ChatGPT

OWASP Amass 4.2.0 CLI Guide (Kali Linux & Ubuntu)

Installation on Kali Linux

Kali Linux includes OWASP Amass in its package repository (version 4.2.0 is available) [1]. You can install it via APT:

```
sudo apt update && sudo apt install amass
```

This will install the Amass binary (and an `amass-common` dependency) on Kali [2] [3]. Once installed, verify by checking the version:

```
amass -version
```

Kali may pre-install Amass as part of certain metapackages (e.g. the `kali-linux-default` set), but running the above command ensures you have it. The Kali package tracker confirms the version as **4.2.0-0kali1**, which matches this guide [1].

> **Note:** If you prefer to use the latest development version or Kali's package is outdated, you can build from source. Ensure Go is installed, then run `go install -v github.com/owasp-amass/amass/v4/...@master` [4]. This compiles Amass from the official repository. However, for most users the Kali package is sufficient and easier to manage.

Installation on Ubuntu

Ubuntu's official repositories do *not* include Amass by default. The simplest way to install Amass 4.2.0+ on Ubuntu is via the Snap package:

```
sudo snap install amass
```

This Snap is published by the Amass team and bundles all dependencies [5] [6]. After installation, you can run `amass` from the terminal. If the Snap installation fails (e.g. due to Snap issues), you have a couple of alternatives:

- **Using APT (with Kali repo):** Not recommended for most users, but you can add the Kali Linux repository and install Amass via apt on Ubuntu [7]. This can lead to repository conflicts, so proceed with caution.

- **Building from source:** Install Go (Golang) via `sudo apt install golang`, set your `$GOPATH`, then run `go install -v github.com/owasp-amass/amass/v4/...@master`. This will place the `amass` binary in your `$GOPATH/bin` (e.g. `~/go/bin/amass`) [4].

After installing, test with `amass -h` to see the help menu and ensure the tool runs. The help menu will show available subcommands: `intel`, `enum`, and `db` in Amass v4 [8]. (Note: Older subcommands like `track` and `viz` have been removed in Amass v4 [9] [10]—functionality like tracking differences is now achieved through the persistent graph database and the `amass db` commands.)

Passive vs. Active Scanning Modes

Amass can run in **passive** or **active** mode, each suited for different scenarios. By **default, Amass 4.2.0 runs in passive mode** if you don't specify otherwise [11]. Below is an overview of each mode:

- **Passive Mode:** Uses only third-party data sources (OSINT) and **does not directly probe** the target's infrastructure. It's faster and quieter because it avoids brute-force DNS queries or direct interactions with target systems [12] [12]. In passive mode, Amass will **not even resolve** discovered subdomains or perform DNS validations [12]. It simply gathers names from APIs, web archives, search engines, etc. Techniques like DNS brute-forcing and name alterations are disabled in passive mode [12]. Use passive mode when you need quick results or must avoid any active footprint (e.g. during "stealth" reconnaissance or when you are not authorized to actively scan).

- **Active Mode:** Enables **direct enumeration techniques** against the target's DNS and network infrastructure for more comprehensive coverage. In active mode, Amass will perform DNS resolution on discovered names, attempt zone transfers, crawl websites, pull SSL/TLS certificates from target servers, and run DNS brute-force permutations (if enabled) [13] [13]. Active mode can uncover additional subdomains (e.g. via brute force and certificate scraping) and validate which ones are truly in use. However, it is **slower and more likely to be detected**, since it generates DNS queries and web requests to the target domain. Use active mode when you have permission for full engagement and after exhausting passive results. Always ensure you're authorized before using active techniques like zone transfers or port scans [13].

When to use each: A good workflow is to start with a passive scan to quickly gather as many known subdomains as possible, then move to an active scan to find additional subdomains and validate results [11] [14]. Passive mode gives a broad historical view (including subdomains that might no longer be active), whereas active mode gives a current view of what's resolvable and live. Keep in mind that passive results may include false positives or stale data (since no validation is done), and active results may miss items that require deeper historical data. In practice, combining both yields the best coverage.

Configuring API Integrations (Data Source APIs)

One of Amass's strengths is aggregating data from dozens of sources (OSINT databases, DNS services, etc.) [15]. Many of these sources require API keys or credentials to unlock their full potential. Amass 4.2.0 uses a YAML configuration file to manage these API keys.

Common API data sources that benefit from keys include:

- **Shodan** – IoT search engine that can provide subdomains via DNS lookups (requires API key) [16].
- **SecurityTrails** – DNS and historical data (requires API key, free community keys available) [16].
- **VirusTotal** – Passive DNS database from uploaded files (requires API key for meaningful use) [16].
- **PassiveTotal (RiskIQ)** – Extensive passive DNS (API key/secret required).
- **Censys** – Certificate search for subdomains (API ID/secret required for expanded queries).
- **Others**: *AlienVault OTX*, *BinaryEdge*, *Spyse/Feed (CertDB)*, *DNSDB (Farsight)*, *WhoisXML*, *Twitter*, *Umbrella*, *Chaos*, etc. (Amass supports many; run `amass enum -list` to see all sources) [17].

Amass will still run without these API keys, but **its coverage will be limited** to the free and open sources. For example, without a Shodan API key, Amass cannot use Shodan's database; without a VirusTotal key, Amass gets only very limited data from VirusTotal, etc. It's highly recommended to configure as many API keys as possible (even free-tier keys) to maximize Amass's capabilities [18]. You can identify which data sources require keys by running `amass enum -list` – sources marked with a `*` in the list need configuration [19].

How to configure API keys: Amass v4.2.0 stores API credentials in a YAML file (separate from the main config). By default this file is named `datasources.yaml`. You will populate this file with your API keys for various services. In section Setup config.yaml and datasources.yaml, we provide detailed steps and examples of this file. In short, each entry in `datasources.yaml` specifies a data source by name and the credentials (e.g. API key, secret, username/password) needed for that source.

For example, an entry for Shodan in `datasources.yaml` might look like:

```
datasources:
  - name: Shodan
    creds:
      account:
        apikey: "<YOUR_SHODAN_API_KEY>"
```

Similarly, SecurityTrails and others will have an entry with their API key. A more complete example is shown below in the `datasources.yaml` setup. Once your keys are in place, Amass will automatically use them during scans – you'll notice significantly more results from sources that were previously "locked" without keys [19].

Setting Up config.yaml and datasources.yaml

Amass 4 moved from a single config file (v3's `config.ini`) to **YAML-based configuration**. There are two main files to be aware of:

- `config.yaml` – Main configuration file for Amass (scanning settings, scope, options).
- `datasources.yaml` – File dedicated to API credentials for data sources.

File Locations and Initialization

After installing Amass, these files are not automatically created. You need to create them (or copy the examples provided by Amass) in the correct location. The default search paths for `config.yaml` on Linux are [20] [21]:

- `$XDG_CONFIG_HOME/amass/config.yaml` (if XDG_CONFIG_HOME is set), or
- `$HOME/.config/amass/config.yaml`, or
- `/etc/amass/config.yaml`.

For most users, `$HOME/.config/amass/` is the place to put your config. Create the directory and files with proper permissions:

```
mkdir -p ~/.config/amass
# Copy example files (if available) or create new ones
nano ~/.config/amass/config.yaml
nano ~/.config/amass/datasources.yaml
```

Amass provides example config files on its GitHub. If you installed via Kali or Snap, check if example files were included (for example, Kali places samples under `/usr/share/doc/amass/examples/`). If not, you can find the official examples online: **config.yaml** and **datasources.yaml** in the Amass repository [22] [23]. Copy those as a starting point.

According to users, **Amass v4 will not automatically detect your config file unless it's in the right place or explicitly specified** [24] [25]. If you find Amass isn't picking up your settings, you may need to use the `-config` flag to point to it every run (or ensure you placed it in `~/.config/amass/` exactly). One community tip confirms: *"They changed the config file and how it is set up: you have to copy the datasources.yaml and configure your API keys there, then copy the new config file from the examples and reference the datasources file in it"* [25]. We'll do exactly that below.

Configuring `datasources.yaml` (API Keys File)

The `datasources.yaml` file holds your API keys. Using a YAML list, you specify each data source by name, with credentials nested under it. Here's an **example snippet** with a few common sources (Shodan, VirusTotal, SecurityTrails) using the format from Amass 4.2.0 [26] [27]:

```
datasources:
  - name: Shodan
    creds:
      account:
        apikey: "<SHODAN_API_KEY>"
  - name: VirusTotal
    creds:
      account:
        apikey: "<VIRUSTOTAL_API_KEY>"
```

```
  - name: SecurityTrails
    creds:
      account:
        apikey: "<SECURITYTRAILS_API_KEY>"
global_options:
  minimum_ttl: 1440
```

A few notes on this structure:

- The file begins with `datasources:` and then a list of entries. Each entry has a `name` matching a data source (exact names can be seen with `amass enum -list`).
- Under `creds`, there may be categories like `account`, `api`, etc., depending on what the source expects. For most, an `account` with an `apikey` is enough. Some sources might require multiple fields (for example, an API key and a "secret" or username/password). The Amass documentation indicates the possible fields are `apikey`, `secret`, `username`, `password` as needed [28]. For instance, **Censys** requires both an API ID (username) and API secret.
- `global_options.minimum_ttl` in the example above is optional – it sets a cache duration (in minutes) for how long Amass will reuse data source results [29]. In this case `1440` minutes = 1 day cache. You can include it or omit it; by default, Amass has a caching mechanism for API queries to avoid repeating the same queries too frequently.

Fill in your keys between the quotes. If you don't have a particular service's API key, you can omit that entry. **Only include sources you have keys for or plan to use.** (Amass will skip those without credentials or with null keys.)

After editing, double-check YAML indentation and formatting. A common pitfall is indentation errors or using tabs – ensure you use spaces as in the example.

Configuring `config.yaml` (Main Config)

The `config.yaml` ties everything together and defines how Amass runs. For basic usage, you need to at least point it to your `datasources.yaml`. A minimal config might look like:

```
options:
  datasources: "/home/<youruser>/.config/amass/datasources.yaml"
```

This `options.datasources` setting tells Amass where to read the API keys [30]. You can use an absolute path (recommended). If the datasources file is in the same directory as config, you could also use a relative path or just `"datasources.yaml"` [31] [32], but absolute path avoids ambiguity.

Beyond that, `config.yaml` can include many sections:

- **mode**: You can set `mode: passive` or `mode: active` under the default section to force a mode, but generally it's easier to use the `-passive` / `-active` CLI flags as needed. If unspecified, Amass uses its default (passive) mode [11].

- **scope**: Define what is in-scope for enumeration. You can list root domains under `scope.domains`, specify IP ranges, ASNs, etc. [33] [34]. For example:

```
scope:
  domains:
    - example.com
    - example.org
  ips:
    - 192.0.2.0/24
  asns:
    - 13374
  blacklisted:
    - subdomain.ignoreme.example.com
```

  This ensures Amass focuses only on the domains/IPs/ASNs you specify and ignores blacklisted subdomains [34] [35]. If you run Amass with the `-d` flag for a domain, specifying scope in config is optional; but it can be useful for project-specific config files (e.g. list all domains of your target program).
- **resolvers**: Custom DNS resolvers for queries. Amass has a default list, but you can override by providing a list of resolver IPs under `options.resolvers` [31] [36]. For example:

```
options:
  resolvers:
    - 1.1.1.1
    - 1.0.0.1
    - 8.8.8.8
```

  Using reliable resolvers (or even running your own DNS resolver) can speed up active enumeration.
- **bruteforce settings**: You can enable DNS brute forcing by default via config:

```
bruteforce:
  enabled: true
  recursive: true
  wordlists:
    - "./wordlists/subdomains.txt"
  minimum_for_recursive: 2
```

  This corresponds to Amass's brute-force options (the `-brute` flag in older usage). `recursive: true` means it will brute-force not just the root domain but also newly found subdomains (if they have at least 2 new sub-subdomains discovered, in this example) [37]. You can specify custom wordlists here (or use Amass's default wordlist if not specified). Amass comes with some example wordlists (like `deepmagic.com_top50kprefixes.txt` for permutations and others [38] [39]).
- **alterations (mutations)**: Configuration for generating permutations/alterations of discovered names. For instance, Amass can swap, add, or mutate letters in known subdomains (these are "active" techniques). In config, `alterations.enabled: true` and providing a wordlist of

common words can improve discovery [39]. Additional toggles like `flip_words`, `add_numbers`, etc., exist with default behaviors [40] [41] (these are advanced and typically you can rely on defaults).
- **output directory**: You can set a default output directory in config with `output_directory: /path/to/dir` in the default section [42]. However, using the `-dir` flag at run-time often makes this unnecessary (the flag will override config).

Linking the files: Ensure that in `config.yaml` under `options`, the `datasources` path is correct [30]. A common mistake is copying the example which might have a placeholder path (e.g. `"/root/.config/amass/datasources.yaml"` as in one example [32]) that doesn't match your user. Update it to your actual home directory path. Once both files are ready, you can test by running Amass with your config:

```
amass enum -config ~/.config/amass/config.yaml -list
```

This lists data sources and should show no errors. If your config is being read properly, you will see the list and any sources with missing API keys marked with `*`. If you still see `*` next to sources you configured, double-check the names and indentation in `datasources.yaml`. Also watch for any startup error messages – Amass will report parsing errors (e.g. "failed to load configuration file") if your YAML has syntax issues.

Tip: You can tell Amass to use your config without specifying `-config` every time by placing it in the default location (`~/.config/amass/config.yaml`) as mentioned. Amass will auto-load it if found [20] [21]. However, if you maintain multiple configs (say one global and one per project), you'll likely be using the `-config` flag or switching the files in and out.

Running Common Scan Types with Amass

With installation and configuration done, let's cover how to execute common enumeration tasks with Amass's CLI. The primary subcommand we use is `amass enum`, which performs DNS enumeration and saves results to Amass's graph database (and optionally output files) [43] [44]. We'll also mention `amass intel` briefly for completeness.

1. Passive Subdomain Scan (Domain Enumeration)

To run a **passive scan** for subdomains of a domain (using only OSINT sources and no direct DNS brute force), use the `-passive` flag. For example:

```
amass enum -passive -d example.com -o output/example_passive.txt -log output/
example_passive.log
```

Explanation:

- `amass enum` – invokes the enumeration mode.

- `-passive` – ensures Amass does not do active techniques [12]. It will query all configured data sources (APIs, web scraping, etc.) but *will not* perform brute forcing or DNS resolution on the found names.
- `-d example.com` – the target domain (you can specify multiple `-d` flags or use `-df <file>` to read from a list of domains).
- `-o output/example_passive.txt` – writes the discovered subdomains to the specified file (in addition to printing them on screen) [45]. The output file will contain subdomain names (and possibly IP addresses, see notes on output format below).
- `-log output/example_passive.log` – saves error messages and debug info to a log file [46]. This is useful to catch issues (like a data source failing or rate-limiting you) without cluttering the console.

Running this command will quickly fetch results from Amass's passive sources. Because it doesn't spend time validating each subdomain or doing brute force, it's usually faster. You might see output streaming to the console as Amass finds subdomains. Each line typically includes the domain name and maybe associated data. For example, you might see lines like:

```
[ThreatCrowd]   sub.example.com
[CertSpotter]   another.example.com
```

If you used `-o`, check the `example_passive.txt` after completion. Passive mode might list subdomains without IP addresses by default (since it didn't resolve them). If the output file contains more than just domain names (e.g. IPs), we'll address that under *Interpreting Output*.

When to stop: By default, Amass will keep querying until it has exhausted all sources or hit your configured TTL cache limits. It won't run indefinitely, but if you want to absolutely cap the run time, use the `-timeout` flag. For example, `-timeout 60` would stop the enumeration after 60 minutes [47]. In passive mode it's usually not necessary to set a timeout unless you have dozens of domains or very slow sources.

2. Active Subdomain Scan (with DNS brute forcing & scraping)

For an **active scan** that includes brute-force and additional techniques, use the `-active` flag. For example:

```
amass enum -active -d example.com -brute -min-for-recursive 2 \
  -o output/example_active.txt -log output/example_active.log \
  -config ~/.config/amass/config.yaml -dir output/amass_db
```

Explanation:

- `-active` – turns on active recon methods [13], such as DNS resolution of found names, certificate collection, and others. This flag by itself doesn't enable brute forcing, it just permits active methods.
- `-brute` – (optional) explicitly enable DNS brute forcing of subdomains. In Amass v4, brute forcing can also be enabled via config (as shown earlier) or by this flag. We include it here to ensure brute

force runs. Brute forcing will try common subdomain names (from wordlists) to find ones not present in OSINT data [48].

- `-min-for-recursive 2` – (optional) sets the threshold for recursive brute forcing. This means if a discovered subdomain itself has at least 2 sub-subdomains discovered, Amass will brute force further down that name [37]. This helps in deeply nested subdomains. We set `2` as an example; adjust as needed or omit for default.
- *Other flags:* `-o` and `-log` as before to capture output and errors. We also show `-config` to ensure Amass uses our configuration (API keys, wordlists, scope, etc.) and `-dir` to specify an output directory for the graph database (more on that shortly).

Active scans will take longer. Amass will resolve each found subdomain to verify it exists and gather its IP address(es). It will also proactively find new names by: - Brute force (trying dictionary words as subdomains, if `-brute` or config enabled). - Zone transfer attempts (if the DNS servers allow it). - TLS certificate parsing: If Amass finds an IP address (from DNS records or provided via `-addr` flags), it may fetch the SSL certificate from common ports (443, etc.) to scrape additional subdomain names (this happens when `-active` is set) [13]. - Web scraping: Amass may crawl certain websites for URLs that contain the target domain names (depending on data sources).

Because of this, active mode generates many DNS queries and possibly HTTP(s) requests. It's good practice to **limit the load** to avoid overloading DNS or getting throttled: - Use the `-max-dns-queries <N>` flag to set concurrency of DNS queries [49]. For example, `-max-dns-queries 5` will only allow 5 DNS lookups at a time. - Ensure you have a reasonable wordlist for brute force. Amass's default is often extensive; if you want to be lighter, use a smaller wordlist or tune `minimum_for_recursive` higher to reduce deep brute forcing. - Set a `-timeout` to cap run time (e.g. `-timeout 120` for 2 hours) so it doesn't run forever on large scopes [47].

Output directory (`-dir`): In the above command we used `-dir output/amass_db`. This causes Amass to store its graph database and other output files in `output/amass_db` directory [50] [51]. The first time you run, Amass will create this directory and a database file inside (usually named like `amass.graphdb` or similar). Reusing the same `-dir` on subsequent runs is important for the **tracking** feature – Amass will merge new findings with old and avoid re-discovering the same subdomains repeatedly [52] [53]. If you run another `amass enum` on *example.com* with the same `-dir`, it will recognize previously found subdomains and even re-check their DNS to see if anything changed [52]. This is great for continuous monitoring. (If you prefer each run totally separate, use a fresh `-dir` or clear the directory between runs.)

3. Limiting Scan Duration and Resource Usage

As mentioned, use `-timeout <minutes>` to stop an enumeration after a certain time [47]. For example, `-timeout 30` ensures Amass quits after 30 minutes, even if it hasn't finished exploring all data sources. This is useful in automated scripts or CI pipelines where you need to bound execution time.

To control network and CPU usage, adjust: - **DNS query rate:** `-max-dns-queries` flag. Amass defaults this to a high number (it tries to be efficient), but you can lower it. Setting `-max-dns-queries 1` will serialize all DNS lookups (very slow but minimal impact on networks), whereas a higher number (like 100) speeds up at the cost of more bandwidth and potential noisy traffic [49]. - **Number of concurrent data sources:** There isn't a direct flag for this, but Amass will query many APIs in parallel. If you find this overwhelms your network or API rate limits, consider disabling some sources or running with a smaller

scope. You can exclude specific sources with `-exclude <sourceName>` or include only some with `-include <sourceName>` flags [54] [55]. - **Logging level:** By default Amass prints info-level messages. If you want less console output, use `-silent` to suppress all output (only results will be shown) [56], or `-v` for verbose if debugging.

4. Other Commands – Amass Intel (Briefly)

While `amass enum` is the main workhorse for finding subdomains, `amass intel` can be useful to discover top-level domains and netblocks related to your target. For example:

```
amass intel -org "Example Corp" -whois
```

This uses Amass's passive intelligence gathering to find domains owned by *Example Corp* (via reverse WHOIS, etc.) [57] [58]. It's a way to find additional root domains beyond the one you know. `amass intel` respects the API keys in your config for things like WHOIS lookup services [58]. If you have a broad target scope (like a whole company), run `intel` first to get all domain names, then feed those into `enum`. For single-domain bug bounty programs, `intel` might not be needed.

Saving and Organizing Output

Organizing Amass output is important for analysis. We've already touched on some options (`-o`, `-log`, `-dir`). Let's summarize best practices for output management:

- **Use the `-o` option** to save discovered subdomains (and related info) to a text file [45]. This captures what you see in the terminal. For example, `-o amass_out.txt` will create `amass_out.txt` with the results of that run.
- **Use the `-log` option** to capture errors, warnings, and debugging info [46]. For example, `-log amass_errors.log` will record things like network timeouts, data source failures, etc. This is extremely helpful for troubleshooting (e.g., if a particular API is failing due to a bad key or quota, you'll see it in the log).
- **Always specify an output directory (`-dir`) for multi-run projects.** By using a dedicated directory per target or engagement, you keep the graph database and any auxiliary files separate from other projects. For instance, for a program "Alpha", use `-dir ~/recon/alpha/amass_data`. This directory will contain the persistent database of findings for Alpha. As you run Amass again for Alpha, use the same `-dir` to update that database. You can periodically copy or backup this directory if needed.
- **One-output-per-run vs. single-accumulated output:** If you append results to the same file over multiple runs, you may end up with duplicates. It's often cleaner to output to a new file each run (perhaps timestamped) and then diff them or use `amass db` to analyze changes. The persistent graph database is how Amass itself tracks changes. You can leverage `amass db` commands to list or compare results (e.g., `amass db -dir <dir> -names -d example.com` will list all unique subdomains found in the DB for that domain). The user guide notes that if you maintain the same output directory, Amass will automatically leverage previous findings to avoid redundant work [52] [53].

- **Format of output:** By default, the text output (`-o`) may include more than just subdomain names. Amass v4 sometimes prints lines showing relationships (like subdomain -> IP, netblock, ASN) rather than a simple list [10] [59]. If you want a clean list of subdomains, you have a few options:
- Use `amass db` after the run: `amass db -dir <output_dir> -names -d example.com > subdomains.txt` will pull all FQDNs from the database for that domain, one per line [60] [61].
- Use `grep` or other text processing on the output file (e.g., filter lines that end with `(FQDN)` or similar).
- Consider using the JSON output feature: `-json results.json` will save results in JSON format, including rich data about each finding. You can then parse this JSON to extract what you need (subdomains, IPs, etc.). JSON output is very useful if you plan to feed results into another tool or script.
- **Database use for tracking:** If you run periodic scans, you can compare the current run's output to the previous run's output to see new vs. gone subdomains. The built-in way is using the `amass db` subcommand with the `-show` and `-df` flags (though this can be a bit complex to use directly). A simpler approach is to maintain sorted lists of subdomains from each run and use `diff` or a version control system to track changes. Because Amass's output includes IPs and netblocks, you may want to filter those out when doing comparisons, focusing on the FQDNs.

In summary, **save your outputs in an organized folder structure**. For example:

```
recon/
  └── example.com/
      ├── amass_db/        # -dir for database
      ├── amass_passive_2025-05-24.txt   # output from passive scan
      ├── amass_active_2025-05-25.txt    # output from active scan next day
      ├── amass_passive_2025-05-24.log   # log file for passive scan
      └── amass_active_2025-05-25.log    # log file for active scan
```

This way, you have a clear record of what was found, when, and you can easily refer back or compare.

Interpreting Amass Output

Amass output can be verbose. Understanding the results is key to incorporating them into your workflow.

When Amass finishes running (or as it runs), you'll see output lines. In **Amass v4**, these lines often represent a graph relationship as discovered in the enumeration. For example, you might see something like:

```
example.com (FQDN) --> ns_record --> ns1.example.com (FQDN)
ns1.example.com (FQDN) --> a_record --> 192.0.2.123 (IPAddress)
192.0.2.0/24 (Netblock) --> contains --> 192.0.2.123 (IPAddress)
13335 (ASN) --> announces --> 192.0.2.0/24 (Netblock)
```

This format is showing that `ns1.example.com` is a nameserver for `example.com`, that `ns1.example.com` resolved to IP `192.0.2.123`, and that IP belongs to a certain netblock and ASN [10]

[59]. Amass's integration of the *Open Asset Model* means it's mapping not just subdomains, but how they connect to IPs, netblocks, and ASNs.

For most subdomain enumeration purposes, you care primarily about the discovered **FQDNs (fully qualified domain names)**. Those are the entries labeled `(FQDN)`. In the above example, `ns1.example.com` is a discovered subdomain of interest. The IP and ASN info is supplementary. If you used `-o amass_out.txt`, check that file: - It may contain lines similar to the above graph format. - Or, depending on recent updates, it might contain simpler entries (e.g. some users reported older versions printed one subdomain per line, and v4 initially printed the graph format by default).

How to extract subdomains only: If your output file or console shows the graph style, you can filter it. All subdomains will appear as `(FQDN)` in the lines. A quick way:

```
grep "(FQDN)" amass_out.txt | cut -d ' ' -f1 | sort -u > subdomains_only.txt
```

This takes lines with "(FQDN)", then cuts the first field (which is the subdomain at beginning of the line), then unique-sorts them. You may need to adjust the cut delimiter depending on the exact format.

Alternatively, use the **Amass DB**:

```
amass db -dir output/amass_db -names -d example.com
```

This will print a clean list of all domain names discovered (no IPs, no metadata) [60] [61]. You can redirect that to a file. The `-names` flag extracts just the names. If you want the IP addresses associated, you could use `-show -d example.com -ip` to list names with their IPs [61].

In your output, you might also see **source tags** (especially if using `-src`). For example:

```
[CertSpotter] beta.example.com - 203.0.113.5
```

This indicates CertSpotter (a CT log source) found `beta.example.com` which resolved to 203.0.113.5. The `-src` flag will include the data source name in brackets [62]. It's useful for validation (knowing where a finding came from) but can be noisy. It's off by default in Amass v4 unless you specifically enable it.

DNS records: Amass primarily finds **A/AAAA records** (addresses) and occasionally NS/MX/CNAME relationships. In the graph output, you saw `ns_record` (for NS) and `a_record`/`aaaa_record`. If you need to see which subdomains were aliases (CNAMEs) or which were discovered via MX, those should appear in the output as well (e.g., `mail.example.com (FQDN) --> mx_record --> example-com.mail.protection.outlook.com (FQDN)`). In practice, if you just need "all subdomains that belong to example.com", you would still consider those CNAME targets if they end in your domain.

Netblocks and ASNs: The output's netblock (`x.x.x.x/24`) and ASN lines inform you of the network space your domain lives in [63] [59]. This is more relevant for network mapping. You can use it to identify if multiple

subdomains fall in the same IP range or if an IP is out-of-range. While not necessary for a basic subdomain list, this info is valuable for comprehensive asset mapping. Amass is essentially telling you "Domain X lives on IP Y, which is in network Z (ASN OrgName)".

Checking the logs: If you specified `-log`, open the log file. Search for "API" or "ERROR". For example, you might see entries like:

```
[SecurityTrails] API key data was not provided
[Shodan] API query limit reached
```

These indicate issues: in this case, either you didn't provide a SecurityTrails key or it wasn't read properly [64]. Use such clues to fix your `datasources.yaml` or manage your API usage.

Best Practices for Modern Recon with Amass

Finally, let's discuss some best practices to get the most out of Amass 4.2.0 in a modern reconnaissance workflow:

- **Combine Passive and Active Wisely:** Start with passive enumeration to gather an initial set of domains quickly and silently [11] [14]. Then switch to active mode to dig deeper. Passive results can be fed as input to active scans (Amass will automatically include previous findings if you use the same directory). This two-phase approach ensures you don't miss historical or non-resolving subdomains, while still finding current live assets.
- **Leverage Amass's Database:** Instead of treating each run as isolated, use the persistent graph database (`-dir` option) to your advantage. Over time, this database becomes a collected knowledge of the target's assets. Amass will reuse and re-check old findings, alerting you to changes (for example, if a subdomain that existed last month no longer resolves, or if a previously seen IP has changed) [52] [53]. For continuous monitoring, you can run Amass periodically (cron job or CI pipeline) with the same output directory and then use `amass db -diff` features or simple diffs on the output to see what's new.
- **Scope Control:** Use the config `scope` settings or command-line flags to constrain Amass to your target. This prevents Amass from following data that might lead outside your intended scope (for example, not enumerating subdomains of a parent company if not allowed, etc.). Blacklist known out-of-scope subdomains (like common CDN domains) if needed [34] [35]. This focuses your scan and reduces noise.
- **API Key Management:** Gather as many API keys as you can for free/community editions of services: Shodan, SecurityTrails, VirusTotal, AlienVault, etc. Even free keys often allow a few queries that Amass can use [19]. Monitor your usage – some services have monthly quotas. If you have multiple accounts or team keys, you can actually configure multiple credentials for the same source in `datasources.yaml` (Amass supports multiple "credential sets" per source) [28] [65]. Amass will cycle through them if one is overused.
- **Update Amass and Sources:** The Amass project is active, and data sources APIs can change. Keep Amass updated to benefit from new sources or fixes. For example, if a source API endpoint changes, an update will fix it. Also, periodically review your `datasources.yaml` against the latest example from the project to catch any new sources you could add or changes in format.

- **Use in Combination with Other Tools:** Amass is powerful but it's not the only tool. In modern recon, people often use Amass alongside tools like Subfinder, assetfinder, etc., to cover any gaps. Each tool has some unique sources. If you integrate Amass in an automation script, consider merging its output with others for completeness. However, note that Amass already covers a **very broad range of sources** (APIs, search engines, cert logs, etc.) [15], so it often finds much of what others do.
- **Noise Management:** Active mode can trigger alerts (DNS alarms at the target, etc.). If you need to be stealthy (e.g., during a bug bounty where you want to avoid detection), stick to passive mode. If you use active mode, limit the speed (`-max-dns-queries`) and maybe target specific subdomains with brute force (you can set `bruteforce.targets` in config to only brute certain subdomains). Also, using public DNS resolvers for brute forcing means your traffic comes from those resolvers to the target, not directly your IP.
- **Interpreting results in context:** After running Amass, you should still verify critical findings manually. Amass might report a subdomain that no longer points to an active host (especially from passive data). Use tools like `dig` or `nslookup` to double-check interesting subdomains. Also, consider feeding the discovered subdomains into port scanners or HTTP probes to see which are live web servers, which have open ports, etc., as part of next steps in recon.
- **Stay aware of known issues:** As of 4.2.0, a known point of confusion was the new output format and the relocation of API keys to `datasources.yaml`. We've covered these changes in this guide. If you come across older tutorials referencing flags like `-ip` or subcommands like `amass track`, remember those either have new meanings or have been deprecated in v4. For example, you don't need `-ip` flag to see IP addresses – Amass v4 includes IP info by default in the graph output [10]. The `amass track` functionality (comparing runs) is now achieved by using the same database and the `amass db` commands to show differences. When in doubt, consult `amass -help` and the official user guide which is updated for v4 [51] [66].

By following this guide, you should be able to install Amass 4.2.0 on Kali or Ubuntu, configure it for maximum data collection (with your API keys), and run both passive and active subdomain enumeration confidently. Amass is a powerful tool – when used with careful configuration and a solid workflow, it becomes a cornerstone of modern reconnaissance, giving you a comprehensive view of your target's external assets. [48] [16]

---

[1] amass - Kali Linux Package Tracker
https://pkg.kali.org/amass

[2] [3] [15] [16] [48] amass | Kali Linux Tools
https://www.kali.org/tools/amass/

[4] Amass: New Config File Update. Bug Bounty Tutorial | by Harshad Shah | Offensive Black Hat Hacking & Security | Medium
https://medium.com/offensive-black-hat-hacking-security/amass-new-config-file-update-e95d09b6eb70

[5] [6] Install amass on Ubuntu using the Snap Store | Snapcraft
https://snapcraft.io/install/amass/ubuntu

[7] How can I install Kali Linux tools on Ubuntu using a command like sudo apt install amass-common - Ask Ubuntu
https://askubuntu.com/questions/1523389/how-can-i-install-kali-linux-tools-on-ubuntu-using-a-command-like-sudo-apt-insta

8 20 21 22 23 28 29 33 34 35 37 40 41 42 43 44 50 51 52 53 56 57 58 65 66 raw.githubusercontent.com
https://raw.githubusercontent.com/OWASP/Amass/master/doc/user_guide.md

9 10 59 63 What is new about AMASS. Amass is a famous discovery tool that... | by Kyrillos Maged | Medium
https://kokomagedd.medium.com/new-era-of-discovery-using-amass-8ee41eed39d1

11 14 17 18 19 26 27 30 Practical amass — How I configure and use amass in my recon flow | by Sam Hilliard | Medium
https://medium.com/@samhilliard/practical-amass-how-i-configure-and-use-amass-in-my-recon-flow-94b8814b9025

12 13 45 46 47 49 54 55 60 61 OWASP Amass Project guide. In-depth Attack Surface Mapping and... | by Andrey Pautov | Medium
https://medium.com/@1200km/owasp-amass-project-guide-94bd55521f91

24 25 Amass v4 config.yaml default location not working : r/bugbounty
https://www.reddit.com/r/bugbounty/comments/1al379n/amass_v4_configyaml_default_location_not_working/

31 32 36 38 39 raw.githubusercontent.com
https://raw.githubusercontent.com/sahil3276/Amass_Config/main/config.yaml

62 How to Use OWASP Amass: An Extensive Tutorial
https://www.dionach.com/how-to-use-owasp-amass-an-extensive-tutorial/

64 API Keys not read from config.ini #192 - owasp-amass/amass - GitHub
https://github.com/OWASP/Amass/issues/192