[![OWASP Logo](https://github.com/owasp-amass/amass/blob/master/images/owasp_logo.png?raw=true) OWASP Amass](https://owasp.org/www-project-amass/) - A Quick Start Tutorial for Amass 4

Assumptions

This guide is written for Linux users.

Introduction

OWASP Amass is an open source security tool which helps security researchers and penetration testers perform reconaissance and map the attack surface of a target network. This tool is designed for penetration testers, auditors, security researchers, CISO/IT managers, and others who have valid reasons for mapping out the external attack surface of an organisation.

Changes to Amass 4

In the beginning, OWASP Amass was a self contained tool that you configured with a single configuration file and ran from the command line. It had sub-commands, a variety of command line parameters, and generated user-defined results in its own SQLite database. Over time the tool gained popularity and its ability to determine attack surfaces expanded.

And there was much rejoicing.

Eventually, the Amass team realized that that it could be more than just a single self contained tool. As a result, Amass 4 is based on the “Open Asset Model” (OAM), which is surrounded by an ecosystem that expands collaboration possibilities and the tool's' capabilities.

Since Amass 4 is built differently than previous versions, it will need to be installed differently. To address these questions, let's start with the Amass GitHub Account.

Changes to Amass GitHub

Because Amass 4 has reorganized its architecture, its GitHub repository reflects these changes. Now different elements of the framework are located in different repositories under the overall `OWASP Amass Project` banner at [OWASP Amass Github](https://github.com/owasp-amass).

The OWASP Amass Project contains the following repositories:

* **open-asset-model**. The results of a community-driven effort to

uniformly describe assets that belong to both organizations and individuals. It defines the assets and their relationships with each other.
* **amass**. The command line tool with installations and usage guides.
* **config**. All the code that parses the new format configuration file.
* **oam-tools**. This repo has a collection of helper tools to convert old config files and extract collected data from the database.
* **homebrew-amass**. All the magic that goes into making a Mac homebrew formula.
* **Resolve**. A DNS Brute forcer.
* **asset-db**. The Database code that supports storing data collected while running the command line tool. It supports either Postgres or SQLite3.
* **engine**. Although empty now it will contain an in-depth attack surface discovery engine with the Open Asset Model.

Each sub-repository has documentation related to the component in the docs folder. It is recommended to review the available documentation. As we quickly attempt to become productive this document will refer to the relevant sections.

For a quick start guide for installation and usage the important directories are amass, oam-tools, and asset-db.

Amass Installation Process

If you need to install the Amass tool yourself, [Amass Install Guide](https://github.com/owasp-amass/amass/blob/master/doc/install.md) will help you either with a docker container or using the package manager of your choice. Some distributions have it preinstalled. Kali Linux is one example but it has the tool (for the amass repository) not a database or oam-tools.

Step 1: Selecting a Database

Amass needs to store what it finds. And before we run any Amass tools we need to define where it will live. Therefore lets start with the database.

While Amass can still install its data in a SQLite database as previously done, the tool can now support a Postgres database to provide better project management. If you have several different targets and you wish some form of compartmentalization without tracking files everywhere, Postgres is the way to go.

The repository for the asset-db (fanfare) `database interaction layer` resides at [asset-db](https://github.com/owasp-amass/asset-db). Within there will be documentation in the *docs* folder.

We could install Postgres on our linux host, but I chose containers because they have been a great addition to our modern lifestyle. You will need to have docker installed (you don't?) and docker-compose, but the installs are out of scope for this guide.

Step 2: Get the docker-compose and .env.local files

First, Clone the asset-db repo or copy the the docker-compose and .env.local files within.

```bash
└─$ git clone https://github.com/owasp-amass/asset-db.git
Cloning into 'asset-db'...
remote: Enumerating objects: 246, done.
remote: Counting objects: 100% (70/70), done.
remote: Compressing objects: 100% (35/35), done.
remote: Total 246 (delta 37), reused 37 (delta 31), pack-reused 176
Receiving objects: 100% (246/246), 85.19 KiB | 1.52 MiB/s, done.
Resolving deltas: 100% (127/127), done.
└─$ cd asset-db
└─$ ls -la
total 180
drwxr-xr-x  8 user user  4096 Oct  2 13:12 .
drwxrwxrwt 17 root     root     20480 Oct  2 13:12 ..
-rw-r--r--  1 user user  4038 Oct  2 13:12 assetdb.go
-rw-r--r--  1 user user 13401 Oct  2 13:12 assetdb_test.go
-rw-r--r--  1 user user   219 Oct  2 13:12 docker-compose.yml
drwxr-xr-x  2 user user  4096 Oct  2 13:12 docs
-rw-r--r--  1 user user    90 Oct  2 13:12 .env.local
drwxr-xr-x  8 user user  4096 Oct  2 13:12 .git
drwxr-xr-x  3 user user  4096 Oct  2 13:12 .github
-rw-r--r--  1 user user   478 Oct  2 13:12 .gitignore
-rw-r--r--  1 user user  1479 Oct  2 13:12 go.mod
-rw-r--r--  1 user user 81881 Oct  2 13:12 go.sum
-rw-r--r--  1 user user 11357 Oct  2 13:12 LICENSE
-rw-r--r--  1 user user   116 Oct  2 13:12 Makefile
drwxr-xr-x  4 user user  4096 Oct  2 13:12 migrations
drwxr-xr-x  2 user user  4096 Oct  2 13:12 repository
drwxr-xr-x  2 user user  4096 Oct  2 13:12 **types**
```

What are the docker-compose.yml file and the .env.local files? The docker-compose.yml file contains the docker instructions for creating the container, references the .env.local file for the Postgres environment, and sets admin user and password for the database.

Step 3: Modify docker-compose to Suit

Modify docker compose file if required. Here, we will only expose the database port to a local address. To prevent the port from being

viewable from outside my computer,I modified ports as below:
```yaml
version: '3'

services:
  postgres:
    container_name: assetdb_postgres
    image: postgres:latest
    restart: always
    env_file: .env.local
    ports:
      - "127.0.0.1:5432:5432"

volumes:
  postgres-db:
    driver: local
```
While this version of docker-compose holds the basics of what we need, it could be more complicated (more detailed network and volume definitions). The environment file contains our Postgres configuration for users, passwords, and databases.

Step 4: Modify .env.local to suit

Modify .env.local to change the admin user and their password.

```
POSTGRES_USER=<database-admin-name>
POSTGRES_PASSWORD=<some-password>
POSTGRES_DB=postgres
```
While I have left database name as “postgres," we could define different environment files for different projects and use different database names as the project evolves. To add databases to the same database server for different projects, we would need to ensure the “pg_trgm” extension if available to it(more on that later). Of course, enter your postgres username and password details in the space provided.

Step 5: Launch Postgres Container

Make sure you have privileges to run docker-compose, then run docker-compose in the same directory as the docker-compose and .env.local files.

```bash
└─$ sudo docker-compose up -d
```

Note that the `-d` flag puts docker-compose in daemon mode

(backgrounds the task). If you want to see the gory details of how the sausage is made, drop the `-d'. Then docker-compose will take over your shell. Press Ctrl-C to halt the process.

Step 6: Confirm Postgres Container Is Listening

Before continuing, view the appropriate port and make sure that the Postgres container is listening:

```bash
└─$ sudo netstat -taupen | grep 5432
tcp   0  0 127.0.0.1:5432   0.0.0.0:*    LISTEN   0    3484470
2100059/docker-prox
```

The netstat command confirms that I am listening on the correct port (5432) and interface (127.0.0.1).

Step 7: Create a Amass Database

After we have verified that that the database server is running, we need to create a database (bag of holding) that will store Amass data. In this case, I will use a tool called **PSQL** to connect and configure that database. First, as mentioned in the [asset-db user guide](https://github.com/owasp-amass/asset-db/blob/master/docs/USER_GUIDE.md) I am going to create the database for my project. You will need to enter the database username and password that you specified in .env.local.

From the command line:
```bash
psql postgres://<database-admin-name>:<some-password>@127.0.0.1:5432
-c "CREATE DATABASE assetdb"
```

Note:
Here we are using the name `assetdb` as used in the guides. Since we may want to compartmentalize my data and create databases for each of my targets, then you would need need to follow these steps for each database that you generate.

Step 8: Set the Timezone

And as per the user guide, I am going to set a timezone for that database.

```bash
psql postgres://postgres:postgres@127.0.0.1:5432 -c "ALTER DATABASE
assetdb SET TIMEZONE to 'UTC'"
```

```

### Step 9: Connect to the Postgres Database

After creating the database and assigning it a timezone, we are going to connect to the database server so we can query and modify settings as well as add any necessary extensions.

**Note**:
If Amass needs to use the `pg_trgm` extension, you must specify the database Amass is to use. In this example we will specify "assetdb", the database I created above.

```bash
psql "host=127.0.0.1 port=5432 user=<database-admin_name>
password=<some-password> dbname=assetdb"
```

Successful connections are presented with the following prompt:

```bash
psql (16.0 (Debian 16.0-2))
Type "help" for help.
assetdb=#
```

If we do not explicitly specify the database in the dbname parameter, then the commands we are about to execute will not apply to our target database and Amass will encounter issues. Note that my prompt “assetdb” implies that we are connected to that specific database on my database server.

Step 10: Confirm pg_trgm Status

We want to determine if the “pg_trgm” extension is available. To list the installed extensions in postgres use `\dx` as described in [psql notes](https://www.commandprompt.com/education/how-to-show-installed-extensions-in-postgresql/). The code snippet below shows the command and its result.

```bash
assetdb=# \dx
                 List of installed extensions
  Name   | Version |   Schema   |         Description
---------+---------+------------+------------------------------
 plpgsql | 1.0     | pg_catalog | PL/pgSQL procedural language
(1 row)
```

The result above tells me that “pg_trgm” is not installed. We could
```

also execute:

```sql
assetdb=# SELECT * FROM pg_extension where extname = 'pg_trgm';
 oid | extname | extowner | extnamespace | extrelocatable | extversion
| extconfig | extcondition
-----+---------+----------+--------------+----------------
+------------+-----------+--------------
(0 rows)
```

Step 10: Install pg_trgm Extension

Since we need to install the extension, we will need to run this as admin or as another less privileged user. In this example I am going to the Postgres server admin.

```sql
assetdb=# CREATE EXTENSION pg_trgm SCHEMA public;
```

Now when I query the server I will see that pg_trgm is installed.
```bash
assetdb=# SELECT * FROM pg_extension where extname = 'pg_trgm';
  oid  | extname | extowner | extnamespace | extrelocatable |
extversion | extconfig | extcondition
-------+---------+----------+--------------+----------------
+------------+-----------+--------------
 16518 | pg_trgm |       10 |         2200 | t              | 1.6
|           |

```

and

```sql
assetdb=# \dx
                                    List of installed extensions
  Name   | Version |   Schema   |
Description
---------+---------+------------
+------------------------------------------------------------------------
 pg_trgm | 1.6     | public     | text similarity measurement and
index searching based on trigrams
 plpgsql | 1.0     | pg_catalog | PL/pgSQL procedural language
```

Step 11. Exit PSQL

We can exit the psql environment with `\q`.

With the database and extension installed we can proceed with configuration.

Data Sources Configuration

Before Amass 4, the tool used data sources listed in an INI file, which contained your API keys for different data sources and held some DNS resolver information and root domain information for queries. Now, Amass 4 has a separate data sources file and a separate configuration file, and are both in `YAML` format.

Fortunately, you will not have to retype all your valuable account and API key information into a new 'YAML' file. The Amass team has created a tool that converts legacy INI file configurations into newer formats. The command [oam_i2y](https://github.com/owasp-amass/oam-tools/blob/master/comprehensive_guide.md#the-oam_i2y-command) is your friend and can create a new data sources file, or a new configuration file, or both. The excellent documentation at the link is all you need.

If you are new to Amass then you ought to create a set of APIs for different targets. It is not explicitly recommended but it is worthwhile for more results.

Advice on Project Configuration

When using Amass 3, I generally did not use the INI file for target configuration. I only really used it for data sources. The fact that the data sources and the configuration file are separated now allows me to have a consistent and evolving set of data sources with a configuration file based on my project targets. Different projects will have a different configuration file with a different target configured. The project configuration file references the data sources file, and this line will probably never change.

A basic Amass 4 configuration file

```yaml
└─# cat testconfig.yaml
scope:
  domains: # domain names to be in scope
    - owasp.org
options:
  datasources: "/home/someuser/.config/amass/datasources.yaml"
  database: "postgres://<db-user>:<db-password>@127.0.0.1:5432/
assetdb" # databases URI to be used when adding entries
```

Let's say I have a client and I wish to determine their attack surface

as part of the gig. Lets also say their company name is “ACME” and their domain is “ACME.com”. Then my project configuration file for running Amass would be:

```yaml
scope:
  domains: # domain names to be in scope
    - acme.com
options:
  datasources: "/home/someuser/.config/amass/datasources.yaml"
  database: "postgres://<db-user>:<db-password>@127.0.0.1:5432/acmedb"
# databases URI to be used when adding entries
```

Then when I ran Amass 4, I would reference this specific config file.

```bash
—# amass enum -config ./acme-amass-config.yaml
```

Of course this can be a more complicated configuration file referencing DNS resolvers and wordlists. But this is a quickstart guide and just show the example below.

Collecting Data for Amass

Now that we have our data sources, a database, and a configuration file for our investigation, we can run Amass to enumerate. To accomplish this I will use the following configuration file.

```yaml
scope:
  domains: # domain names to be in scope
    - owasp.org
  ports: # ports to be used when actively reaching a service
    - 80
    - 443
options:
  resolvers:
    - "/home/username/.config/amass/25resolvers.txt" # array of 1 path or multiple IPs to use as a resolver
  datasources: "/home/username/.config/amass/datasources.yaml" # the file path that will point to the data source configuration
  database: "postgres://dbuser:dbpasswd@127.0.0.1:5432/assetdb?testing=works" # databases URI to be used when adding entries
```

Of course, change usernames and passwords to suit your configuration.

Note here that my target domain is owasp.org. I specified port 80 and

443 if I am actively searching. There is my database and data sources. In addition, I have DNS resolvers specified, which is a list of DNS servers that I know respond so I am not wasting time (creating a resolvers file is out of scope here). And finally, bruteforce and alterations sections are disabled.

Lets start collecting:

```bash
amass enum -config ./target-config.yaml
```

As Amass collects information on the target you should start to see some information displayed. Anything that is not "error-like" is a good sign.

Displaying Target Data

Once the above command completes we can now use the tools in the oam_tools repository to view the data. The excellent documentation at [oam_tools](https://github.com/owasp-amass/oam-tools/blob/master/comprehensive_guide.md#the-oam_subs-command) shows how to use the oam_subs command to extract information from the enumeration. In our case I am going to specify that same configuration file I used for enumeration and and use some additional flags to filter what I want to see.

You may need to clone the repository to install the commands on your system. Below I cloned the repository and changed directory into the cloned directory:

```bash
└─# ./oam_subs -config /directory-for-my-config/target-config.yaml -d
owasp.org -names
mas.owasp.org
na.secureflag.owasp.org
dev.owasp.org
k2._domainkey.owasp.org
   :
   :
```

Wrap Up

This has been a quick start guide to get you up and running. If you have any questions, contact the Amass team via the Discord channel.