Reconnaissance Techniques and Subdomain Enumeration

Tiago Almeida

October 1, 2025

Abstract

This report presents the resolution of the exercises from the second class of the *Theory and Practice of Security Attacks* course. The assignment consisted of applying reconnaissance techniques and tools to the domain **chime.com**. The following sections document the methodology, commands, results, and findings from passive and active subdomain enumeration, HTTP service discovery, URL scanning, and Google dorking performed against the target domain.

Contents

1 Passive subdomain enumeration

1.1 Tools used

For this exercise I used both recommended tools: **subfinder** and **assetfinder**. The reasons for using both were:

- to learn how to run and interpret the outputs of both tools;
- to build a more complete initial subdomain list (`subs.txt`);
- to be able to compare each tool's contributions and identify which one found the most useful domains (discussed later in the Conclusion).

1.2 Methodology

I ran each tool independently, applied lightweight filtering to reduce obvious noise, collected the raw outputs for reproducibility, and then merged and deduplicated the results. I also ran a small statistics script to compare the per-tool results before merging so I could quantify overlap and unique findings.

1.3 Commands

```
1
2  # Run subfinder
3
4  subfinder -d chime.com -silent | sort -u | grep -v '*' >
       subfinder_out.txt
5
6  # Run assetfinder
7
8  assetfinder --subs-only chime.com | sort -u | grep -v '*' >
       assetfinder_out.txt
9
10 # Run a small stats script to compare outputs
11
12 python3 stats.py assetfinder_out.txt subfinder_out.txt
13
14 # Merge and deduplicate into passive_subs.txt
15
16 cat subfinder_out.txt assetfinder_out.txt | sort -u > passive_subs.
       txt
```

1.4 Results

The statistics script produced the following output:

```
====================================
|               STATS                |
====================================

- Number of subdomains found
     assetfinder_out.txt :  85
     subfinder_out.txt :  269

- Number of subdomains found
     By both tools :  77
     By just assetfinder_out.txt :  8
     By just subfinder_out.txt :  192

- Number of INVALID subdomains found (no <chime.com> suffix)
     By both tools :  0
     By just assetfinder_out.txt :  0
     By just subfinder_out.txt :  0

====================================
```

Key observations from the results:

- **subfinder** returned substantially more candidate subdomains than **assetfinder**. Many of **assetfinder**'s results were also present in **subfinder**.
- There were no obviously invalid entries (for example, external domains incorrectly listed as `chime.com` subdomains).
- The overlap (77 entries) indicates a solid common core; the larger number of unique results from **subfinder** warrants manual inspection for quality (quantity does not imply usefulness).

Note: During an earlier run, **assetfinder** returned an unexpected entry (`google.com`), which motivated the validation step.

2 Active subdomain enumeration

2.1 Tools and wordlists

For the active enumeration I used a single tool: **dnsx**. The reason for choosing one tool was pragmatic: most active enumeration tools perform similar DNS probing (the main differences are performance and feature set), so I selected dnsx for its simplicity and resolver options. In active enumeration, the choice of wordlist typically has a greater impact on results than the specific tool; for wordlists I used entries from SecLists.

I document below the wordlists used, rationale, and performance considerations:

- SecLists/Discovery/DNS/subdomains-top1million-110000.txt — large, broad list used against the base domain to maximise discovery.
- SecLists/Discovery/DNS/subdomains-top1million-5000.txt — smaller list used when enumerating under each discovered subdomain to limit total query volume.
- Performance notes: applying large wordlists across many parent hosts produces a combinatorial explosion of DNS queries;

2.2 Commands

```
# Active enumeration against the base domain

dnsx -d chime.com -silent -w /SecLists/Discovery/DNS/subdomains-top1million-110000.txt > dnsx_out.txt

# Active enumeration against each discovered subdomain (brute-forcing children) (high query volume)

dnsx -d passive_subs.txt -silent -w SecLists/Discovery/DNS/subdomains-top1million-5000.txt > dnsx_out1.txt

# Merge and deduplicate into passive_subs.txt

cat dnsx_out.txt dnsx_out1.txt | sort -u > active_subs.txt
```

Notes on the commands above:

- The `-silent` flag reduces noisy output and the results were appended to output files for later analysis.
- Running the 110k-wordlist against the base domain produced many DNS requests (on the order of 110,000 queries) and took a noticeable amount of time; running the 5k-wordlist across the 277 subdomains found earlier resulted in roughly 1.385 million queries and took several hours.
- When doing large active scans, it seems better to: use a fast, reliable resolver pool (resolvers.txt), implement rate limiting, and document your scan window and permissions.

2.3 Results and analysis

I compared the dnsx outputs with the passive results using the same `stats.py` script used previously.

Example command invocations:

```
python3 stats.py dnsx_out.txt subs.txt
python3 stats.py dnsx_out1.txt subs.txt
python3 stats.py dnsx_out.txt dnsx_out1.txt
```

Summary of findings (condensed):

- `dnsx_out.txt` (base-domain run) returned 32 subdomains, of which 13 were new (not present in `subs.txt`).
- `dnsx_out1.txt` (children-of-subdomains run) returned 26 subdomains, of which 10 were new.
- Intersection between the two dnsx runs:

  Found by both runs: `nautilus.chime.com`, `dds.chime.com`, `kyc.chime.com`, `wireless.chime.com`, `tags.chime.com`

  Only in the base-domain run: `persona.chime.com`, `cde.chime.com`, `mcd.chime.com`, `notifications.chime.com`, `beacon.chime.com`, `lending.chime.com`, `atomic.chime.com`, `payday.chime.com`

Only in the children-of-subdomains run: `www.workdayxchime.com`, `compass.chime.com`, `access.chime.com`, `bounce.updates.chime.com`, `bounce.accounts.chime.com`

Observations:

- The base-domain brute force found slightly more unique subdomains overall, but probing subdomains-for-children also yielded unique entries that the base run missed. Each approach can be complementary.
- The additional unique hosts discovered by the children-of-subdomains run included some suspicious-looking names (for example, `www.workdayxchime.com`), which may be misconfigurations, CNAMEs, or external services. These should be validated (resolve, check TLS certificate CN/SANs, and inspect HTTP responses) before concluding they are in-scope assets.
- Given the high query count and limited additional value gained from the children-of-subdomains pass in this case, I concluded that it is better deprioritising exhaustive children enumeration unless the base-domain results have been exhausted and more depth is clearly needed.

3 HTTP(S) service discovery

3.1 Tools

For this exercise I only used httpx. Before using it however I searched for out of scope domains on bugcrowd (https://bugcrowd.com/engagements/chime), copied them into a file called `out_of_scope.txt`, and made sure I didn't target them.

3.2 Commands (examples)

```
1
2  # Merge and deduplicate passive_subs.txt and active_subs into subs.
       txt
3
4  cat passive_subs.txt active_subs.txt | sort -u > subs.txt
5
6  # Remove out-of-scope (I created a small python script for that)
7  # The output valid domains are written to the file "subs_in_scope.
       txt"
8
9  python3 remove_out_of_scope.py subs.txt out_of_scope.txt
10
11 # Probe with httpx (outputs urls.txt)
12
13 httpx -silent -l subs_in_scope.txt -no-fallback -status-code > urls.
       txt
```

4 URL scanning (detailed analysis)

For this exercice I had to choose one url from the file `urls.txt` created on the previous section. To do so, I followed this steps:

1. Eliminate non-useful responses

- **301 / 307 / 308 (redirects):** Usually just point to canonical domains (e.g. www.chime.com).
- **403 Forbidden:** Can be useful for fuzzing, but less ideal for initial analysis.
- **400 Bad Request:** Often unhelpful (may indicate incomplete config or wrong host header).
- **404 Not Found:** Possible for hidden content, but not good for first choice.

2. Focus on live (200 OK) responses
   These are most promising for service and tech fingerprinting:

   - https://flashpaper-dev3.chime.com [200]
   - https://docs-dev3.chime.com [200]
   - https://flashpaper.chime.com [200]
   - https://flashpaper-dev1.chime.com [200]
   - https://chime-com-dev1.chime.com [200]
   - https://flashpaper-qa.chime.com [200]
   - https://flashpaper-dev4.chime.com [200]
   - https://vpn.chime.com [200]
   - https://wp-dev2.chime.com [200]

3. Potencial analysis

   - **Internal/Dev/QA targets** are often misconfigured and leak more than production.
   - **Docs sites** sometimes expose APIs, Swagger, or internal endpoints.
   - **VPN portals** can leak version info, auth endpoints, or be vulnerable to auth bypass.
   - **Flashpaper** sounds like an internal tool/product, possibly admin dashboard. High chance of findings.
   - **WordPress dev** (wp-dev2). WordPress is notorious for vulnerabilities and may have leftover plugins.

4. Final choice
   I choose the url https://wp-dev2.chime.com. The reasons are:

   - Dev environment (likely weaker security).
   - Built with WordPress (high probability for vunerabilities).
   - Good chance of endpoints, debug info, or exposed APIs.

4.1 Top 5 Identified Technologies

For this I used both the Wappalyzer browser extension and nuclei CLI tool. I started by using Wappalyzer, and, although it showed me a lot of technologies being used on the website, it also didn't give me a lot of information about the version of each one. So I tried also using nuclei and run the command:

- nuclei -u ̈https://wp-dev2.chime.com ̈-tags tech

The scan found multiple technologies, mostly WordPress plugins plus infrastructure. After analysing the results, I would say the top 5 findings are the following:

1. **WordPress (CMS)**

   - Detected by: **nuclei** & **Wappalyzer**

- Multiple plugins detected (Classic Editor, SEO, Super Cache, etc.), some with outdated versions.
- Core CMS platform used to serve the website.

2. **PHP (Language)**
   - Detected by: **Wappalyzer**
   - Backend language powering WordPress.
   - No version information available.
3. **MySQL (Database)**
   - Detected by: **Wappalyzer**
   - Stores site content, user data, and configuration.
   - No version information available.
4. **Cloudflare (CDN/WAF)**
   - Detected by: **nuclei** & **Wappalyzer**
   - Provides caching, content delivery network, and web application firewall.
   - Masks the origin server IP.
5. **Matrix (Messaging Protocol)**
   - Detected by: **nuclei**
   - Integration with external Matrix homeserver (`chime.ems.host`).
   - Used for decentralized real-time messaging and federation.

4.2 Content discovery

For this exercise I used ffuf and the wordlist common.txt from SecLists:

Unfortnately, while trying to use ffuf, curl and httpx, it seems I was blocked by their cloudflare security. As such, I continued using a different subdomain that is also interesting, that being `https://flashpaper-dev3.chime.com`.

```
1
2  # Get error size to filter on ffuf command
3  curl -s -o /dev/null -w "%{http_code} %{size_download}\n" https://
       flashpaper-dev3.chime.com/nonexistent-abc123
4
5  # Quick, low-noise probe (Need to replace <404SIZE> with the value
       got on the first command, in my case was 19)
6  # Changed to only use 5 threads to not alert cloudflare
7  ffuf -u https://flashpaper-dev3.chime.com/FUZZ \
8       -w ~/SecLists/Discovery/Web-Content/common.txt \
9       -e .php,.html,.htm,.txt,.js \
10      -t 5 \
11      -mc 200,301,302,403 \
12      -fs <404SIZE> \
13      -o ffuf-quick.json -of json
```

The scan wielded multiple results, mostly with http status code 403. From the results, there were only found 4 endpoints with status code 200, those being:

- https://flashpaper-dev3.chime.com/add

- https://flashpaper-dev3.chime.com/favicon.ico
- https://flashpaper-dev3.chime.com/ping
- https://flashpaper-dev3.chime.com/.well-known/http-opportunistic

None of them being particularly useful...