Foot printing (Information Gathering)

Foot printing :

- Foot printing is the systematic collection of publicly available (and optionally directly queried) information about a target (domain, network, web app, person, organization) to build an asset/profile map.
- It's the first phase of any security assessment or attack lifecycle — and the best place for defenders to find and reduce exposure.

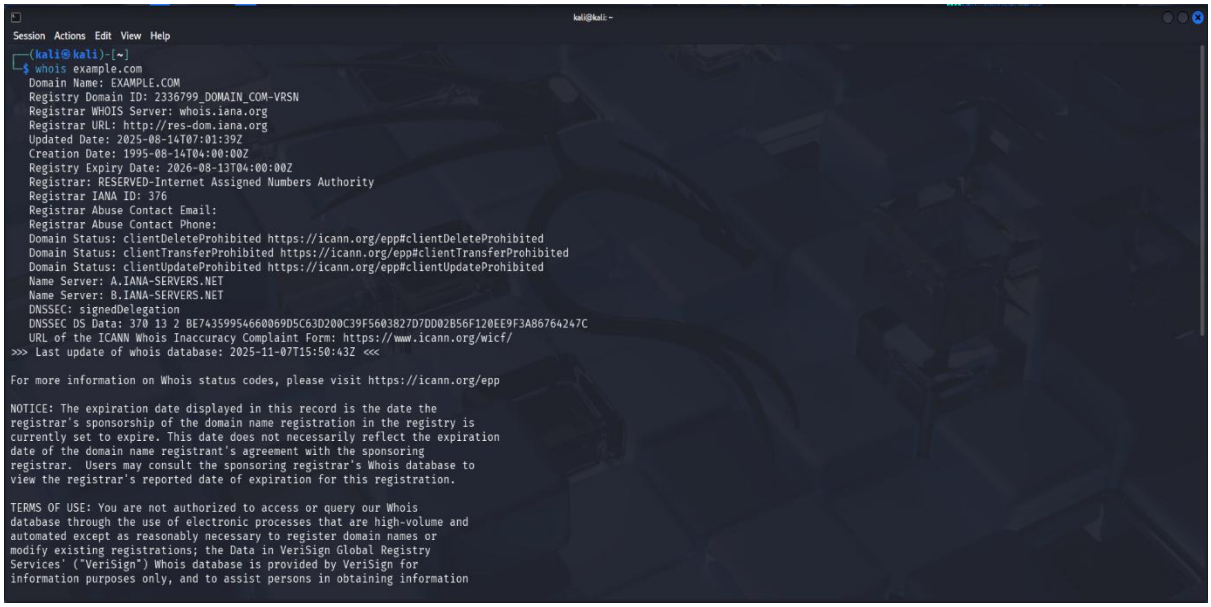
Type of foot printing :

1. Passive :
   - Collecting information about a target **without directly contacting the target's systems**.
   - You use publicly available sources (search engines, archives, public APIs, third-party indexes) so the target's infrastructure isn't queried and is less likely to detect you.
   - Build an initial picture of the target safely and legally; good for discovery and reconnaissance when you must avoid detection.
2. Active :
   - Gathering information by **directly interacting with the target's infrastructure or services**.
   - Examples: DNS queries to authoritative servers, HTTP requests, crawling, traceroute, or controlled port checks. These actions touch the target and are likely to be logged.
   - To confirm reachability, enumerate live hosts/services, discover hidden endpoints, or collect banners/version info that passive sources don't show.

Various step of information gathering :

1) WHOIS (domain registrar info)

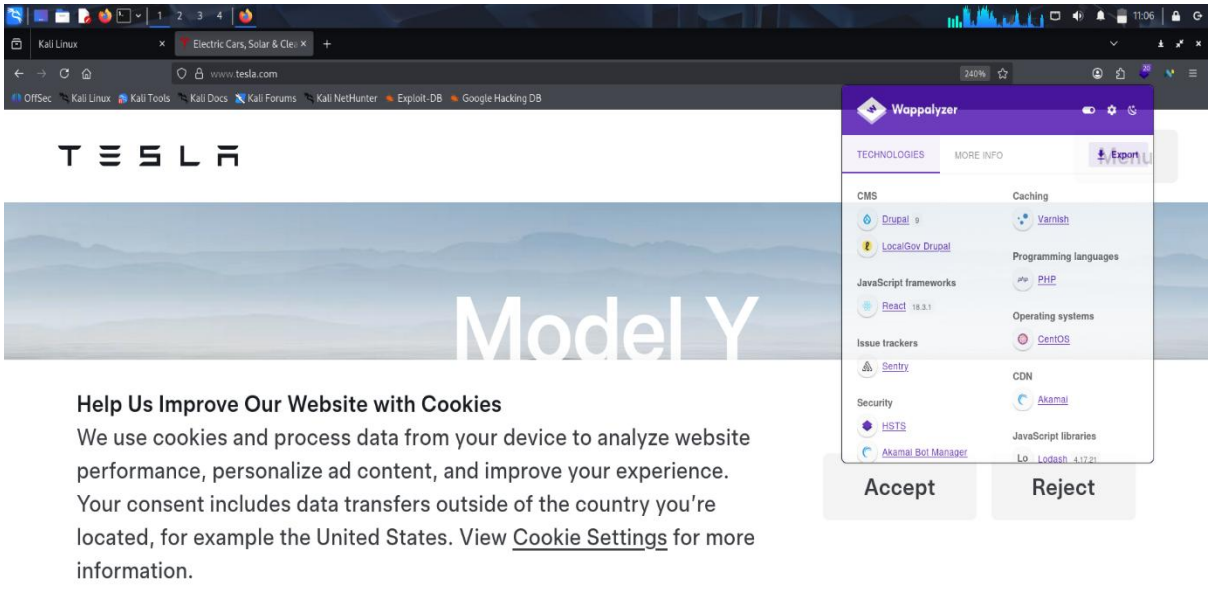
- **Goal:** Learn what WHOIS reveals and how to interpret registrar/expiry data.
- **Setup:** Create a fake test domain in a local hosts file (e.g., lab-example.local) and also use a public test domain you own OR use example.com for demonstration (passive read-only).



- **Observe:** registrar, creation/expiry, name servers, registrant contact (may be redacted).
- **Deliverable:** short note: registrar | created | expires | nameservers | privacy on/off.
- **Precautions:** WHOIS is passive; avoid querying personal/private domains without consent.

2) Wappalyzer / BuiltWith (tech fingerprinting)

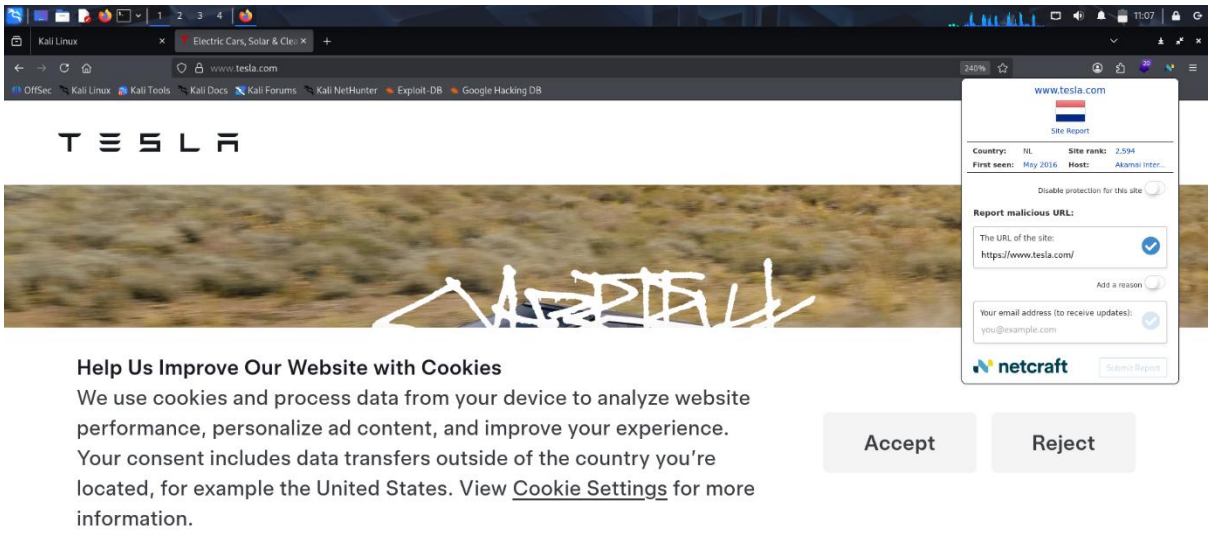
- **Goal:** Identify web technologies used by a target site.
- **Setup:** Run OWASP Juice Shop on target VM and open its web UI.
- Install Wappalyzer browser extension on your Kali desktop browser OR run:



- **Observe:** frameworks, server, JS libraries, analytics, CDN.
- **Deliverable:** table: component | evidence (header, script, cookie).
- **Precautions:** Passive detection — run only against lab targets.

3) Netcraft (historical/hosting info)

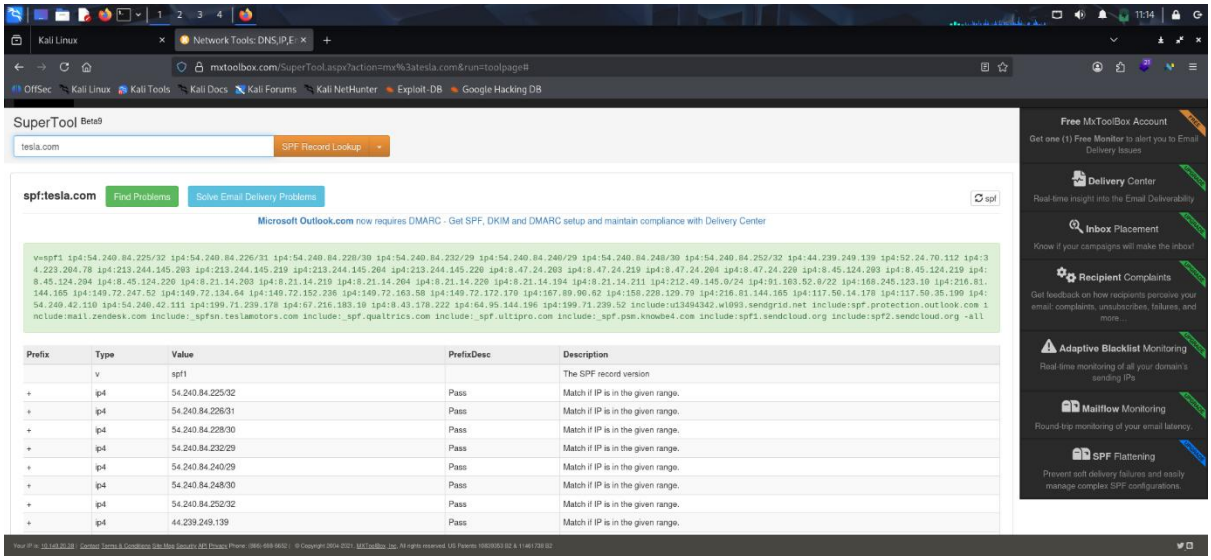
- **Goal:** See hosting/provider metadata and historical hosting if available.
- **Setup:** Use a public demo site you own OR use results from Netcraft for a public benign domain (only lookup).
- **Steps:** Visit Netcraft's site and enter the domain; note historical hosting, server header info.



- **Observe:** hosting provider, server/version, historical hosting changes.
- **Deliverable:** note of hosting provider + any interesting historical change.
- **Precautions:** Passive lookup only.

4) MX/DNS lookups (dig, nslookup, mxlookup)

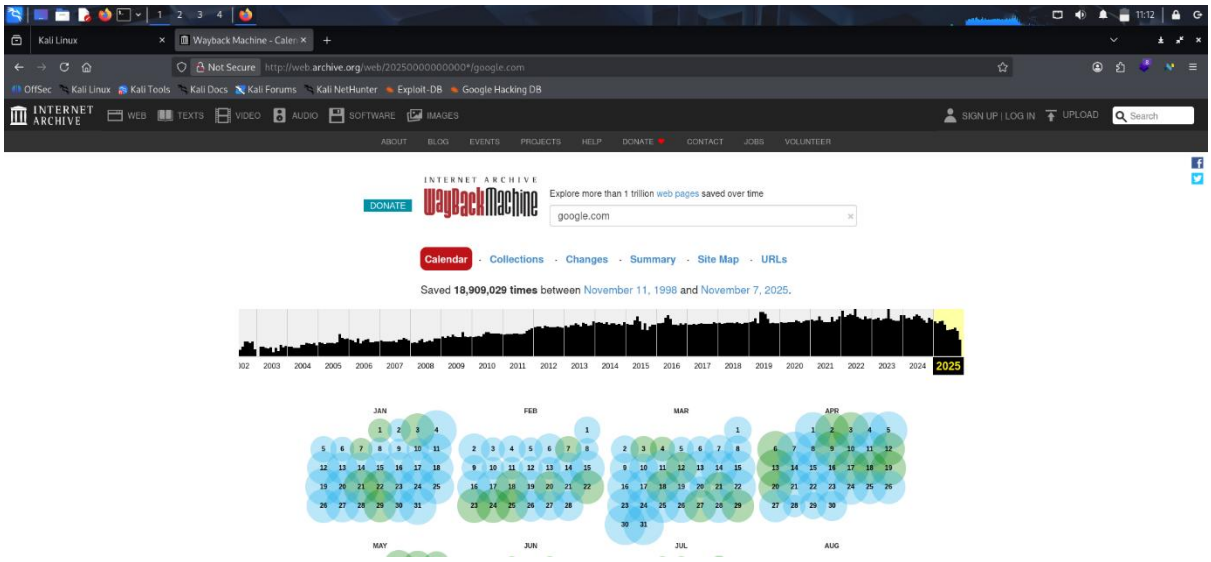
- **Goal:** Extract DNS records and mail configuration (MX, TXT/SPF, NS).
- **Setup:** Use a domain you control for full visibility. For lab practice use dig against public resolver for example.com.



- **Observe:** MX hostnames, SPF/DKIM/DMARC TXT entries, authoritative NS.
- **Deliverable:** DNS table: record type | value | notes (third-party services, misconfig).
- **Precautions:** DNS queries are low-impact but respect scope.

5) Wayback Machine (archive discovery)

- **Goal:** Discover historical pages/removed content that may expose endpoints.
- **Setup:** Use the Wayback Machine web UI for a benign public domain (or your own).
- **Steps:** Search https://web.archive.org/web/*/https://<your-domain> and browse snapshots.

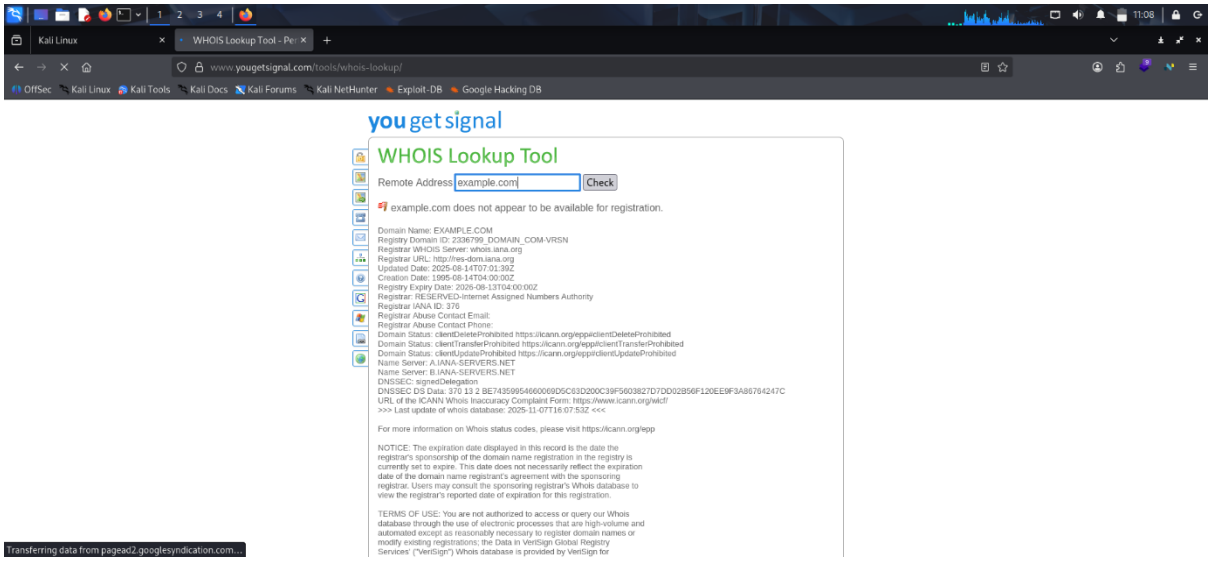


- **Observe:** old paths, previous pages, removed content, backup filenames.
- **Deliverable:** list of any sensitive-looking endpoints with snapshot date.

- **Precautions:** Passive only.

6) YouGetSignal-style reachability (simulate with nc / local checks)

- **Goal:** Understand remote port reachability; **do not** probe external hosts without permission.
- **Setup:** On target VM open a service (e.g., python -m http.server 8000) and from Kali test reachability.



- **Observe:** connection success/failure, open port presence.
- **Deliverable:** table: target IP | port | reachable | service**.**
- **Precautions:**Use local lab only. If using third-party web tools (YouGetSignal), be aware they perform remote checks.

7) GHDB (Google Dorks) & GitHub search

- **Goal:** Learn how Google dorks and GitHub searches reveal exposed files and credentials.
- **Setup:** Use Google with safe dorks on your own test repo or on public benign sites you manage.
- **Example dorks:**
- site:github.com "your-organization" "API_KEY"
- site:example.com filetype:pdf "confidential"
- **Observe:** exposed files, leaked config snippets, references to hosts/APIs.
- **Deliverable:** list of dorks used + results + remediation suggestions.

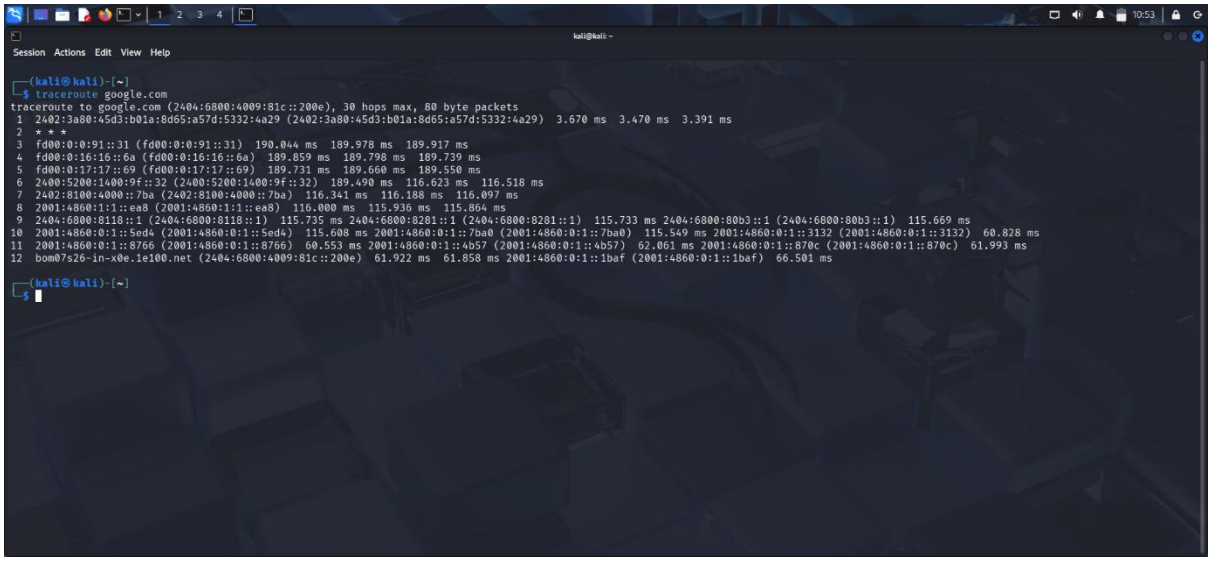
- **Precautions:** Do not use aggressive dorks on domains you don't control. Respect robots and privacy.

8) Shodan (internet-exposed service discovery)

- **Goal:** See what Shodan can reveal about exposed services (banners, ports).
- **Setup:** Use Shodan web UI or CLI against your lab public IP only if you expose it to internet (not recommended). Safer: use Shodan to search for your own public IPs that you control.
- **Example CLI:**
- shodan host <your-public-ip>
- **Observe:** service banners, ports, device type (if indexed).
- **Deliverable:** screenshot/export of Shodan host output + mitigation plan.
- **Precautions:** Don’t index or expose lab VMs to the public network unless intentional and secure.

9) traceroute / tracert (network path mapping)

- **Goal:** Map routing path from attacker to target and identify hops.
- **Setup:** Use local lab network.



- **Observe:** intermediate hops, latency spikes, possible NATs/routers.
- **Deliverable:** traceroute output + notes on network topology.

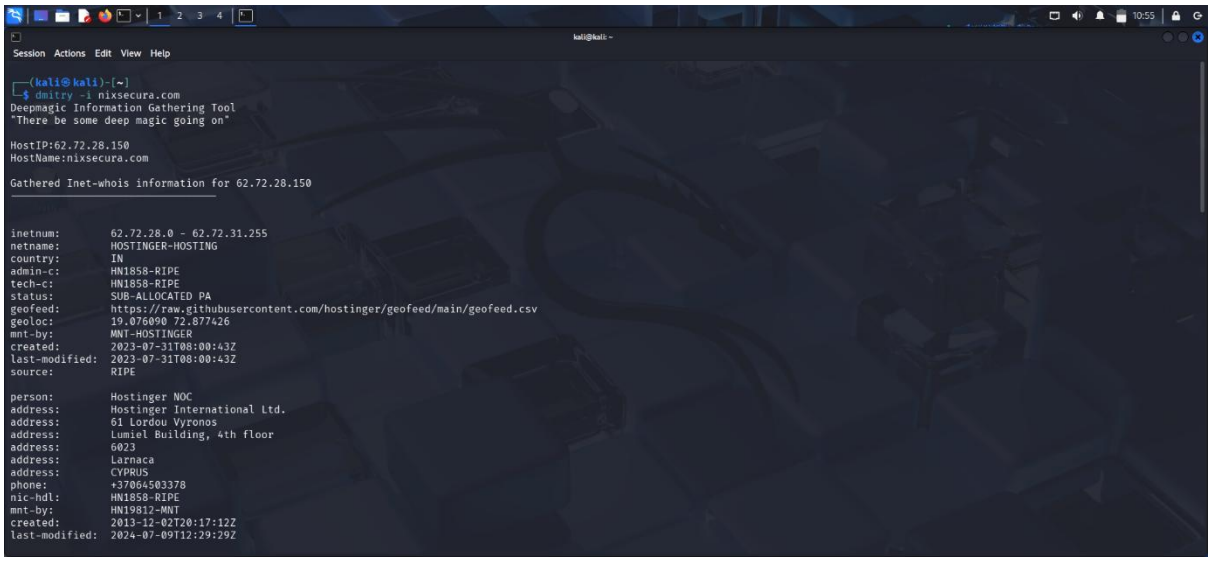
- **Precautions:** traceroute is low-risk; avoid using it for external reconnaissance without permission.

10) SpiderFoot (automated OSINT aggregator)

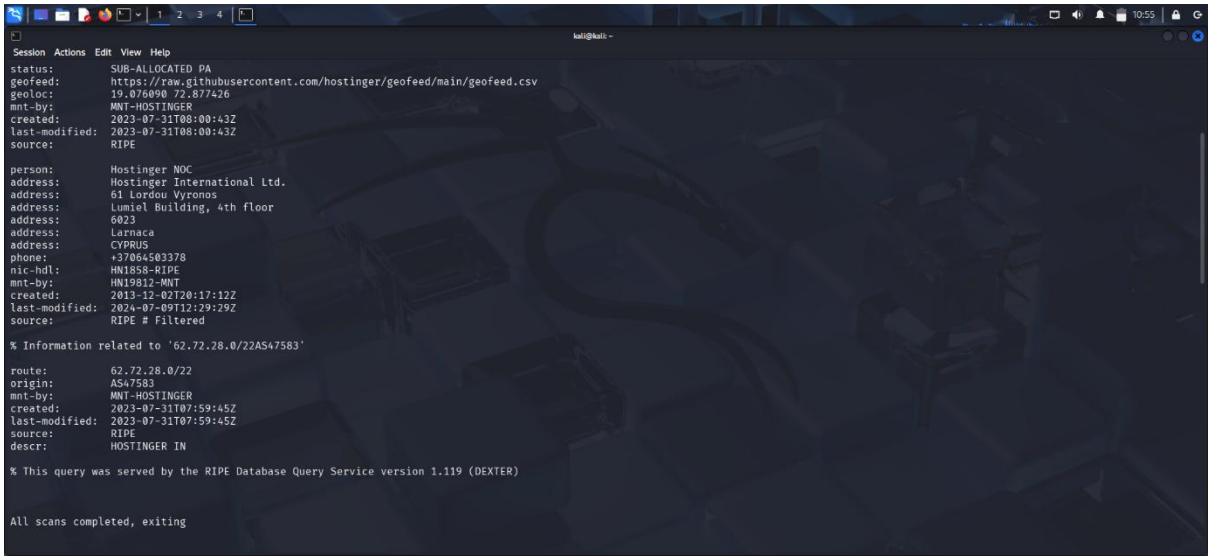
- **Goal:** Use SpiderFoot to aggregate passive OSINT into one report.
- **Setup:** Install SpiderFoot on attacker VM and configure with no active modules enabled unless authorized. Point at example.com or your own domain.
- **Example:**
- spiderfoot -s example.com -o spider_report.html
- **Observe:** subdomains, certs, public leaks, email addresses, third-party links.
- **Deliverable:** export HTML report + CSV of discovered assets.
- **Precautions:** Disable active modules (port scans, zone transfers) unless in-scope.

11) Dmitry & theHarvester (quick harvest tools)

- **Goal:** Compare outputs from small footprinting tools.
- **Setup:** Use lab domain.



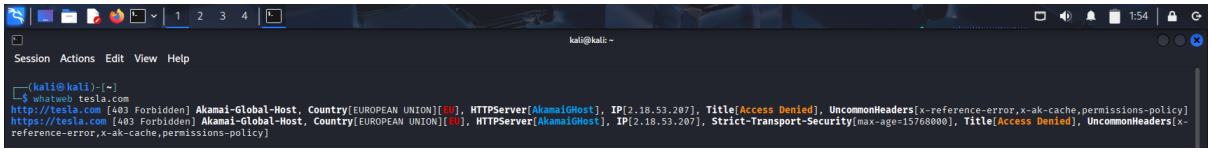
- **Observe:** email lists, subdomains, netcraft/whois results aggregated.
- **Deliverable:** combined list of emails/subdomains found and confidence level.
- **Precautions:** theHarvester uses public sources; be mindful of API limits and terms.



- **Goal:** Correlate all findings into a single asset inventory and prioritize.
- **Steps:** Merge outputs (WHOIS, DNS, subdomains, certs, services) into a spreadsheet with columns: asset | type | evidence | source | confidence | remediation.
- **Deliverable:** final spreadsheet + short slide with top 5 risky findings and mitigation.

12) Whatweb

- **Fast tech fingerprinting**: finds server types, CMS, frameworks, common plugins and sometimes version hints.
- **Prioritization**: helps you decide which follow-up checks to run (e.g., look for WordPress plugins, Node endpoints, or outdated server versions).
- **Defensive check**: defenders run it against their own sites to see what version/metadata they leak and what to harden.
- **Mostly active (low-noise)**: **WhatWeb** makes HTTP requests to the target and inspects responses.
- It's less noisy than a port scan but still *touches* the target, so it's active and will be logged by the target's web servers.
- Use it only on sites you own or have explicit permission to test.



13) wafw00f

- wafw00f is a small Python tool that detects and fingerprints Web Application Firewalls (WAFs) protecting a website.

- It starts with a normal HTTP request and, if needed, sends additional crafted requests to match signatures in its database.
- Use it to learn whether a site is behind a WAF and (often) which vendor/product is present.
- **Role in footprinting**: it's an *active* web-fingerprinting step used after passive OSINT to confirm whether a target is protected by a WAF and to inform further (authorized) testing.