.txt -t ~/nuclei-templates/ -severity critical,high -o nuclei_high.txt
```

- Only shows **critical or high-impact findings** for quicker triage.
-

Filtering for the Best Results

1. Deduplicate Findings

```
cat nuclei_high.txt | sort -u > nuclei_final.txt
```

- Removes duplicates from multiple templates triggering on the same host.

1. Filter by Specific Vulnerability Type

```
grep -i "subdomain-takeover" nuclei_final.txt > takeover_targets.txt
```

- Focuses on **subdomain takeover vulnerabilities** or any other specific bug class.

1. Sort by Host/URL

```
sort -k2 nuclei_final.txt > nuclei_sorted.txt
```

- Makes it easy to **triage findings by host**.

🚀 Workflow Example

```
# 1 Scan all live web URLs for vulnerabilities  
nuclei -l live_web.txt -t ~/nuclei-templates/ -o nuclei_results.txt  
  
# 2 Focus on critical/high vulnerabilities  
nuclei -l live_web.txt -t ~/nuclei-templates/ -severity critical,high -o  
nuclei_high.txt  
  
# 3 Deduplicate and sort  
cat nuclei_high.txt | sort -u > nuclei_final.txt  
sort -k2 nuclei_final.txt > nuclei_sorted.txt
```

💡 Pro Tip

- **Update Nuclei templates daily** to catch the latest CVEs:

```
nuclei -update-templates
```

- Combine Nuclei findings with **HTTPX + WhatWeb**: scan only **live hosts with known software versions** to avoid unnecessary scans.

This step ensures you **quickly identify actionable vulnerabilities** and can focus on **high-value targets first**.

Part 2.6 – JS Secret Finder (Extract Secrets from JavaScript Files)

🎯 Goal

Automatically extract **API keys, tokens, endpoints, and secrets** from **JavaScript files** hosted on target websites.

✅ Why to Use It

- Developers often **hardcode secrets** in JS files.
- Provides **high-value targets** for API abuse, authentication bypass, or internal service access.
- Works well in combination with **Hakrawler**, which collects JS URLs.

When to Use It

- After **Hakrawler** or similar tools have crawled **JS files** from live hosts.
 - Before running **manual tests or fuzzing APIs**.
 - Ideal during **post-recon and pre-exploitation** phases.
-

Best Tip (Basic Command)

```
python3 SecretFinder.py -i live_js.txt -o cli
```

- `-i` → Input file containing JS URLs (from Hakrawler).
- `-o cli` → Output directly to terminal for quick review.

Save results to a file

```
python3 SecretFinder.py -i live_js.txt -o results.txt
```

Filtering for the Best Results

1. Focus on API Keys or Tokens

```
grep -i "api_key\|token\|secret\|client_id" results.txt > secrets_highvalue.txt
```

- Filters output for **most exploitable secrets**.

1. Check Only Live JS URLs

```
cat live_js.txt | httpx -silent -status-code -mc 200 -o live_js_checked.txt
```

- Ensures you only scan **JS files that actually exist**.

1. Prioritize By File Location

```
grep -iE "/static/|/assets/|/js/" secrets_highvalue.txt
```

- Files in `/static/` or `/assets/` often contain **production keys** or API endpoints.
-

Workflow Example

```
# 1 Extract JS URLs from Hakrawler  
cat live_subs.txt | hakrawler -depth 3 -plain | grep "\.js$" | sort -u > live_js.txt  
  
# 2 Run SecretFinder  
python3 SecretFinder.py -i live_js.txt -o cli | tee secrets_raw.txt  
  
# 3 Filter for high-value secrets  
grep -i "api_key\|token\|secret" secrets_raw.txt > secrets_highvalue.txt
```

Pro Tip

- Feed **high-value JS secrets** into **Burp Suite** or **custom scripts** to test API endpoints.
- Always check if the **API key or token is live**—some may be old or invalid.

This step helps you **uncover hidden, high-impact targets** that often lead to **direct exploit opportunities**.

Part 2.7 – Aquatone (Visual Recon & Screenshotting)

Goal

Take **automated screenshots** of live web assets to quickly identify **interesting pages**, dashboards, or misconfigurations.

Why to Use It

- Provides a **visual overview** of large scopes.
- Helps spot **admin panels, staging environments, login pages**, or unusual endpoints.
- Useful for **documentation** and **triaging high-value targets**.

When to Use It

- After confirming **live web hosts** with **HTTPX**.
- Before starting **manual testing**, to prioritize visually interesting pages.
- For **reporting**, since screenshots help **demonstrate findings**.

Best Tip (Basic Command)

```
cat live_web.txt | aquatone -out aquatone_report
```

- `-out` → Directory to save screenshots and reports.
- Generates **HTML report** and **images** for each live host.

Advanced Options

```
cat live_web.txt | aquatone -out aquatone_report -ports xlarge -threads 50
```

- `-ports xlarge` → Scan more ports.
- `-threads 50` → Speed up screenshotting for large scopes.

Filtering for the Best Results

1. Focus on Active Pages (200/403)

- Open `aquatone_report/aquatone_report.html` and filter **status codes** to prioritize pages that respond successfully or with restricted access.

1. Highlight Admin, Login, or Staging Pages

```
grep -iE "admin|login|dashboard|staging|dev" aquatone_report/*.html > aquatone_priority.txt
```

- Finds pages most likely to have **high-value access points**.

1. Sort by Host/Domain

```
cat aquatone_priority.txt | sort -u > aquatone_sorted.txt
```

- Deduplicates and organizes targets for **quick review**.

Workflow Example

```
# 1 Screenshot all live web hosts  
cat live_web.txt | aquatone -out aquatone_report  
  
# 2 Extract visually interesting pages  
grep -iE "admin|login|dashboard|staging" aquatone_report/*.html > aquatone_priority.txt  
  
# 3 Sort and remove duplicates  
cat aquatone_priority.txt | sort -u > aquatone_sorted.txt
```

Pro Tip

- Combine **Aquatone screenshots + HTTPX page titles** for faster triage.
- Focus on pages with **status 200/403**—these are often **admin panels or restricted endpoints** worth testing first.

This step ensures you have a **visual, high-priority map of live assets**, making it easier to spot **potential high-value targets** quickly.

Part 2.8 – Cron + Alerts (Tracking New or Changed Assets)

Goal

Automatically track **new subdomains, URLs, or vulnerabilities** over time to catch **fresh targets** before competitors.

Why to Use It

- Bug bounty targets **change frequently**—new assets appear, old ones disappear.
 - Running **regular recon** ensures you never miss new endpoints.
 - Automates **comparison between scans**, highlighting **what's new or valuable**.
-

When to Use It

- For **ongoing bug bounty programs** with multiple domains.
 - After your initial **full recon** is complete.
 - Daily or weekly, depending on the **scope update frequency**.
-

Best Tip (Basic Cron + Diff Workflow)

1. Save today's results

```
cat live_subs.txt | sort -u > today.txt
```

1. Compare with yesterday

```
diff <(sort yesterday.txt) <(sort today.txt) > new_assets.txt
```

- Shows only **new subdomains or URLs** added since the last scan.
-

Filtering for the Best Results

1. Check which new assets are alive

```
cat new_assets.txt | httpx -silent -status-code -title -tech-detect -o new_live.txt
```

- Filters **only live hosts**, highlighting **valuable changes**.

1. Highlight high-priority endpoints

```
grep -iE "admin|login|dashboard|staging|api" new_live.txt > new_priority.txt
```

- Focus on **high-value pages** for immediate testing.

1. Track changes in vulnerabilities

- Save **Nuclei** or **SecretFinder** results daily:

```
diff <(sort old_nuclei.txt) <(sort new_nuclei.txt) > new_vulns.txt
```

- Identifies **newly discovered vulnerabilities**.

🚀 Workflow Example

```
# 1 Save today's live hosts  
cat live_subs.txt | sort -u > today.txt  
  
# 2 Compare with previous scan  
diff <(sort yesterday.txt) <(sort today.txt) > new_assets.txt  
  
# 3 Verify which new hosts are alive  
cat new_assets.txt | httpx -silent -status-code -title -tech-detect -o  
new_live.txt  
  
# 4 Highlight admin or high-value endpoints  
grep -iE "admin|login|dashboard|api|staging" new_live.txt > new_priority.txt
```

💡 Pro Tip

- Schedule this workflow in **cron** to run daily:

```
0 2 * * * /path/to/recon_script.sh
```

- Always **archive yesterday's results** for comparison.
- Combining this with **HTTPX**, **Nuclei**, and **SecretFinder** ensures you **catch all new high-value targets immediately**.

This step makes recon **continuous and proactive**, ensuring you never miss new attack surfaces or vulnerabilities in active programs.

[Start: Target Domain]

```
|  
Subfinder  
|  
Amass  
|  
Chaos  
|  
crt.sh  
|  
HTTPX -----  
| \  
Gau Hakrawler
```

```
|           |
ParamSpider      JS Files
|           |
Nuclei          SecretFinder
|
Git Dorks
|
Naabu
|
Nmap
|
HTTPX (verify live web)
|
WhatWeb (tech detection)
|
Nuclei
|
Aquatone (visual mapping)
|
Cron + Alerts / Diff (continuous)
```

Key Notes / Best Practices

- **Always filter for live hosts first** before running heavy scans.
- **Prioritize high-value targets:** admin panels, staging, unusual ports, API endpoints.
- **Combine tools** for cross-validation: HTTPX + WhatWeb + Nuclei → reduces false positives.
- **Continuous monitoring** ensures you catch new targets or secrets over time.
- **Output files are your “golden chain”**—feed results from one tool to the next to maintain order.