Maigret

Release 0.6.1

soxoj

Jun 23, 2026

SECTIONS

1 You may be interested in: 3

1.1 Quick start . . . 3
1.2 Installation . . . 5
1.3 Usage examples . . . 9
1.4 FAQ . . . 10
1.5 Command line options . . . 12
1.6 Features . . . 17
1.7 Philosophy . . . 24
1.8 Supported identifier types . . . 25
1.9 Tags . . . 25
1.10 Development . . . 26
1.11 Library usage . . . 34
1.12 Settings . . . 37
1.13 Tor, I2P, and proxies . . . 41
1.14 Cryptocurrency & Web3 Investigations . . . 43
1.15 Academic & Research Investigations . . . 46

Maigret is an easy-to-use and powerful OSINT tool for collecting a dossier on a person by a username (alias) only.

This is achieved by checking for accounts on a huge number of sites and gathering all the available information from web pages.

The project's main goal — give to OSINT researchers and pentesters a **universal tool** to get maximum information about a person of interest by a username and integrate it with other tools in automatization pipelines.

> **Warning**
>
> **This tool is intended for educational and lawful purposes only.** The developers do not endorse or encourage any illegal activities or misuse of this tool. Regulations regarding the collection and use of personal data vary by country and region, including but not limited to GDPR in the EU, CCPA in the USA, and similar laws worldwide.
>
> It is your sole responsibility to ensure that your use of this tool complies with all applicable laws and regulations in your jurisdiction. Any illegal use of this tool is strictly prohibited, and you are fully accountable for your actions.
>
> The authors and developers of this tool bear no responsibility for any misuse or unlawful activities conducted by its users.

CHAPTER

ONE

YOU MAY BE INTERESTED IN:

- *Quick start*
- *Usage examples*
- *FAQ*
- *Command line options*
- *Features list*
- *Library usage*
- *Tor, I2P, and proxies*

1.1 Quick start

After *installing Maigret*, you can begin searching by providing one or more usernames to look up:

```
maigret username1 username2 ...
```

Maigret will search for accounts with the specified usernames across a vast number of websites. It will provide you with a list of URLs to any discovered accounts, along with relevant information extracted from those profiles.

```
➜  maigret git:(main) ✗ maigret soxoj --html --timeout 5
[-] Starting a search on top 500 sites from the Maigret database...
[!] You can run search by full list of sites with flag `-a`
[*] Checking username soxoj on:
[+] Telegram: https://t.me/soxoj
       ├─fullname: Soxoj
       ├─image: https://cdn4.cdn-telegram.org/file/JtPY01pKiMYT7E3J4hkuugFl4Y4el5H8RfFUjTrTM4
0Ifh1mgsDwrqDJVDOFS4KiFsBEB3fvml8_WXtxJCURi2ifgKIgglAh-XoNimq9ecq93HKgB0DZzOcO1FOktiuCCZGJ9au
NwOFtwfxpWTvIpVLSkvZtW8rr2zhJuO_WGH5RagpPhNnuRHvSigzrcvd7cgjWZ5N3ctv96npSDYMrP6lm5ivHWEivHj89
k3w6rRnYS32T4_ex7Hj57s-rAuO87mGknFv2JGj_F-fTl9hw-xeVm8bTS2E3DvdaB6SBQAwGcEwnoRm17ODYd8bH7dG01
EYVRZbPrpouh4HxBOS9Zw.jpg
       └─bio: OSINT 👉■ soxoj.com,
      @soxoj_insides
      . In case I don't respond, pls remind me or resend the msg
[+] Disqus: https://disqus.com/soxoj
       ├─id: 296653102
       ├─fullname: Soxoj
       ├─disqus_username: soxoj
       ├─reputation: 1.235563
       ├─reputation_label: Average
       ├─following_count: 1
       ├─follower_count: 1
       ├─is_power_contributor: False
       ├─is_anonymous: False
       ├─created_at: 2018-08-30T00:08:14
       ├─upvotes_count: 0
       ├─forums_count: 0
       ├─image: https://disqus.com/api/users/avatars/soxoj.jpg
       ├─forums_following_count: 0
       ├─is_private: False
       └─comments_count: 0
[?] StackOverflow: https://stackoverflow.com/users/filter?search=soxoj
[+] GitHub: https://github.com/soxoj
        ├─uid: 31013580
        ├─image: https://avatars.githubusercontent.com/u/31013580?v=4
        ├─created_at: 2017-08-14T17:03:07Z
        ├─location: Amsterdam, Netherlands
        ├─follower_count: 1304
        ├─following_count: 54
        ├─fullname: Soxoj
        ├─public_gists_count: 3
        ├─public_repos_count: 88
        ├─twitter_username: sox0j
        ├─bio: Head of OSINT Center of Excellence in @SocialLinks-IO
        ├─is_company: Social Links
        └─blog_url: soxoj.com
[+] TikTok: https://www.tiktok.com/@soxoj
[+] Xvideos: https://xvideos.com/profiles/soxoj
[+] GitHubGist [GitHub]: https://gist.github.com/soxoj
[+] VK: https://vk.com/soxoj
        └─fullname: Aleksej Sokol | VK
Searching |████████                                | ▄▄▄ 94/500 [19%] in 1s (~6s, 78.0/s)
```

1.2 Installation

Maigret can be installed using pip, Docker, or simply can be launched from the cloned repo. Also, it is available online via the community Telegram bot, source code of a bot is available on GitHub.

1.2.1 Windows Standalone EXE-binaries

A standalone `maigret_standalone.exe` for Windows is published in the Releases section of the GitHub repository. A fresh build is produced automatically after each commit to the **main** and **dev** branches.

There are two ways to launch the EXE:

- **Double-click it from Explorer.** Maigret will prompt you for a username, run a default search, and pause at the end so the printed report links remain on screen until you press Enter.
- **Run it from a terminal** for full control over options:
  1. Press `Win+R`, type `cmd`, and hit Enter (or use PowerShell).
  2. Change to the folder where you saved the file, e.g. `cd %USERPROFILE%\Downloads`.
  3. Run it with at least one username:

```
maigret_standalone.exe USERNAME
maigret_standalone.exe USERNAME --html       :: also save an HTML report
maigret_standalone.exe USERNAME --pdf        :: also save a PDF report
maigret_standalone.exe --help                :: list all options
```

Reports are written next to the EXE in a `reports\` subfolder.

Video guide on how to run it: https://youtu.be/qIgwTZOmMmM.

1.2.2 Cloud Shells and Jupyter notebooks

In case you don't want to install Maigret locally, you can use cloud shells and Jupyter notebooks. Press one of the buttons below and follow the instructions to launch it in your browser.

Cloud Shell: https://console.cloud.google.com/cloudshell/open?git_repo=https://github.com/soxoj/maigret&tutorial=README.md

Replit: https://repl.it/github/soxoj/maigret

Google Colab: https://colab.research.google.com/gist/soxoj/879b51bc3b2f8b695abb054090645000/maigret-collab.ipynb

Binder: https://mybinder.org/v2/gist/soxoj/9d65c2f4d3bec5dd25949197ea73cf3a/HEAD

1.2.3 Local installation from PyPi

Maigret ships with a bundled site database. After installation from PyPI (or any other method), it can **automatically fetch a newer compatible database from GitHub** when you run it—see *Database auto-update* in *Settings*.

> **Note**
>
> Python 3.10 or higher and pip is required, **Python 3.11 is recommended.**

```
# install from pypi
pip3 install maigret

# usage
maigret username
```

PDF report support is shipped as an **optional extra** because it relies on system-level graphics libraries that pip cannot install for you. If you plan to use `--pdf`, install Maigret with the `pdf` extra:

```
pip3 install 'maigret[pdf]'
```

See *Optional: PDF reports (maigret[pdf])* below for the full background on why PDF support is optional and how to fix the most common build errors.

1.2.4 Development version (GitHub)

```
git clone https://github.com/soxoj/maigret && cd maigret
pip3 install .

# OR
pip3 install git+https://github.com/soxoj/maigret.git

# usage
maigret username

# OR use poetry in case you plan to develop Maigret
pip3 install poetry
poetry run maigret
```

1.2.5 Docker

```
# official image of the development version, updated from the github repo
docker pull soxoj/maigret

# usage
docker run -v /mydir:/app/reports soxoj/maigret:latest username --html

# manual build
docker build -t maigret .
```

1.2.6 Troubleshooting

If you encounter build errors during installation such as `cannot find ft2build.h` or errors related to `reportlab` / `_renderPM`, you need to install system-level dependencies required to compile native extensions.

Debian/Ubuntu/Kali:

```
sudo apt install -y libfreetype6-dev libjpeg-dev libffi-dev
```

Fedora/RHEL/CentOS:

```
sudo dnf install -y freetype-devel libjpeg-devel libffi-devel
```

Arch Linux:

```
sudo pacman -S freetype2 libjpeg-turbo libffi
```

macOS (Homebrew):

```
brew install freetype
```

After installing the system dependencies, retry the maigret installation.

If you continue to have issues, consider using Docker instead, which includes all necessary dependencies.

1.2.7 Optional: PDF reports (`maigret[pdf]`)

The `--pdf` report format is shipped as an optional extra. To enable it:

```
pip3 install 'maigret[pdf]'
```

If PDF support is not installed and you pass `--pdf`, Maigret prints a warning and continues without crashing — every other output format (`--html`, `--json`, `--csv`, `--txt`, `--xmind`, `--graph`) keeps working.

Why is PDF optional?

Maigret renders PDFs by converting an HTML template, and that conversion pipeline ultimately depends on the `cairo` graphics library through a chain of Python packages roughly shaped like:

```
maigret[pdf] → xhtml2pdf → svglib → rlPyCairo → pycairo → libcairo2 (system)
```

The bottom of that chain is a C library — `libcairo2` — that has to exist on the host *before* pip can build the Python bindings. The Python binding package (`pycairo`) currently ships **only Windows wheels** on PyPI; on Linux and macOS pip falls back to building from source, and the build fails the moment `pkg-config` cannot find `cairo`. The error looks like:

```
../cairo/meson.build:31:12: ERROR: Dependency "cairo" not found (tried pkg-config)
note: This error originates from a subprocess, and is likely not a problem with pip.
error: metadata-generation-failed
```

Pulling this whole chain for every Maigret install just so the much smaller group of users who actually want PDFs can have them is a poor trade — so `xhtml2pdf` is gated behind the `pdf` extra.

Two more packages — `arabic-reshaper` and `python-bidi` — are bundled into the same extra. Maigret core never imports them; they are only used by `xhtml2pdf` to shape Arabic glyphs and lay out right-to-left text in PDFs. `python-bidi` v0.5+ is also a Rust binding, so on niche platforms without a published wheel it would otherwise pull in a Cargo build for users who never asked for PDF support.

Installing the system prerequisites

Install the cairo headers, `pkg-config`, and a working C toolchain *before* running `pip install 'maigret[pdf]'`.

Debian / Ubuntu / Linux Mint / Kali:

```
sudo apt update
sudo apt install -y libcairo2-dev pkg-config python3-dev build-essential
pip3 install --upgrade pip setuptools wheel
pip3 install 'maigret[pdf]'
```

Fedora / RHEL / CentOS:

```
sudo dnf install -y cairo-devel pkgconfig python3-devel gcc
pip3 install 'maigret[pdf]'
```

Arch Linux:

```
sudo pacman -S cairo pkgconf base-devel
pip3 install 'maigret[pdf]'
```

Alpine Linux:

```
sudo apk add cairo-dev pkgconf python3-dev build-base
pip3 install 'maigret[pdf]'
```

macOS (Homebrew):

```
brew install cairo pkg-config
pip3 install --upgrade pip setuptools wheel
pip3 install 'maigret[pdf]'
```

Windows:

No system packages are needed — `pycairo` ships prebuilt wheels for Windows. Just run:

```
pip install 'maigret[pdf]'
```

Google Cloud Shell / Colab / Replit / generic CI:

These environments behave like Debian/Ubuntu — install the same `libcairo2-dev pkg-config python3-dev build-essential` triple before `pip install 'maigret[pdf]'`. If you do not control the base image and cannot `apt install`, skip the extra and use `--html` reports instead; HTML reports contain the same data and open in any browser.

`maigret: command not found` after install

If pip prints warnings like:

```
WARNING: The scripts maigret and update_sitesmd are installed in
'/home/<user>/.local/bin' which is not on PATH.
```

…and `maigret --version` then fails with `command not found`, your `--user` install put the entry-point script in a directory the shell does not search. Add it to `PATH`:

```
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

Or install into a virtual environment, where the entry point lands in the venv's `bin/` automatically:

```
python3 -m venv ~/.venvs/maigret
source ~/.venvs/maigret/bin/activate
pip install 'maigret[pdf]'   # or just `pip install maigret`
```

1.2.8 Optional: Cloudflare bypass solver

> **Warning**
>
> **Experimental.** The Cloudflare webgate is under active development; the configuration schema and CLI behaviour may change without backwards-compatibility guarantees.

Sites tagged `cf_js_challenge` / `cf_firewall` need a real browser to pass their JavaScript challenge. To check those sites you can run a local FlareSolverr instance — Maigret will route protected checks to it when `--cloudflare-bypass` is set:

```
docker run -d -p 8191:8191 --name flaresolverr ghcr.io/flaresolverr/flaresolverr:latest
```

This is **optional** — Maigret runs without it; only sites whose `protection` field intersects `settings.cloudflare_bypass.trigger_protection` require the solver. See *Cloudflare webgate bypass* for details.

1.3 Usage examples

You can use Maigret as:

- a command line tool: initial and a default mode
- a *web interface*: view the graph with results and download all report formats on a single page
- a library: integrate Maigret into your own project

1.3.1 Use Cases

1. Search for accounts with username `machine42` on top 500 sites (by default, according to Majestic Million rank) from the Maigret DB.

```
maigret machine42
```

2. Search for accounts with username `machine42` on **all sites** from the Maigret DB.

```
maigret machine42 -a
```

> **Note**
>
> Maigret will search for accounts on a huge number of sites, and some of them may return false positive results. At the moment, we are working on autorepair mode to deliver the most accurate results.
>
> If you experience many false positives, you can do the following:
>
> - Install the last development version of Maigret from GitHub
> - Run Maigret with `--self-check --auto-disable` flag and agree on disabling of problematic sites

3. Search for accounts with username `machine42` and generate HTML and PDF reports.

```
maigret machine42 -HP
```

or

```
maigret machine42 -a --html --pdf
```

4. Search for accounts with username `machine42` on Facebook only.

```
maigret machine42 --site Facebook
```

5. Extract information from the Steam page by URL and start a search for accounts with found username `machine42`.

```
maigret --parse https://steamcommunity.com/profiles/76561199113454789
```

6. Search for accounts with username `machine42` only on US and Japanese sites.

```
maigret machine42 --tags us,jp
```

7. Search for accounts with username `machine42` only on sites related to software development.

```
maigret machine42 --tags coding
```

8. Search for accounts with username `machine42` on uCoz sites only (mostly CIS countries).

```
maigret machine42 --tags ucoz
```

1.4 FAQ

Short answers to the questions users most often type into the docs search. For deeper coverage each section links to the relevant page.

1.4.1 Can I search by email address?

No. Maigret only takes a username (or one of the *supported identifier types*) as input — searching by an email or mail address is out of scope. Looking up a mail address requires different techniques (probing password-reset flows, registration endpoints) and is the job of a separate class of tool.

Recommended open-source tools for email lookup:

- mailcat
- holehe
- user-scanner

Online services:

- Noimosiny
- OSINT Industries
- Epieos

Note: if Maigret has already found an account for a username, it often extracts the linked email from the profile page automatically — see *Extraction of information from account pages*.

1.4.2 Can I configure a proxy / SOCKS / Tor / I2P?

Yes. Three flags cover three distinct goals:

- `--proxy URL` — route **every** check through the given HTTP or SOCKS proxy (also the right flag for routing the whole run through Tor with `socks5://127.0.0.1:9050`, a residential proxy, or a corporate gateway).
- `--tor-proxy URL` — used **only** for `.onion` sites in the database. Clearweb sites still go via your direct connection (or `--proxy` if set).
- `--i2p-proxy URL` — same idea, only for `.i2p` hosts.

The most common confusion is `--proxy` vs `--tor-proxy`: `--proxy` is "everything through this gateway", `--tor-proxy` is "only onion sites through Tor".

Full walkthrough (Tor Browser vs system `tor` port numbers, Tails OS, timeout / retry tuning): *Tor, I2P, and proxies*.

If your goal is actually "bypass WAF blocks / fix 403 errors", see the *Sites fail / timeout / 403* section below — a residential proxy almost always outperforms Tor or a VPN for that.

1.4.3 Can I use a VPN with Maigret?

Yes, but `--proxy` is usually a better choice. A VPN works transparently at the OS level — Maigret needs no special configuration to use one. However:

- `--proxy` is per-process: it does not affect other apps and does not leak when toggled.
- `--proxy` makes the egress IP visible in logs, which is useful when diagnosing why a batch of sites returned `UNKNOWN`.
- `--proxy` accepts a different value per run, so you can rotate between residential and datacenter exits without touching system network settings.

If a lot of sites are returning 403, the cause is almost certainly that the VPN exit IP is on a WAF blocklist (Cloudflare, DDoS-Guard, Akamai all blanket-block common VPN ranges). A residential proxy via `--proxy` is the usual fix — see the "Lots of sites fail / timeout / return 403" section in TROUBLESHOOTING.md.

1.4.4 Does Maigret check domains via DNS?

Yes, experimentally. With `--with-domains` Maigret resolves a small set of `{username}.<tld>` patterns through DNS (A-records) in parallel with the normal HTTP checks. The current set is `.ddns.net`, `.com`, `.pro`, `.me`, `.biz`, `.email`, `.guru` — 7 entries in the database with `protocol: dns`.

```
maigret <username> --with-domains
```

The flag is marked **experimental**: DNS-only checks can flag parking domains and catch-all wildcards as if the username were registered, so treat hits as a lead rather than confirmation.

If your task is wider DNS reconnaissance — subdomain enumeration, WHOIS history, typo-squatting — Maigret is the wrong tool. Established alternatives:

- dnstwist — typo-squatting and look-alike domains.
- amass / subfinder — subdomain enumeration.
- theHarvester — email / host / subdomain harvesting by domain.

1.4.5 Is there a Maigret Telegram bot?

Yes. A community-maintained bot lets you run Maigret without installing anything locally:

- Working instance: sites.google.com/view/maigret-bot-link (redirect — the hosted bot may move between providers).
- Source code: github.com/soxoj/maigret-tg-bot.

On the question of *searching Telegram itself*: Maigret checks whether a `t.me/<user>` page exists as part of the normal run, but it does not parse channels, posts, members, or message contents. For Telegram content OSINT you need a dedicated tool.

1.4.6 Where is the web interface?

```
maigret --web 5000
```

Then open http://127.0.0.1:5000. Screenshots and a full walkthrough are in *Web Interface*.

1.4.7 Sites fail / timeout / return 403 — connection failures

This is the most common report and is almost always caused by anti-bot protection (Cloudflare, DDoS-Guard, Akamai) or a slow link, not by a bug in Maigret. Quick tweaks, in order:

1. `--timeout 60` — the default 30 s is tight for slow networks and for Tor.
2. `--retries 2` — covers transient failures.
3. `-n 20` — lower concurrency reduces WAF rate-limiting.
4. `--proxy http://user:pass@residential-proxy:port` — datacenter IPs (AWS, GCP, DigitalOcean) and most VPN ranges are blanket-blocked; residential / mobile exits usually fix the bulk of 403s.

The full troubleshooting matrix (per-error recipes for 403, timeout, SSL, captcha, `UNKNOWN` floods) lives in TROUBLESHOOTING.md.

1.4.8 How do I generate a PDF report?

PDF support is an optional extra because it pulls heavy graphics dependencies:

```
pip install 'maigret[pdf]'
maigret <username> --pdf
```

On Linux / macOS you also need system libraries (Pango, Cairo, GDK-PixBuf). Per-OS install steps are in the *Optional: PDF reports* section of *Installation*.

For other report formats (`--html`, `--md`, `--json`, `--csv`, `--txt`, `--xmind`), see *Command line options*.

1.5 Command line options

1.5.1 Usernames

```
maigret username1 username2 ...
```

You can specify several usernames separated by space. Usernames are **not** mandatory as there are other operations modes (see below).

1.5.2 Parsing of account pages and online documents

`maigret --parse URL`

Maigret will try to extract information about the document/account owner (including username and other ids) and will make a search by the extracted username and ids. See examples in the *Extraction of information from account pages* section.

1.5.3 Main options

Options are also configurable through settings files, see *settings section*.

`--tags` - Filter sites for searching by tags: sites categories and two-letter country codes (**not a language!**). E.g. photo, dating, sport; jp, us, global. Multiple tags can be associated with one site. **Warning**: tags markup is not stable now. Read more *in the separate section*.

`--exclude-tags` - Exclude sites with specific tags from the search (blacklist). E.g. `--exclude-tags porn,dating` will skip all sites tagged with `porn` or `dating`. Can be combined with `--tags` to include certain categories while excluding others. Read more *in the separate section*.

`-n`, `--max-connections` - Allowed number of concurrent connections **(default: 100)**.

`-a`, `--all-sites` - Use all sites for scan **(default: top 500)**.

`--top-sites` - Count of sites for scan ranked by Majestic Million **(default: top 500)**.

Mirrors: After the top *N* sites by Majestic Million rank are chosen (respecting `--tags`, `--use-disabled-sites`, etc.), Maigret may add extra sites whose database field `source` names a **parent platform** that itself falls in the Majestic Million top *N* when ranking **including disabled** sites. For example, if `Twitter` ranks in the first 500 by Majestic Million, a mirror such as `memory.lol` (with `source: Twitter`) is included even though it has no rank and would otherwise be cut off. The same applies to Instagram-related mirrors (e.g. Picuki) when `Instagram` is in that parent top *N* by rank—even if the official `Instagram` entry is disabled and not scanned by default, its mirrors can still be pulled in. The final list is the ranked top *N* plus these mirrors (no fixed upper bound on mirror count).

`--timeout` - Time (in seconds) to wait for responses from sites **(default: 30)**. A longer timeout will be more likely to get results from slow sites. On the other hand, this may cause a long delay to gather all results. The choice of the right timeout should be carried out taking into account the bandwidth of the Internet connection.

Network and proxy options

`--proxy PROXY_URL` / `-p PROXY_URL` - Route **every** check through the given HTTP or SOCKS proxy. Example: `socks5://127.0.0.1:1080`, `http://user:pass@proxy.example:3128`. This is the flag to use for routing the whole run through Tor (`--proxy socks5://127.0.0.1:9050`), a residential proxy, or any corporate gateway. No default.

`--tor-proxy TOR_PROXY_URL` - Gateway used **only** for `.onion` sites in the database **(default: socks5://127.0.0.1:9050)**. Clearweb sites are unaffected — for them Maigret uses your direct connection or `--proxy` if you set one. Without this flag, `.onion` sites are silently skipped.

`--i2p-proxy I2P_PROXY_URL` - Gateway used **only** for `.i2p` sites in the database **(default: http://127.0.0.1:4444)**. Same "only matching protocol" rule as `--tor-proxy`.

Maigret does not start the Tor or I2P daemon for you — launch it first. For a full walkthrough (Tor Browser vs system `tor` port numbers, Tails OS recipe, timeout/retry tuning), see *Tor, I2P, and proxies*.

`--cookies-jar-file` - File with custom cookies in Netscape format (aka cookies.txt). You can install an extension to your browser to download own cookies (Chrome, Firefox).

`--no-recursion` - Disable parsing pages for other usernames and recursive search by them.

`--use-disabled-sites` - Use disabled sites to search (may cause many false positives).

`--id-type` - Specify identifier(s) type (default: username). Supported types: gaia_id, vk_id, yandex_public_id, ok_id, wikimapia_uid. Currently, you must add `-a` flag to run a scan on sites with custom id types, sites will be filtered automatically.

`--ignore-ids` - Do not make search by the specified username or other ids. Useful for repeated scanning with found known irrelevant usernames.

`--db` - Load Maigret database from a JSON file or an online, valid, JSON file. See *Using a custom sites database* below.

`--no-autoupdate` - Disable the automatic database update check that runs at startup. The currently cached (or bundled) database is used as-is.

`--force-update` - Force a database update check at startup, ignoring the usual check interval. Implies `--no-autoupdate` for the rest of the run after the explicit update finishes.

`--retries RETRIES` - Count of attempts to restart temporarily failed requests.

`--with-domains` *(experimental)* - Also resolve a small set of `{username}.<tld>` patterns through DNS (A-records) in parallel with the normal HTTP checks. Currently 7 entries in the database use this path (`.ddns.net`, `.com`, `.pro`, `.me`, `.biz`, `.email`, `.guru`). DNS-only hits can include parking domains and catch-all wildcards, so treat results as a lead rather than confirmation. See the *FAQ entry on DNS domain checks*.

`--cloudflare-bypass` *(experimental)* - Route checks for sites tagged `protection: ["cf_js_challenge"]` / `["cf_firewall"]` / `["webgate"]` through a local Chrome-based solver (FlareSolverr by default). The bypass is opt-in — without this flag (or `settings.cloudflare_bypass.enabled = true`) those sites are checked the usual way, which Cloudflare almost always blocks: you get an UNKNOWN status with a JS-challenge / firewall error rather than a real result. Configure the backend in `settings.cloudflare_bypass.modules`. See *Cloudflare webgate bypass*. **Experimental** — the flag, schema and routing rules may change without backwards-compatibility guarantees.

Using a custom sites database

The `--db` flag accepts three forms:

1. **HTTP(S) URL** — fetched as-is, e.g. `--db https://example.com/my_db.json`.
2. **Local file path** — absolute (`--db /tmp/private.json`) or relative to the current working directory (`--db LLM/maigret_private_db.json`).
3. **Module-relative path** — kept for backwards compatibility, resolved against the installed `maigret/` package directory (e.g. the default `resources/data.json`).

Resolution order for local paths: the path is first tried as given (absolute or cwd-relative); if that file does not exist, Maigret falls back to the legacy module-relative resolution. If neither location contains the file, Maigret exits with an error rather than silently loading the bundled database.

When `--db` points to a custom file, automatic database updates are skipped — the file is used exactly as provided.

On every run Maigret prints the database it actually loaded, for example:

```
[+] Using sites database: /path/to/maigret_private_db.json (6 sites)
```

If loading the requested database fails for any other reason (corrupt JSON, missing required keys, ...), Maigret prints a warning, falls back to the bundled database, and reports the fallback explicitly:

```
[-] Falling back to bundled database: /.../maigret/resources/data.json
[+] Using sites database: /.../maigret/resources/data.json (3154 sites)
```

A typical invocation against a private database, with auto-update disabled and all sites scanned, looks like:

```
python3 -m maigret username \
    --db LLM/maigret_private_db.json \
    --no-autoupdate -a
```

1.5.4 Reports

`-P`, `--pdf` - Generate a PDF report (general report on all usernames).

`-H`, `--html` - Generate an HTML report file (general report on all usernames).

`-X`, `--xmind` - Generate an XMind 8 mindmap (one report per username).

`-C`, `--csv` - Generate a CSV report (one report per username).

`-T`, `--txt` - Generate a TXT report (one report per username).

`-J`, `--json` - Generate a JSON report of specific type: simple, ndjson (one report per username). E.g. `--json ndjson`

`-M`, `--md` - Generate a Markdown report (general report on all usernames). See *Markdown report (LLM-friendly)* below.

`--ai` - Run an AI-powered analysis of the search results using an OpenAI-compatible chat completion API. The internal Markdown report is sent to the model, which returns a short investigation summary that is streamed to the terminal. See *AI analysis (built-in)* below.

`--ai-model` - Model name to use with `--ai`. Defaults to `openai_model` from settings (`gpt-4o` out of the box).

`-fo`, `--folderoutput` - Results will be saved to this folder, `results` by default. Will be created if doesn't exist.

`--web PORT` - Start the built-in web interface on the given port and serve results / downloadable reports from a single page. Example: `maigret --web 5000` → open `http://127.0.0.1:5000`. Full walkthrough with screenshots: *Web Interface*.

1.5.5 Output options

`-v`, `--verbose` - Display extra information and metrics. *(loglevel=WARNING)*

`-vv`, `--info` - Display service information. *(loglevel=INFO)*

`-vvv`, `--debug`, `-d` - Display debugging information and site responses. *(loglevel=DEBUG)*

`--print-not-found` - Print sites where the username was not found.

`--print-errors` - Print errors messages: connection, captcha, site country ban, etc.

1.5.6 Other operations modes

`--version` - Display version information and dependencies.

`--self-check` - Do self-checking for sites and database. Each site is tested by looking up its known-claimed and known-unclaimed usernames and verifying that the results match expectations. Individual site failures (network errors, unexpected exceptions, etc.) are caught and logged without stopping the overall process, so the check always runs to completion. After checking, Maigret reports a summary of issues found. If any sites were disabled (see `--auto-disable`), Maigret asks if you want to save updates; answering y/Y will rewrite the local database.

`--auto-disable` - Used with `--self-check`: automatically disable sites that fail checks (incorrect detection of claimed/unclaimed usernames, connection errors, or unexpected exceptions). Without this flag, `--self-check` only **reports** issues without modifying the database.

`--diagnose` - Used with `--self-check`: print detailed diagnosis information for each failing site, including the check type, the list of issues found, and recommendations (e.g. suggesting a different `checkType`).

`--submit URL` - Do an automatic analysis of the given account URL or site main page URL to determine the site engine and methods to check account presence. After checking Maigret asks if you want to add the site, answering y/Y will rewrite the local database.

1.5.7 Markdown report (LLM-friendly)

The `--md` / `-M` flag generates a Markdown report designed for both human reading and analysis by AI assistants (ChatGPT, Claude, etc.).

```
maigret username --md
```

The report includes:

- **Summary** with aggregated personal data (all fullnames, locations, bios found across accounts), country tags, website tags, first/last seen timestamps.
- **Per-account sections** with profile URL, site tags, and all extracted fields (username, bio, follower count, linked accounts, etc.).
- **Possible false positives** disclaimer explaining that accounts may belong to different people.
- **Ethical use** notice about applicable data protection laws.

Using with AI tools:

The Markdown format is optimized for LLM context windows. You can feed the report directly to an AI assistant for follow-up analysis:

```
# Generate the report
maigret johndoe --md

# Feed it to an AI tool
cat reports/report_johndoe.md | llm "Analyze this OSINT report and summarize key findings
↪"
```

The structured Markdown with per-site sections makes it easy for AI tools to extract relationships, cross-reference identities, and identify patterns across accounts.

For a built-in alternative that calls the model for you and prints the summary directly, see *AI analysis (built-in)* below.

1.5.8 AI analysis (built-in)

The `--ai` flag turns the search results into a short investigation summary by sending the internal Markdown report to an OpenAI-compatible chat completion API and streaming the model's reply to the terminal.

```
export OPENAI_API_KEY=sk-...
maigret username --ai

# use a smaller / cheaper model
maigret username --ai --ai-model gpt-4o-mini
```

While `--ai` is active, per-site progress lines and the short text report at the end are suppressed so the streamed summary is the main output. The Markdown report itself is built in memory and is **not** written to disk by `--ai` alone — combine with `--md` if you also want the file on disk.

The summary follows a fixed format with sections for the most likely real name, location, occupation, interests, languages, main website, username variants, number of platforms, active years, a confidence rating, and a short list of follow-up leads. The model is instructed to rely only on what is supported by the report and to avoid mixing clearly unrelated profiles into the main identity.

Configuration. The API key is resolved from `settings.openai_api_key` first, then from the `OPENAI_API_KEY` environment variable. The endpoint defaults to `https://api.openai.com/v1` and can be redirected to any OpenAI-compatible service (Azure OpenAI, OpenRouter, a local server, ...) by setting `openai_api_base_url` in `settings.json`. See *Settings* for the full list of options.

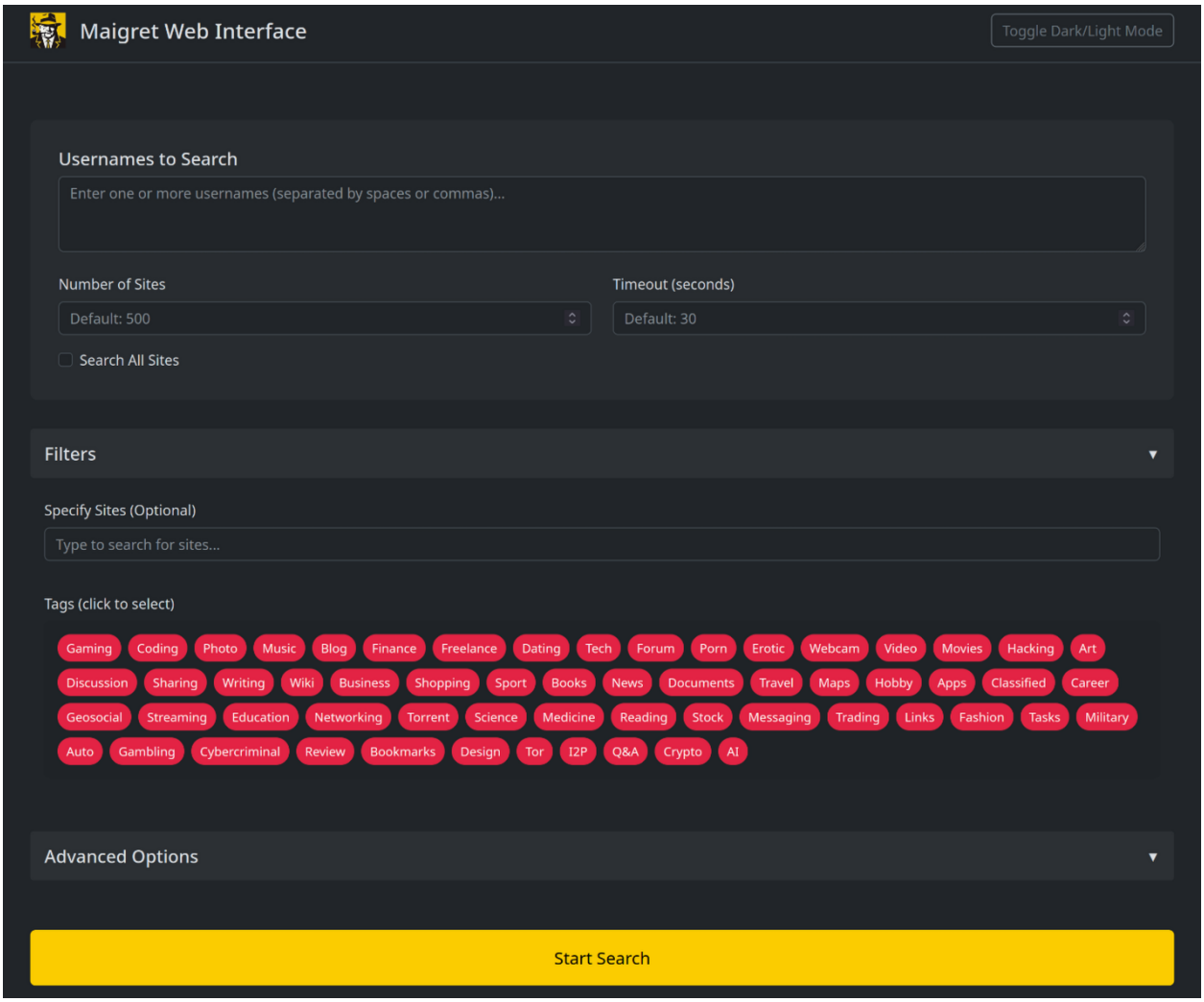
> **Note**
>
> `--ai` makes a network request to the configured chat completion endpoint and sends the full Markdown report (which contains the gathered profile data). Use it only with providers and accounts you trust with that data.

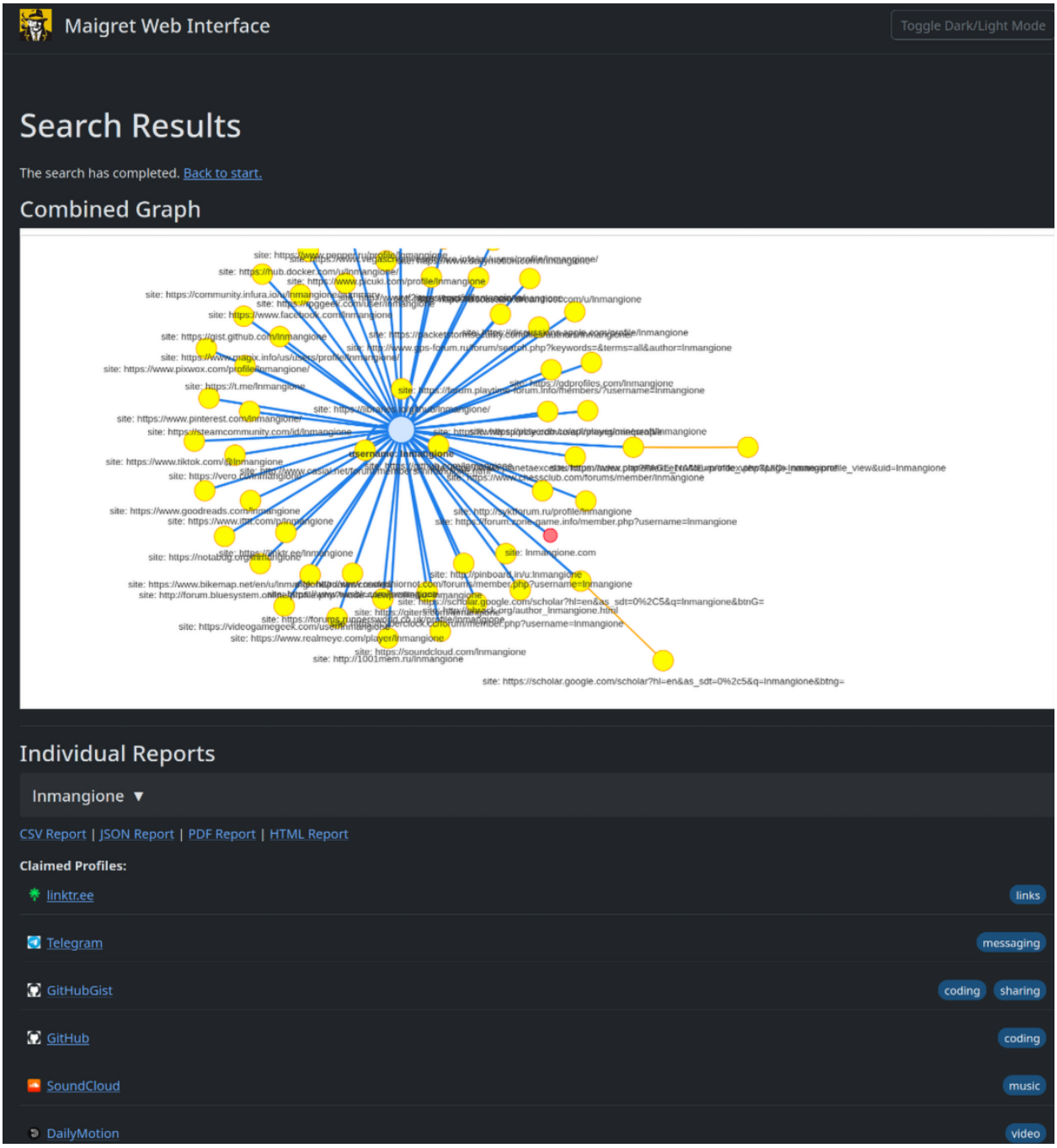
1.6 Features

This is the list of Maigret features.

1.6.1 Web Interface

You can run Maigret with a web interface, where you can view the graph with results and download reports of all formats on a single page.





Instructions:

1. Run Maigret with the `--web` flag and specify the port number.

```
maigret --web 5000
```

2. Open http://127.0.0.1:5000 in your browser and enter one or more usernames to make a search.
3. Wait a bit for the search to complete and view the graph with results, the table with all accounts found, and download reports of all formats.

1.6.2 Telegram bot

A community-maintained Telegram bot lets you run Maigret without installing anything locally.

- Working instance: sites.google.com/view/maigret-bot-link (redirect — the hosted bot may move between providers).
- Source code: github.com/soxoj/maigret-tg-bot.

1.6.3 Personal info gathering

Maigret does the parsing of accounts webpages and extraction of personal info, links to other profiles, etc. Extracted info displayed as an additional result in CLI output and as tables in HTML and PDF reports. Also, Maigret use found ids and usernames from links to start a recursive search.

Enabled by default, can be disabled with `--no extracting`.

```
$ python3 -m maigret soxoj --timeout 5
    [-] Starting a search on top 500 sites from the Maigret database...
    [!] You can run search by full list of sites with flag `-a`
    [*] Checking username soxoj on:
    ...
    [+] GitHub: https://github.com/soxoj
            ├─uid: 31013580
            ├─image: https://avatars.githubusercontent.com/u/31013580?v=4
            ├─created_at: 2017-08-14T17:03:07Z
            ├─location: Amsterdam, Netherlands
            ├─follower_count: 1304
            ├─following_count: 54
            ├─fullname: Soxoj
            ├─public_gists_count: 3
            ├─public_repos_count: 88
            ├─twitter_username: sox0j
            ├─bio: Head of OSINT Center of Excellence in @SocialLinks-IO
            ├─is_company: Social Links
            └─blog_url: soxoj.com
    ...
```

1.6.4 Recursive search

Maigret has the ability to scan account pages for *common identifiers* and usernames found in links. When people include links to their other social media accounts, Maigret can automatically detect and initiate new searches for those profiles. Any information discovered through this process will be shown in both the command-line interface output and generated reports.

Enabled by default, can be disabled with `--no-recursion`.

```
$ python3 -m maigret soxoj --timeout 5
    [-] Starting a search on top 500 sites from the Maigret database...
    [!] You can run search by full list of sites with flag `-a`
    [*] Checking username soxoj on:
    ...
    [+] GitHub: https://github.com/soxoj
            ├─uid: 31013580
            ├─image: https://avatars.githubusercontent.com/u/31013580?v=4
            ├─created_at: 2017-08-14T17:03:07Z
```

(continues on next page)

(continued from previous page)

```
            ├─location: Amsterdam, Netherlands
            ├─follower_count: 1304
            ├─following_count: 54
            ├─fullname: Soxoj
            ├─public_gists_count: 3
            ├─public_repos_count: 88
            ├─twitter_username: sox0j     <===== another username found here
            ├─bio: Head of OSINT Center of Excellence in @SocialLinks-IO
            ├─is_company: Social Links
            └─blog_url: soxoj.com
    ...
    Searching || 500/500 [100%] in 9.1s (54.85/s)
    [-] You can see detailed site check errors with a flag `--print-errors`
    [*] Checking username sox0j on:
    [+] Telegram: https://t.me/sox0j
        ├─fullname: @Sox0j
        ...
```

1.6.5 Username permutations

Maigret can generate permutations of usernames. Just pass a few usernames in the CLI and use `--permute` flag. Thanks to @balestek for the idea and implementation.

```
$ python3 -m maigret --permute hope dream --timeout 5
[-] 12 permutations from hope dream to check...
    ├─ hopedream
    ├─ _hopedream
    ├─ hopedream_
    ├─ hope_dream
    ├─ hope-dream
    ├─ hope.dream
    ├─ dreamhope
    ├─ _dreamhope
    ├─ dreamhope_
    ├─ dream_hope
    ├─ dream-hope
    └─ dream.hope
[-] Starting a search on top 500 sites from the Maigret database...
[!] You can run search by full list of sites with flag `-a`
[*] Checking username hopedream on:
...
```

1.6.6 Reports

Maigret currently supports HTML, PDF, TXT, XMind 8 mindmap, and JSON reports.

HTML/PDF reports contain:

- profile photo
- all the gathered personal info
- additional information about supposed personal data (full name, gender, location), resulting from statistics of all found accounts

Also, there is a short text report in the CLI output after the end of a searching phase.

> **Warning**
>
> XMind 8 mindmaps are incompatible with XMind 2022!

1.6.7 AI analysis

Maigret can produce a short, human-readable investigation summary on top of the raw search results using the `--ai` flag. It builds the internal Markdown report, sends it to an OpenAI-compatible chat completion endpoint, and streams the model's reply directly to the terminal.

```
export OPENAI_API_KEY=sk-...
maigret username --ai
```

The summary uses a fixed format with the most likely real name, location, occupation, interests, languages, main website, username variants, number of platforms, active years, a confidence rating, and a short list of follow-up leads. While `--ai` is active, per-site progress and the short text report are suppressed so the streamed summary is the main output.

The endpoint, model, and API key are configured via `settings.json` (`openai_api_key`, `openai_model`, `openai_api_base_url`) or the `OPENAI_API_KEY` environment variable. Any OpenAI-compatible API can be used (Azure OpenAI, OpenRouter, a local server, ...). See *AI analysis (built-in)* and *Settings* for details.

1.6.8 Tags

The Maigret sites database very big (and will be bigger), and it is maybe an overhead to run a search for all the sites. Also, it is often hard to understand, what sites more interesting for us in the case of a certain person.

Tags markup allows selecting a subset of sites by interests (photo, messaging, finance, etc.) or by country. Tags of found accounts grouped and displayed in the reports.

See full description *in the Tags Wiki page*.

1.6.9 Censorship and captcha detection

Maigret can detect common errors such as censorship stub pages, CloudFlare captcha pages, and others. If you get more them 3% errors of a certain type in a session, you've got a warning message in the CLI output with recommendations to improve performance and avoid problems.

1.6.10 Retries

Maigret will do retries of the requests with temporary errors got (connection failures, proxy errors, etc.).

One attempt by default, can be changed with option `--retries N`.

1.6.11 Database self-check

Maigret includes a self-check mode (`--self-check`) that validates every site in the database by looking up its known-claimed and known-unclaimed usernames and verifying that the detection results match expectations.

The self-check is **error-resilient**: if an individual site check raises an unexpected exception (e.g. a network error or a parsing failure), the error is caught, logged, and recorded as an issue — the remaining sites continue to be checked without interruption. This means the process always runs to completion, even when checking hundreds of sites with `-a --self-check`.

Use `--auto-disable` together with `--self-check` to automatically disable sites that fail checks. Without it, issues are only reported. Use `--diagnose` to print detailed per-site diagnosis including the check type, specific issues, and recommendations.

```
# Report-only mode (no changes to the database)
maigret --self-check

# Automatically disable failing sites and save updates
maigret -a --self-check --auto-disable

# Show detailed diagnosis for each failing site
maigret -a --self-check --diagnose
```

1.6.12 Archives and mirrors checking

The Maigret database contains not only the original websites, but also mirrors, archives, and aggregators. For example:

- Picuki, Instagram mirror
- (no longer available) Reddit BigData search
- (no longer available) Twitter shadowban checker

It allows getting additional info about the person and checking the existence of the account even if the main site is unavailable (bot protection, captcha, etc.)

1.6.13 Cloudflare webgate bypass

> **Warning**
>
> **Experimental feature.** The Cloudflare webgate is under active development. The configuration schema, CLI flag behaviour, and the set of sites that route through it may change without backwards-compatibility guarantees. Expect rough edges (CF rate limits, occasional solver failures) and report issues so they can be ironed out.

Some sites sit behind a full Cloudflare JavaScript challenge or a CF firewall hard block — these are tagged `protection: ["cf_js_challenge"]` or `protection: ["cf_firewall"]` in the database and are normally kept disabled because neither aiohttp nor curl_cffi can solve the JS challenge on their own.

Maigret can offload these checks to a local Chrome-based solver. Two backends are supported, configured in `settings.json` under `cloudflare_bypass.modules` (the first reachable module wins; subsequent ones are tried as a fallback chain):

- **FlareSolverr** (recommended). Runs a real Chrome instance and exposes a JSON API. The upstream HTTP status, headers and final URL are preserved, so `checkType: status_code` and `checkType: response_url` keep working through the bypass.

  ```
  docker run -d -p 8191:8191 --name flaresolverr ghcr.io/flaresolverr/
  ↪flaresolverr:latest
  ```

- **CloudflareBypassForScraping** (legacy fallback). Returns rendered HTML only, so the upstream status code is lost — `checkType: message` keeps working but `status_code` checks misfire (treated as 200 on success).

Activate the bypass either with the CLI flag:

```
maigret --cloudflare-bypass <username>
```

or by setting `cloudflare_bypass.enabled` to `true` in `settings.json`. The web UI (`python -m maigret.web.app`) reads the same setting — there is no separate toggle in the form, so flipping `enabled` is what activates the bypass for browser-driven runs.

The bypass only fires for sites whose `protection` field intersects `cloudflare_bypass.trigger_protection` (default `["cf_js_challenge", "cf_firewall", "webgate"]`); all other sites use the normal aiohttp / curl_cffi path.

If all configured modules are unreachable, affected sites get an UNKNOWN status with an actionable error pointing at the first module's URL — the fix is almost always to start the FlareSolverr container.

FlareSolverr session reuse is automatic: Maigret pins a single `session: <session_prefix>-<pid>` per run, so cf_clearance cookies are shared between checks of the same domain (5–10× faster on subsequent requests to that host).

1.6.14 Activation

The activation mechanism helps make requests to sites requiring additional authentication like cookies, JWT tokens, or custom headers.

It works by implementing a custom function that:

1. Makes a specialized HTTP request to a specific website endpoint
2. Processes the response
3. Updates the headers/cookies for that site in the local Maigret database

Since activation only triggers after encountering specific errors, a retry (or another Maigret run) is needed to obtain a valid response with the updated authentication.

The activation mechanism is enabled by default, and cannot be disabled at the moment.

See for more details in Development section *Activation mechanism*.

1.6.15 Extraction of information from account pages

Maigret can parse URLs and content of web pages by URLs to extract info about account owner and other meta information.

You must specify the URL with the option `--parse`, it's can be a link to an account or an online document. List of supported sites see here.

After the end of the parsing phase, Maigret will start the search phase by *supported identifiers* found (usernames, ids, etc.).

```
$ maigret --parse https://docs.google.com/spreadsheets/d/
↪1HtZKMLRXNsZ0HjtBmo0Gi03nUPiJIA4CC4jTYbCAnXw/edit\#gid\=0

Scanning webpage by URL https://docs.google.com/spreadsheets/d/
↪1HtZKMLRXNsZ0HjtBmo0Gi03nUPiJIA4CC4jTYbCAnXw/edit#gid=0...
org_name: Gooten
mime_type: application/vnd.google-apps.ritz
Scanning webpage by URL https://clients6.google.com/drive/v2beta/files/
↪1HtZKMLRXNsZ0HjtBmo0Gi03nUPiJIA4CC4jTYbCAnXw?fields=alternateLink
↪%2CcopyRequiresWriterPermission%2CcreatedDate%2Cdescription%2CdriveId%2CfileSize
↪%2CiconLink%2Cid%2Clabels(starred%2C%20trashed)%2ClastViewedByMeDate%2CmodifiedDate
↪%2Cshared%2CteamDriveId%2CuserPermission(id%2Cname%2CemailAddress%2Cdomain%2Crole
↪%2CadditionalRoles%2CphotoLink%2Ctype%2CwithLink)%2Cpermissions(id%2Cname
↪%2CemailAddress%2Cdomain%2Crole%2CadditionalRoles%2CphotoLink%2Ctype%2CwithLink)
↪%2Cparents(id)%2Ccapabilities(canMoveItemWithinDrive%2CcanMoveItemOutOfDrive
↪%2CcanMoveItemOutOfTeamDrive%2CcanAddChildren%2CcanEdit%2CcanDownload%2CcanComment
```

(continues on next page)

(continued from previous page)

```
↪%2CcanMoveChildrenWithinDrive%2CcanRename%2CcanRemoveChildren
↪%2CcanMoveItemIntoTeamDrive)%2Ckind&supportsTeamDrives=true&enforceSingleParent=true&
↪key=AIzaSyC1eQ1xj69IdTMeii5r7brs3R90eck-m7k...
created_at: 2016-02-16T18:51:52.021Z
updated_at: 2019-10-23T17:15:47.157Z
gaia_id: 15696155517366416778
fullname: Nadia Burgess
email: nadia@gooten.com
image: https://lh3.googleusercontent.com/a-/
↪AOh14GheZe1CyNa3NeJInWAl70qkip4oJ7qLsD8vDy6X=s64
email_username: nadia
```

```
$ maigret.py --parse https://steamcommunity.com/profiles/76561199113454789
Scanning webpage by URL https://steamcommunity.com/profiles/76561199113454789...
steam_id: 76561199113454789
nickname: Pok
username: Machine42
```

1.6.16 Simple API

Maigret can be easily integrated with the use of Python package maigret.

Example: the community Telegram bot

1.7 Philosophy

> *The Commissioner Jules Maigret is a fictional French police detective, created by Georges Simenon. His investigation method is based on understanding the personality of different people and their interactions.*

TL;DR: Username => Dossier

Maigret is designed to gather all the available information about person by his username.

What kind of information is this? First, links to person accounts. Secondly, all the machine-extractable pieces of info, such as: other usernames, full name, URLs to people's images, birthday, location (country, city, etc.), gender.

All this information forms some dossier, but it also useful for other tools and analytical purposes. Each collected piece of data has a label of a certain format (for example, `follower_count` for the number of subscribers or `created_at` for account creation time) so that it can be parsed and analyzed by various systems and stored in databases.

1.7.1 Origins

Maigret started from studying what OSINT investigators actually use in practice — and from the realization that many popular tools do not deliver real investigative value. The original research behind this observation is summarized in the article What's wrong with namecheckers. For a broader landscape of username-checking tools, see the curated OSINT namecheckers list.

Two ideas grew out of that research:

- socid-extractor — a library focused on pulling structured identity data (user IDs, full names, linked accounts, bios, timestamps, etc.) out of account pages and public API responses, so that finding an account is not the end of the pipeline.
- **Maigret** itself — which started as a fork of Sherlock but has long since outgrown the original project in coverage, extraction depth, and check reliability. Today Maigret is used as a component by major OSINT vendors in their commercial products.

1.8 Supported identifier types

Maigret can search against not only ordinary usernames, but also through certain common identifiers. There is a list of all currently supported identifiers.

- **gaia_id** - Google inner numeric user identifier, in former times was placed in a Google Plus account URL.
- **steam_id** - Steam inner numeric user identifier.
- **wikimapia_uid** - Wikimapia.org inner numeric user identifier.
- **uidme_uguid** - uID.me inner numeric user identifier.
- **yandex_public_id** - Yandex sites inner letter user identifier. See also: YaSeeker.
- **vk_id** - VK.com inner numeric user identifier.
- **ok_id** - OK.ru inner numeric user identifier.
- **yelp_userid** - Yelp inner user identifier.

1.9 Tags

The use of tags allows you to select a subset of the sites from big Maigret DB for search.

> **Warning**
>
> Tags markup is still not stable.

There are several types of tags:

1. **Country codes**: `us`, `jp`, `br`... (ISO 3166-1 alpha-2). A country tag means that having an account on the site implies a connection to that country — either origin or residence. The goal is attribution, not perfect accuracy.
   - **Global sites** (GitHub, YouTube, Reddit, Medium, etc.) get **no country tag** — an account there says nothing about where a person is from.
   - **Regional/local sites** where an account implies a specific country **must** have a country tag: `VK` → `ru`, `Naver` → `kr`, `Zhihu` → `cn`.
   - Multiple country tags are allowed when a service is used predominantly in a few countries (e.g. `Xing` → `de`, `eu`).
   - Do **not** assign country tags based on traffic statistics alone — a site popular in India by traffic is not "Indian" if it is used globally.
2. **Site engines**. Most of them are forum engines now: `uCoz`, `vBulletin`, `XenForo` et al. Full list of engines stored in the Maigret database.
3. **Sites' subject/type and interests of its users**. Full list of "standard" tags is present in the source code only for a moment.

1.9.1 Usage

`--tags us,jp` – search on US and Japanese sites (actually marked as such in the Maigret database)

`--tags coding` – search on sites related to software development.

`--tags ucoz` – search on uCoz sites only (mostly CIS countries)

1.9.2 Blacklisting (excluding) tags

You can exclude sites with certain tags from the search using `--exclude-tags`:

`--exclude-tags porn,dating` – skip all sites tagged with `porn` or `dating`.

`--exclude-tags ru` – skip all Russian sites.

You can combine `--tags` and `--exclude-tags` to fine-tune your search:

`--tags forum --exclude-tags ru` – search on forum sites, but skip Russian ones.

In the web interface, the tag cloud supports three states per tag: click once to **include** (green), click again to **exclude** (dark/strikethrough), and click once more to return to **neutral** (red).

1.10 Development

1.10.1 Frequently Asked Questions

1. Where to find the list of supported sites?

The human-readable list of supported sites is available in the sites.md file in the repository. It's been generated automatically from the main JSON file with the list of supported sites.

The machine-readable JSON file with the list of supported sites is available in the data.json file in the directory *resources*.

2. Which methods to check the account presence are supported?

The supported methods (`checkType` values in `data.json`) are:

- `message` - the most reliable method, checks if any string from `presenceStrs` is present and none of the strings from `absenceStrs` are present in the HTML response
- `status_code` - checks that status code of the response is 2XX
- `response_url` - check if there is not redirect and the response is 2XX

> **Note**
>
> Maigret natively treats specific anti-bot HTTP status codes (like LinkedIn's `HTTP 999`) as a standard "Not Found/Available" signal instead of throwing an infrastructure Server Error, gracefully preventing false positives.

See the details of check mechanisms in the checking.py file.

> **Note**
>
> Maigret now uses the **Majestic Million** dataset for site popularity sorting instead of the discontinued Alexa Rank API. For backward compatibility with existing configurations and parsers, the ranking field in *data.json* and internal site models remains named `alexaRank` and `alexa_rank`.

Mirrors and ``–top-sites``: When you limit scans with `--top-sites N`, Maigret also includes *mirror* sites (entries whose `source` field points at a parent platform such as Twitter or Instagram) if that parent would appear in the Majestic Million top *N* when disabled sites are considered for ranking. See the **Mirrors** paragraph under `--top-sites` in *Command line options*.

1.10.2 Testing

It is recommended use Python 3.10 for testing.

Install test requirements:

```
poetry install --with dev
```

Use the following commands to check Maigret:

```
# run linter and typing checks
# order of checks:
# - critical syntax errors or undefined names
# - flake checks
# - mypy checks
make lint

# run black formatter
make format

# run testing with coverage html report
# current test coverage is 58%
make test

# open html report
open htmlcov/index.html

# get flamechart of imports to estimate startup time
make speed
```

1.10.3 Site naming conventions

Site names are the keys in `data.json` and appear in user-facing reports. Follow these rules:

- **Title Case** by default: `Product Hunt`, `Hacker News`.
- **Lowercase** only if the brand itself is written that way: `kofi`, `note`, `hi5`.
- **No domain suffix** (`calendly.com` → `Calendly`), unless the domain is part of the recognized brand name: `last.fm`, `VC.ru`, `Archive.org`.
- **No full UPPERCASE** unless the brand is an acronym: `VK`, `CNET`, `ICQ`, `IFTTT`.
- **No `www.` or `https://` prefix** in the name.
- **Spaces** are allowed when the brand uses them: `Star Citizen`, `Google Maps`.
- **{username} templates** in names are acceptable: `{username}.tilda.ws`.

When in doubt, check how the service refers to itself on its homepage.

1.10.4 How to fix false-positives

If you want to work with sites database, don't forget to activate statistics update git hook, command for it would look like this: `git config --local core.hooksPath .githooks/`.

You should make your git commits from your maigret git repo folder, or else the hook wouldn't find the statistics update script.

1. Determine the problematic site.

If you already know which site has a false-positive and want to fix it specifically, go to the next step.

Otherwise, simply run a search with a random username (e.g. *laiuhi3h4gi3u4hgt*) and check the results. Alternatively, you can use the community Telegram bot.

2. Open the account link in your browser and check:

- If the site is completely gone, remove it from the list
- If the site still works but looks different, update in data.json how we check it
- If the site requires login to view profiles, disable checking it

3. Find the site in the data.json file.

If the `checkType` method is not `message` and you are going to fix check, update it: - put `message` in `checkType` - put in `absenceStrs` a keyword that is present in the HTML response for an non-existing account - put in `presenceStrs` a keyword that is present in the HTML response for an existing account

If you have trouble determining the right keywords, you can use automatic detection by passing the account URL with the `--submit` option:

```
maigret --submit https://my.mail.ru/bk/alex
```

To disable checking, set `disabled` to `true` or simply run:

```
maigret --self-check --site My.Mail.ru@bk.ru
```

To debug the check method using the response HTML, you can run:

```
maigret soxoj --site My.Mail.ru@bk.ru -d 2> response.txt
```

There are few options for sites data.json helpful in various cases:

- `engine` - a predefined check for the sites of certain type (e.g. forums), see the `engines` section in the JSON file
- `headers` - a dictionary of additional headers to be sent to the site
- `requestHeadOnly` - set to `true` if it's enough to make a HEAD request to the site
- `regexCheck` - a regex to check if the username is valid, in case of frequent false-positives
- `requestMethod` - set the HTTP method to use (e.g., `POST`). By default, Maigret natively defaults to GET or HEAD.
- `requestPayload` - a dictionary with the JSON payload to send for POST requests (e.g., `{"username": "{username}"}`), extremely useful for parsing GraphQL or modern JSON APIs.
- `protection` - a list of protection types detected on the site (see below).

`protection` (site protection tracking)

The `protection` field records what kind of anti-bot protection a site uses. Maigret reads this field and automatically applies the appropriate bypass mechanism where one exists.

Two categories of tag:

- **Load-bearing.** Maigret changes its HTTP client or headers based on the tag. Currently only `tls_fingerprint` (switches to `curl_cffi` with Chrome-class TLS).
- **Documentation-only.** Maigret does **not** change behavior based on the tag; it records *why* the site is hard so a future solver can target the right set of sites without re-auditing.

Within the documentation-only tags, there is a further split that dictates whether the site is `disabled: true`:

- `ip_reputation` is the **only** doc-tag that **keeps the site enabled**. It means "works for most users, fails from datacenter/cloud IPs." Disabling would silently hide a working site from anyone with a clean IP. The fix is **external** to Maigret (residential IP or `--proxy`).
- `cf_js_challenge`, `cf_firewall`, `aws_waf_js_challenge`, `ddos_guard_challenge`, `custom_bot_protection`, `js_challenge` all pair with `disabled: true`. They mean "does not work for anyone right now"; the tag identifies the provider so that when a bypass ships, every site with that tag can be re-enabled in one pass.

Supported values:

- `tls_fingerprint` *(load-bearing; site stays enabled)* — the site fingerprints the TLS handshake (JA3/JA4) and blocks non-browser clients. Maigret automatically uses `curl_cffi` with Chrome browser emulation to bypass this. Requires the `curl_cffi` package (included as a dependency). Examples: Instagram, NPM, Codepen, Kickstarter, Letterboxd.
- `ip_reputation` *(documentation-only; site stays enabled)* — the site blocks requests from datacenter/cloud IPs regardless of headers or TLS. Cannot be bypassed automatically; run Maigret from a regular internet connection (not a datacenter) or use a proxy (`--proxy`). The site is **not** marked `disabled` because it continues to work for users on residential IPs. Examples: Reddit, Patreon, Figma, OnlyFans.
- `cf_js_challenge` *(documentation-only; pair with ``disabled: true``)* — Cloudflare Managed Challenge / Turnstile JS challenge. Symptom: HTTP 403 with `cf-mitigated: challenge` header; body contains `challenges.cloudflare.com`, `_cf_chl_opt`, `window._cf_chl`, or "Just a moment". Not bypassable via `curl_cffi` TLS impersonation (verified across Chrome 123/124/131, Safari 17/18, Firefox 133/135, Edge 101 — all return the same 403 challenge page); a real browser executing the challenge JS is required to obtain the clearance cookie. Sites stay `disabled: true` until a CF-challenge solver is integrated. Examples: DMOJ, Elakiri, Fanlore, Bdoutdoors, TheStudentRoom, forum.hr.
- `cf_firewall` *(documentation-only; pair with ``disabled: true``)* — Cloudflare firewall rule / bot score block (WAF action=block, **not** action=challenge). Symptom: HTTP 403 served by Cloudflare (`server: cloudflare`, `cf-ray` header) **without** JS-challenge markers — body typically shows "Access denied", "Attention Required", or just a bare 1015/1016/1020 error page. Unlike `ip_reputation`, residential IPs are **not** sufficient to bypass — Cloudflare decides based on a composite of bot score, TLS fingerprint, UA, ASN, and custom site-owner rules, so `curl_cffi` Chrome impersonation from a residential line still returns 403. Sites stay `disabled: true` until a per-site bypass (cookies, real browser, or residential+clean session) is found. Examples: Fark, Fodors, Huntingnet, Hunttalk.
- `aws_waf_js_challenge` *(documentation-only; pair with ``disabled: true``)* — the site is protected by AWS WAF with a JavaScript challenge. Symptom: HTTP 202 with empty body and `x-amzn-waf-action: challenge` header (a token-granting challenge that requires executing the CAPTCHA/challenge JS bundle). Neither `curl_cffi` TLS impersonation nor User-Agent changes bypass this — a real browser or the official AWS WAF challenge-solver SDK is required. Sites stay `disabled: true` until a solver is integrated. Example: Dreamwidth.
- `ddos_guard_challenge` *(documentation-only; pair with ``disabled: true``)* — DDoS-Guard (ddos-guard.net) anti-bot page. Symptom: HTTP 403 with `server: ddos-guard` header; body contains "DDoS-Guard". DDoS-Guard fingerprints different UAs per source IP, so a single User-Agent override does not work across environments; a JS-capable bypass or DDoS-Guard-aware solver is required. Sites stay `disabled: true` until a solver is integrated. Example: ForumHouse.
- `js_challenge` *(documentation-only; pair with ``disabled: true``)* — **fallback** for JavaScript-challenge systems whose provider cannot be identified (custom in-house challenge pages that are not Cloudflare, AWS WAF, or any other recognized vendor). Prefer a provider-specific tag whenever the provider can be pinned down from response headers or body signatures.
- `custom_bot_protection` *(documentation-only; pair with ``disabled: true``)* — **fallback** for non-JS-challenge bot protection served by a custom/in-house system (not Cloudflare, not AWS WAF, not DDoS-Guard). Typical symptom: HTTP 403 from the site's own origin server (`server: nginx`, AWS ELB, etc.) with a branded block

page, returned regardless of TLS fingerprint or residential IP. Not generically bypassable; investigate per site (cookies, session, proxy geography). Examples: Hackerearth ("HackerEarth Guardian"), FreelanceJob (nginx-level block).

Rule: prefer provider-specific protection tags. When a site is blocked by an identifiable anti-bot vendor, always record the vendor in the tag (`cf_js_challenge`, `cf_firewall`, `aws_waf_js_challenge`, `ddos_guard_challenge`, and future additions such as `sucuri_challenge`, `incapsula_challenge`). The generic `js_challenge` and `custom_bot_protection` tags are reserved for custom/unknown systems. Rationale: bypass solvers are inherently provider-specific (a Cloudflare Turnstile solver does not help with AWS WAF); recording the provider in advance lets us fan out fixes the moment a per-provider solver is added, without re-auditing every disabled site. The same principle applies to other protection categories when the provider is identifiable.

Example:

```
"Instagram": {
    "url": "https://www.instagram.com/{username}/",
    "checkType": "message",
    "presenseStrs": ["\"routePath\":\"\\/"],
    "absenceStrs": ["\"routePath\":null"],
    "protection": ["tls_fingerprint"]
}
```

`urlProbe` (optional profile probe URL)

By default Maigret performs the HTTP request to the same URL as `url` (the public profile link pattern).

If you set `urlProbe` in `data.json`, Maigret **fetches** that URL for the presence check (API, GraphQL, JSON endpoint, etc.), while **reports and ``url_user``** still use `url` — the human-readable profile page users should open.

Placeholders: `{username}`, `{urlMain}`, `{urlSubpath}` (same as for `url`). Example: GitHub uses `url` `https://github.com/{username}` and `urlProbe` `https://api.github.com/users/{username}`; Picsart uses the web profile `https://picsart.com/u/{username}` and probes `https://api.picsart.com/users/show/{username}.json`.

Implementation: `make_site_result` in checking.py.

1.10.5 Site check fixes using LLM

> **Note**
>
> The `LLM/` directory at the root of the repository contains detailed instructions for editing site checks (in Markdown format): checklist, full guide to `checkType` / `data.json` / `urlProbe`, handling false positives, searching for public JSON APIs, and the proposal log for `socid_extractor`.

Main files:

- site-checks-playbook.md — short checklist
- site-checks-guide.md — detailed guide
- socid_extractor_improvements.log — template and entries for identity extractor improvements

These files should be kept up-to-date whenever changes are made to the check logic in the code or in `data.json`.

1.10.6 Activation mechanism

The activation mechanism helps make requests to sites requiring additional authentication like cookies, JWT tokens, or custom headers.

Let's study the Vimeo site check record from the Maigret database:

```
"Vimeo": {
    "tags": [
        "us",
        "video"
    ],
    "headers": {
        "Authorization": "jwt eyJ0..."
    },
    "activation": {
        "url": "https://vimeo.com/_rv/viewer",
        "marks": [
            "Something strange occurred. Please get in touch with the app's creator."
        ],
        "method": "vimeo"
    },
    "urlProbe": "https://api.vimeo.com/users/{username}?fields=name...",
    "checkType": "status_code",
    "alexaRank": 148,
    "urlMain": "https://vimeo.com/",
    "url": "https://vimeo.com/{username}",
    "usernameClaimed": "blue",
    "usernameUnclaimed": "noonewouldeverusethis7"
},
```

The activation method is:

```
def vimeo(site, logger, cookies={}):
    headers = dict(site.headers)
    if "Authorization" in headers:
        del headers["Authorization"]
    import requests

    r = requests.get(site.activation["url"], headers=headers)
    jwt_token = r.json()["jwt"]
    site.headers["Authorization"] = "jwt " + jwt_token
```

Here's how the activation process works when a JWT token becomes invalid:

1. The site check makes an HTTP request to `urlProbe` with the invalid token
2. The response contains an error message specified in the `activation/marks` field
3. When this error is detected, the `vimeo` activation function is triggered
4. The activation function obtains a new JWT token and updates it in the site check record
5. On the next site check (either through retry or a new Maigret run), the valid token is used and the check succeeds

Examples of activation mechanism implementation are available in activation.py file.

1.10.7 How to publish new version of Maigret

Collaborats rights are requires, write Soxoj to get them.

For new version publishing you must create a new branch in repository with a bumped version number and actual changelog first. After it you must create a release, and GitHub action automatically create a new PyPi package.

- New branch example: https://github.com/soxoj/maigret/commit/e520418f6a25d7edacde2d73b41a8ae7c80ddf39
- Release example: https://github.com/soxoj/maigret/releases/tag/v0.4.1

1. Make a new branch locally with a new version name. Check the current version number here: https://pypi.org/project/maigret/. **Increase only patch version (third number)** if there are no breaking changes.

```
git checkout -b 0.4.0
```

2. Update Maigret version in four files manually. **All four must be in sync** — the previous bump missed `docs/source/conf.py` and `snapcraft.yaml` and they fell behind by a release.

- `pyproject.toml` — single line `version = "X.Y.Z"` under `[tool.poetry]`.
- `maigret/__version__.py` — single line `__version__ = 'X.Y.Z'`.
- `docs/source/conf.py` — **two** Sphinx fields. `release` is the full version (`'X.Y.Z'`); `version` is the short `major.minor` (`'X.Y'`, **without** the patch number). Update **both**.
- `snapcraft.yaml` — single line `version:  X.Y.Z` (no quotes, no `v` prefix).

After editing, sanity-check with `grep -rE '0\.5\.|0\.6\.|<old>'` to catch any straggler reference.

3. Create a new empty text section in the beginning of the file *CHANGELOG.md* with a current date:

```
## [0.4.0] - 2022-01-03
```

4. Get auto-generate release notes:

- Open https://github.com/soxoj/maigret/releases/new
- Click *Choose a tag*, enter *v0.4.0* (your version)
- Click *Create new tag*
- Press + *Auto-generate release notes*
- Copy all the text from description text field below
- Paste it to empty text section in *CHANGELOG.txt*
- Remove redundant lines *## What's Changed* and *## New Contributors* section if it exists
- *Close the new release page*

5. Commit all the changes, push, make pull request

```
git add -p
git commit -m 'Bump to YOUR VERSION'
git push origin head
```

6. Merge pull request
7. Create new release

- Open https://github.com/soxoj/maigret/releases/new again
- Click *Choose a tag*

- Enter actual version in format *v0.4.0*
- Also enter actual version in the field *Release title*
- Click *Create new tag*
- Press + *Auto-generate release notes*
- **Press "Publish release" button**

8. That's all, now you can simply wait push to PyPi. You can monitor it in Action page: https://github.com/soxoj/maigret/actions/workflows/python-publish.yml

1.10.8 Documentation updates

Documentations is auto-generated and auto-deployed from the `docs` directory.

To manually update documentation:

1. Change something in the `.rst` files in the `docs/source` directory.
2. Install `python -m pip install -e .` in the docs directory.
3. Run `make singlehtml` in the terminal in the docs directory.
4. Open `build/singlehtml/index.html` in your browser to see the result.
5. If you edited any English `.rst` text (not just code blocks), refresh the per-language translation catalogs — see *Translations* below. Skipping this step lets the non-English builds silently fall back to English on the changed strings.
6. If everything is ok, commit and push your changes to GitHub.

1.10.9 Translations

The docs are translated via Sphinx's standard gettext workflow. English `.rst` files are the source of truth; translations live as `.po` catalogs under `docs/source/locale/<lang>/LC_MESSAGES/` (currently only `zh_CN`).

After editing any English `.rst` file, refresh the catalogs so existing translations stay aligned with the new strings:

```
cd docs
make intl-update LANG=zh_CN
```

This regenerates the `.pot` files via `sphinx-build -b gettext` and runs `sphinx-intl update` to merge them into the per-language `.po` files. New English strings appear with an empty `msgstr ""`; changed strings get a `#, fuzzy` marker that translators should review and re-translate.

Preview a translated build locally:

```
make html-zh_CN
open build/html_zh_CN/index.html
```

CJK escape-spaces gotcha

reStructuredText inline markup (bold, inline code, hyperlinks) requires whitespace or punctuation on both sides to close. In English this is free: a space or full stop always follows. In Chinese / Japanese / Korean translations the next character is often a CJK letter with no separator, and docutils then emits warnings like:

```
<translated>:1: WARNING: Inline strong start-string without end-string.
<translated>:1: WARNING: Inline interpreted text or phrase reference start-string␣
↪without end-string.
```

The fix is an explicit RST escape-space — a backslash followed by a space — between the closing marker and the next CJK character. In the rendered `.rst` this is written as \<space>; inside a `.po` `msgstr` it must be written as \\<space> because the `.po` parser eats one backslash level.

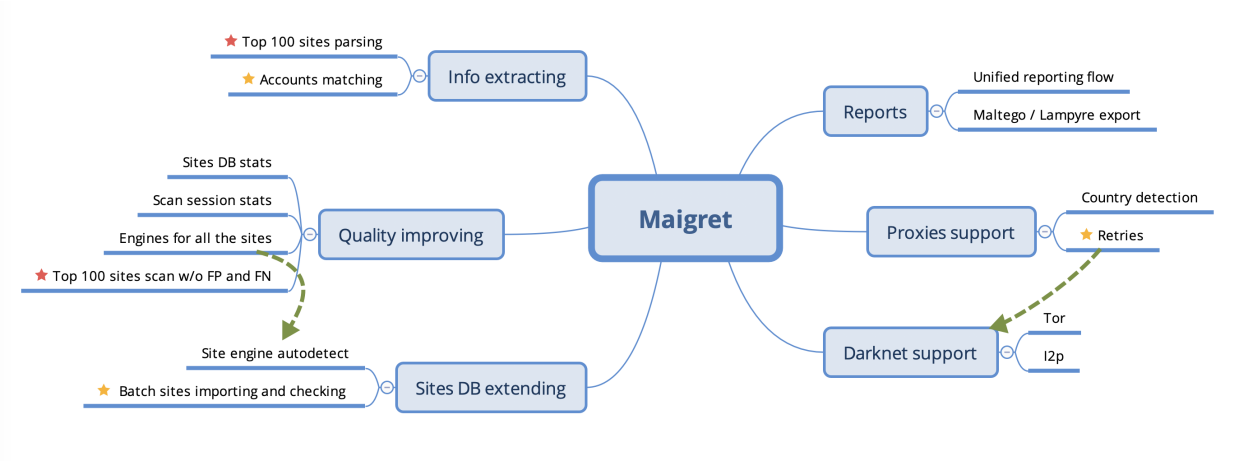
```
# WRONG - warning, markup leaks past the bold
msgstr "****"

# RIGHT - regular space breaks markup cleanly
msgstr " **** "

# RIGHT - escape-space when no visual space is wanted
msgstr "\\ ****\\ "
```

The same rule applies after inline code before a CJK character, and after a hyperlink before a CJK opening bracket — always insert \\<space>. After editing any `.po` file, run `make html-zh_CN`; these warnings only surface at build time.

To add a new language, run `make intl-update LANG=<code>` (e.g. `ja`, `de`, `pt_BR`) — this scaffolds an empty catalog. Read the Docs publishes each language as a separate project linked under the parent (see the Localization guide); a maintainer needs to create the translation project once in the RTD admin UI, set its language, and mark it as a translation of the main `maigret` project to enable the language switcher.

1.10.10 Roadmap

> **Warning**
>
> This roadmap requires updating to reflect the current project status and future plans.



1.11 Library usage

Maigret's CLI is a thin wrapper around an async Python API. You can embed Maigret in your own tools, pipelines, and OSINT workflows — no need to shell out.

This page covers the common patterns. For the full argument list of the underlying function, see `maigret.checking.maigret` in the source.

1.11.1 Installation

```
pip install maigret
```

1.11.2 Minimal example

A working end-to-end search against the top 500 sites:

```
import asyncio
import logging

from maigret import search as maigret_search
from maigret.sites import MaigretDatabase

# Load the bundled site database
db = MaigretDatabase().load_from_path(
    "maigret/resources/data.json"
)

# Pick which sites to scan (same filtering the CLI uses)
sites = db.ranked_sites_dict(top=500)

results = asyncio.run(
    maigret_search(
        username="soxoj",
        site_dict=sites,
        logger=logging.getLogger("maigret"),
        timeout=30,
        is_parsing_enabled=True,
    )
)

for site_name, result in results.items():
    if result["status"].is_found():
        print(site_name, result["url_user"])
```

Key points:

- `maigret_search` is an `async` function — wrap it with `asyncio.run(...)` or `await` it from inside your own event loop.
- `is_parsing_enabled=True` turns on `socid_extractor` so `result["ids_data"]` is populated with profile fields (bio, linked accounts, uids, etc.).
- Each entry in the returned dict has a `"status"` object with `is_found()`, plus `url_user`, `http_status`, `rank`, `ids_data`, and more.

1.11.3 Filtering sites

`ranked_sites_dict` accepts the same filters as the CLI:

```
# All sites tagged as coding, top 200 by rank
sites = db.ranked_sites_dict(top=200, tags=["coding"])

# Exclude NSFW and dating sites
```

(continues on next page)

(continued from previous page)

```
sites = db.ranked_sites_dict(excluded_tags=["nsfw", "dating"])

# Only specific sites by name
sites = db.ranked_sites_dict(names=["GitHub", "Reddit", "VK"])

# Include disabled sites (useful for maintenance / self-check)
sites = db.ranked_sites_dict(disabled=True)
```

1.11.4 Running inside an existing event loop

If your application already runs an asyncio loop (FastAPI, aiohttp server, a Discord bot, etc.), `await maigret_search` directly instead of calling `asyncio.run`:

```
async def check_username(username: str) -> dict:
    results = await maigret_search(
        username=username,
        site_dict=sites,
        logger=logger,
        timeout=30,
    )
    return {
        name: r["url_user"]
        for name, r in results.items()
        if r["status"].is_found()
    }
```

1.11.5 Routing through a proxy

The same proxy / Tor / I2P flags the CLI exposes are plain keyword arguments:

```
results = await maigret_search(
    username="soxoj",
    site_dict=sites,
    logger=logger,
    proxy="socks5://127.0.0.1:1080",
    tor_proxy="socks5://127.0.0.1:9050",   # used for .onion sites
    i2p_proxy="http://127.0.0.1:4444",     # used for .i2p sites
    timeout=30,
)
```

1.11.6 Full function signature

```
async def maigret(
    username: str,
    site_dict: Dict[str, MaigretSite],
    logger,
    query_notify=None,
    proxy=None,
    tor_proxy=None,
    i2p_proxy=None,
    timeout=30,
```

(continues on next page)

(continued from previous page)

```
    is_parsing_enabled=False,
    id_type="username",
    debug=False,
    forced=False,
    max_connections=100,
    no_progressbar=False,
    cookies=None,
    retries=0,
    check_domains=False,
) -> Dict[str, Any]
```

See *Command line options* for a description of each option — the semantics match the CLI flags one-to-one.

1.12 Settings

> **Warning**
>
> The settings system is under development and may be subject to change.

Options are also configurable through settings files. See settings JSON file for the list of currently supported options.

After start Maigret tries to load configuration from the following sources in exactly the same order:

```
# relative path, based on installed package path
resources/settings.json

# absolute path, configuration file in home directory
~/.maigret/settings.json

# relative path, based on current working directory
settings.json
```

Missing any of these files is not an error. If the next settings file contains already known option, this option will be rewritten. So it is possible to make custom configuration for different users and directories.

1.12.1 Database auto-update

Maigret ships with a bundled site database, but it gets outdated between releases. To keep the database current, Maigret automatically checks for updates on startup.

How it works:

1. On startup, Maigret checks if more than 24 hours have passed since the last update check.
2. If so, it fetches a lightweight metadata file (~200 bytes) from GitHub to see if a newer database is available.
3. If a newer, compatible database exists, Maigret downloads it to `~/.maigret/data.json` and uses it instead of the bundled copy.
4. If the download fails or the new database is incompatible with your Maigret version, the bundled database is used as a fallback.

The downloaded database has **higher priority** than the bundled one — it replaces, not overlays.

Status messages are printed only when an action occurs:

```
[*] DB auto-update: checking for updates...
[+] DB auto-update: database updated successfully (3180 sites)
[*] DB auto-update: database is up to date (3157 sites)
[!] DB auto-update: latest database requires maigret >= 0.6.0, you have 0.5.0
```

Forcing an update:

Use the `--force-update` flag to check for updates immediately, ignoring the check interval:

```
maigret username --force-update
```

The update happens at startup, then the search continues normally with the freshly downloaded database.

Disabling auto-update:

Use the `--no-autoupdate` flag to skip the update check entirely:

```
maigret username --no-autoupdate
```

Or set it permanently in `~/.maigret/settings.json`:

```
{
    "no_autoupdate": true
}
```

This is recommended for **Docker containers**, **CI pipelines**, and **air-gapped environments**.

Configuration options (in `settings.json`):

Setting	Default	Description
`no_autoupdate`	`false`	Disable auto-update entirely
`autoupdate_check_interval_hour`	`24`	How often to check for updates (in hours)
`db_update_meta_url`	GitHub raw URL	URL of the metadata file (for custom mirrors)

Using a custom database with `--db` always skips auto-update — you are explicitly choosing your data source.

1.12.2 Cloudflare webgate

> **Warning**
>
> **Experimental.** The `cloudflare_bypass` block is under active development; field names, defaults, and the trigger-protection routing rules may change without backwards-compatibility guarantees.

The `cloudflare_bypass` block in `settings.json` configures the optional bypass described in *Cloudflare webgate bypass*. The same block is honoured by the CLI, the Python API, and the web UI (`python -m maigret.web.app`); set `enabled: true` once and every entry point routes cf-protected sites through the solver.

Minimal FlareSolverr setup. Start the solver in Docker, then drop the following snippet into `~/.maigret/settings.json` (or any path listed in *Settings*):

```
docker run -d -p 8191:8191 --name flaresolverr \
    ghcr.io/flaresolverr/flaresolverr:latest
```

```
{
    "cloudflare_bypass": {
        "enabled": true,
        "modules": [
            {
                "name": "flaresolverr",
                "method": "json_api",
                "url": "http://localhost:8191/v1",
                "max_timeout_ms": 60000
            }
        ]
    }
}
```

That is enough — `session_prefix` and `trigger_protection` fall back to sensible defaults (`"maigret"` and `["cf_js_challenge", "cf_firewall", "webgate"]` respectively). On the next run, Maigret logs `Cloudflare webgate active: ...` and routes matching sites through the solver.

Default value (full schema with every supported field):

```
{
    "cloudflare_bypass": {
        "enabled": false,
        "session_prefix": "maigret",
        "trigger_protection": ["cf_js_challenge", "cf_firewall", "webgate"],
        "modules": [
            {
                "name": "flaresolverr",
                "method": "json_api",
                "url": "http://localhost:8191/v1",
                "max_timeout_ms": 60000
            },
            {
                "name": "chrome_webgate",
                "method": "url_rewrite",
                "url": "http://localhost:8000/html?url={url}&retries=1"
            }
        ]
    }
}
```

Fields.

Field	Description
`enabled`	When `true`, the bypass is active for every run; when `false` (the default), it activates only on `--cloudflare-bypass`.
`trigger_protection`	List of `site.protection` values that route a check through the webgate. Sites whose protection is empty or doesn't intersect this list use the default (aiohttp / curl_cffi) checker.
`session_prefix`	Prefix for the FlareSolverr `session` field. Maigret appends the process PID so concurrent runs don't collide. Reusing a session caches cf_clearance between checks of the same domain.
`modules`	Ordered list of backend modules. The first reachable module handles the check; later ones serve as a fallback chain.

Module methods.

- `json_api` — FlareSolverr-compatible POST endpoint at `url`. Preserves real upstream HTTP status, headers and final URL. Optional `max_timeout_ms` (default `60000`) is the per-request budget the solver is allowed to spend on the JS challenge.
- `url_rewrite` — legacy CloudflareBypassForScraping endpoint. The `url` must contain a `{url}` placeholder; the original probe URL is URL-encoded and substituted in. Returns rendered HTML only — `checkType: status_code` and `response_url` checks misfire under this method (treated as a synthetic HTTP 200 on success).

Optional ``proxy`` field (``json_api`` only).

A module may carry a `proxy` entry that the solver routes the upstream request through. Useful when a site enforces `ip_reputation` rules that block the solver host. Two forms are accepted:

```
{ "proxy": "socks5://localhost:1080" }
```

```
{ "proxy": { "url": "http://gw.example:3128",
             "username": "u",
             "password": "p" } }
```

Only `url`/`username`/`password` are forwarded; other keys are dropped. Cloudflare `Error 1015 / 1020` responses indicate the IP is rate-limited or banned — switch the proxy rather than retrying. .. _ai-analysis-settings:

1.12.3 AI analysis

The `--ai` flag (see *AI analysis (built-in)*) talks to an OpenAI-compatible chat completion API. Three settings control how that request is made:

Setting	Default	Description
`openai_api_key`	`""` (empty)	API key. If empty, Maigret falls back to the `OPENAI_API_KEY` environment variable.
`openai_model`	`gpt-4o`	Default model name. Overridable per-run with `--ai-model`.
`openai_api_base_url`	`https://api.openai.com/v1`	Base URL of the chat completion API. Point this at any OpenAI-compatible service (Azure OpenAI, OpenRouter, a local server, …) to use it instead of OpenAI directly.

Example `~/.maigret/settings.json` snippet using a non-OpenAI endpoint:

```
{
    "openai_api_key": "sk-...",
    "openai_model": "gpt-4o-mini",
    "openai_api_base_url": "https://openrouter.ai/api/v1"
}
```

The key resolution order is `settings.openai_api_key` → `OPENAI_API_KEY` environment variable; the first non-empty value wins.

Note

`--ai` sends the full internal Markdown report (which contains the gathered profile data) to the configured endpoint. Only use providers and accounts you trust with that data.

1.13 Tor, I2P, and proxies

Maigret can route checks through an HTTP/SOCKS proxy, the Tor network, or I2P. Three CLI flags cover three distinct goals — knowing which one you need is the most common stumbling block.

1.13.1 `--proxy` vs `--tor-proxy` (and `--i2p-proxy`)

The most-asked question (see issue #544):

- **You want every check to go through Tor** (e.g. you're on Tails OS, or behind a country-level block, or your IP is rate-limited). → Use `--proxy`, pointing at your Tor SOCKS port:

  ```
  maigret <username> --proxy socks5://127.0.0.1:9050
  ```

- **You want to reach ``.onion`` sites in the Maigret database**, while the rest of the run still uses your normal connection. → Use `--tor-proxy`:

  ```
  maigret <username> --tor-proxy socks5://127.0.0.1:9050
  ```

  `--tor-proxy` is **only** consulted for sites whose `url` is a `.onion` host. For every other site Maigret uses your direct connection (or `--proxy` if set). Without `--tor-proxy`, `.onion` sites are silently skipped.

The same split applies to `--i2p-proxy`: it is consulted only for `.i2p` hosts, never for clearweb sites.

Defaults: `--tor-proxy` defaults to `socks5://127.0.0.1:9050` and `--i2p-proxy` to `http://127.0.0.1:4444`. `--proxy` has no default. Maigret does **not** launch `tor` or an I2P router for you — start the daemon first.

1.13.2 Tor Browser vs system `tor`: port numbers

The SOCKS port differs by Tor installation:

- **System ``tor`` daemon** (`apt install tor`, `brew install tor`, Tails) listens on `9050`.
- **Tor Browser bundle** ships its own `tor` listening on `9150`.

If a connection refuses, try the other port:

```
# system tor
maigret <username> --proxy socks5://127.0.0.1:9050
```

(continues on next page)

(continued from previous page)

```
# Tor Browser running in the background
maigret <username> --proxy socks5://127.0.0.1:9150
```

1.13.3 A note on results over Tor

Most public WAFs (Cloudflare, DDoS-Guard, AWS WAF, Akamai) block Tor exit nodes by default — usually more aggressively than they block datacenter IPs. A Tor run typically produces **more UNKNOWNs and fewer CLAIMEDs** than the same run from a residential connection. This is not a bug in Maigret; it is the cost of anonymity.

Recommended flags for a Tor run:

```
maigret <username> --proxy socks5://127.0.0.1:9050 --timeout 60 --retries 2
```

- `--timeout 60` — Tor circuits add 1–3 seconds per request; the default 30 s causes spurious timeouts.
- `--retries 2` — retries cover transient circuit failures, which are common on Tor.
- Optional `-n 20` — lowering concurrency (default 100) reduces the chance of exits rate-limiting you.

If you mainly need to bypass WAFs (rather than to remain anonymous), a residential proxy will usually outperform Tor by a wide margin. See the **"Lots of sites fail / timeout / return 403"** section in TROUBLESHOOTING.md.

1.13.4 Running on Tails OS

Tails forces every outbound connection through Tor at the network layer. Maigret needs no special configuration to comply — pointing `--proxy` at the Tails Tor daemon is enough:

```
maigret <username> --proxy socks5://127.0.0.1:9050 --timeout 60
```

Things that are **not** needed:

- `torsocks maigret ...` and `torify maigret ...` — these wrap libc socket calls, but Maigret's HTTP client (`aiohttp` / `curl_cffi`) bypasses libc for network I/O, so the wrapper has no effect. Use `--proxy` instead.
- `--tor-proxy` — on Tails, *everything* must go via Tor (the OS enforces this), so the niche "only .onion via Tor" mode that `--tor-proxy` provides does not apply.

Installation over Tor on Tails

`pip` itself does not know about Tor; on Tails you need `torsocks` to wrap it:

```
torsocks pip install --user maigret
```

After install, the binary lands in `~/.local/bin/maigret`. If `maigret: command not found`, either add `~/.local/bin` to `PATH` or invoke it as `python3 -m maigret <username>`.

Persisting Maigret across Tails sessions

Tails wipes `~/.local/` on reboot unless you configure the Persistent Storage to keep it. This is Tails configuration, not Maigret configuration — see the official Tails docs:

- Persistent Storage on Tails
- Configuring Persistent Storage features

A step-by-step recipe contributed by a user (persisting `~/.local/lib/python3.9` and `~/.local/bin` and patching `.bashrc`) is in issue #544. Treat it as a starting point: the Python version and Tails internals change between Tails releases.

Reports on Tails — where to save them

The default `reports/` directory lives next to the working directory and is wiped with the amnesiac session. To save reports somewhere persistent, either pass `-fo`:

```
maigret <username> --html -fo "/home/amnesia/Persistent/maigret-reports"
```

or set `"reports_path"` in your `settings.json` to a persistent path. See *Settings*.

1.13.5 Programmatic equivalents (Python library)

The same options are available through the Python API. See *Library usage* — the relevant keyword arguments are `proxy=`, `tor_proxy=` and `i2p_proxy=`, accepting the same URL formats as the CLI flags.

1.13.6 See also

- *Command line options* — full reference for the three flags.
- TROUBLESHOOTING.md — quick recipes for `.onion` / I2P sites and for WAF-induced 403s.
- *Library usage* — proxy options for embedded use.

1.14 Cryptocurrency & Web3 Investigations

Blockchain transactions are public, but the people behind wallets are not. Maigret helps bridge this gap by finding Web3 accounts tied to a username, revealing the person behind a pseudonymous crypto persona.

1.14.1 Why it matters

Crypto investigations often start with a wallet address or an ENS name but hit a wall — the blockchain tells you *what* happened, not *who* did it. A username, however, is reused across platforms. If someone trades on OpenSea as `zachxbt` and posts on Warpcast as `zachxbt`, Maigret connects the dots and builds a full profile.

Common scenarios:

- **Scam attribution.** A rug-pull promoter uses the same alias on Fragment (Telegram username marketplace), OpenSea, and a personal blog.
- **Sanctions compliance.** Verifying whether a counterparty's online footprint matches known sanctioned individuals.
- **Due diligence.** Before an OTC deal or DAO vote, checking whether the other party has a consistent online presence or is a freshly created sockpuppet.
- **Stolen funds tracing.** A stolen NFT appears on OpenSea under a new account — but the username matches a Warpcast profile with real-world links.

1.14.2 Supported sites

Maigret currently checks the following crypto and Web3 platforms:

Site	What it reveals	Notes
OpenSea	NFT collections, trading history, profile bio, linked website	
Rarible	NFT marketplace profile, collections, listing history	Complements OpenSea for NFT attribution across marketplaces
Zora	Zora Network profile, minted NFTs, creator activity	Ethereum L2 creator platform; useful for on-chain art attribution
Polymarket	Prediction-market profile, positions, public portfolio P&L	Useful for political/financial prediction attribution
Warpcast (Farcaster)	Decentralized social profile, posts, follower graph, Farcaster ID	Every Farcaster ID maps to an Ethereum address via the on-chain ID registry
Fragment	Telegram username ownership, TON wallet address, purchase date and price	Valuable for linking Telegram identities to TON wallets
Paragraph	Web3 blog/newsletter, ETH wallet address, linked Twitter handle	Richest cross-platform data among crypto sites
Tonometerbot	TON wallet balance, subscriber count, NFT collection, rankings	TON blockchain analytics
Spatial	Metaverse profile, linked social accounts (Discord, Twitter, Instagram, LinkedIn, TikTok)	Rich cross-platform links
Revolut.me	Payment handle: first/last name, country code, base currency, supported payment methods	Not strictly Web3, but widely used by crypto OTC traders for fiat off-ramps; the public API returns structured KYC-adjacent data

1.14.3 Real-world example: zachxbt

ZachXBT is a well-known on-chain investigator. Let's see what Maigret can find from just the username `zachxbt`:

```
maigret zachxbt --tags crypto
```

Maigret finds 5 accounts and automatically extracts structured data from each:

Fragment — confirms the Telegram username `@zachxbt` is claimed, reveals the TON wallet address (`EQBisZrk...`), purchase price (10 TON), and date (January 2023).

Paragraph — the richest result. Returns the real name used on the platform (`ZachXBT`), bio (`Scam survivor turned 2D investigator`), an Ethereum wallet address (`0x23dBf066...`), and a linked Twitter handle (`zachxbt`). The `wallet_address` field is especially valuable — it directly links the pseudonym to an on-chain identity.

Warpcast — Farcaster profile with a Farcaster ID (`fid: 20931`), profile image, and social graph (33K followers). Every Farcaster ID is tied to an Ethereum address via the on-chain ID registry, so this is another on-chain anchor.

OpenSea — NFT marketplace profile with bio (`On-chain sleuth | 10x rug pull survivor`), avatar (hosted on `seadn.io` with an Ethereum address in the URL path), and a link to an external investigations page.

Hive Blog — blockchain-based blog account created in March 2025. Low activity (1 post), but confirms the username is claimed across blockchain ecosystems.

From a single username, Maigret produces:

- **2 wallet addresses** — one TON (from Fragment), one Ethereum (from Paragraph)
- **1 confirmed Twitter handle** — `zachxbt` (from Paragraph)
- **1 Telegram username** — `@zachxbt` (from Fragment)
- **1 external link** — `investigations.notion.site` (from OpenSea)

- **Social graph data** — 33K Farcaster followers, blog activity timestamps

This is enough to pivot into blockchain analysis tools (Etherscan, Arkham, Nansen) using the wallet addresses, or into social media analysis using the Twitter handle.

1.14.4 Workflow: from username to wallet

Step 1: Search crypto platforms

```
maigret <username> --tags crypto -v
```

Review the results. Pay attention to:

- **Fragment** — if the username is claimed, you get a TON wallet address directly.
- **Paragraph** — blog profiles often contain an ETH address and a Twitter handle.
- **Warpcast** — Farcaster IDs map to Ethereum addresses via the on-chain registry.
- **OpenSea** — avatar URLs sometimes contain wallet addresses in the path.

Step 2: Expand with extracted identifiers

Maigret automatically extracts additional identifiers from found profiles (real names, linked accounts, profile URLs) and recursively searches for them. This is enabled by default. If Maigret finds a linked Twitter handle on a Paragraph profile, it will automatically search for that handle across all sites.

Step 3: Cross-reference with non-crypto platforms

The real power is connecting crypto personas to mainstream accounts. Drop the tag filter:

```
maigret <username> -a
```

This checks all 3000+ sites. A match on GitHub, Reddit, or a forum can reveal the person behind the wallet.

1.14.5 Workflow: from wallet to identity

If you start with a wallet address rather than a username, you can use complementary tools to get a username first:

1. **ENS / Unstoppable Domains** — resolve the wallet address to a human-readable name (`vitalik.eth`). Then search that name in Maigret.
2. **Etherscan labels** — check if the address has a public label (exchange, known entity).
3. **Fragment** — search the TON wallet address to find which Telegram usernames it purchased.
4. **Arkham Intelligence / Nansen** — blockchain attribution platforms that may tag the address with a known identity.

Once you have a username candidate, feed it to Maigret.

1.14.6 Tips

- **Username reuse is the #1 signal.** Crypto-native users often reuse their ENS name (`alice.eth`) or a variation (`alice_eth`, `aliceeth`) across platforms. Try all variations.
- **Fragment is uniquely valuable** because it directly links Telegram usernames to TON wallet addresses — a rare on-chain / off-chain bridge.
- **Warpcast profiles are Ethereum-native.** Every Farcaster account is tied to an Ethereum address via the ID registry contract. If you find a Warpcast profile, you implicitly have a wallet address.

- **Paragraph often has the richest data** — wallet address, Twitter handle, bio, and activity timestamps in a single API response.
- **Use `--exclude-tags` to skip irrelevant sites** when you're focused on crypto:

```
maigret alice_eth --exclude-tags porn,dating,forum
```

1.15 Academic & Research Investigations

Academic identity has a unique anchor that social-media identity does not: the **ORCID** (Open Researcher and Contributor ID). Every ORCID is verified, globally unique, and tied to a confirmed email — a single confirmed ORCID match between two platforms means the **same human**, with effectively zero false-positive rate.

Maigret uses ORCID as a first-class identifier: extract it once from a username-keyed profile (iNaturalist, GitHub bio, lab homepage), then pivot into the academic graph (ORCID, OpenAlex, arXiv, DBLP, Scholia) — a chain of HTTP calls that turns an anonymous handle into a CV with employer, education, publication list, citation count, co-author graph, and topical area.

1.15.1 Why it matters

Most OSINT work on academic targets stalls at the same bottleneck: people use one alias on a citizen-science site, a different alias on Twitter, a third in a forum signature, and only their real name on PubMed. Connecting the personas by hand means trawling Google Scholar, ResearchGate, university web pages, and old conference programmes. ORCID short-circuits this entirely — one verified identifier resolves all of them.

Common scenarios:

- **Researcher background check.** Before a collaboration, a preprint review, or a grant decision — verifying that a claimed h-index, employment history, and publication count actually match the public ORCID record.
- **Co-author graph reconstruction.** From a single ORCID, OpenAlex returns every co-author, their institutions, and topical clusters — useful for spotting undisclosed conflicts of interest.
- **Paper mill / fraud investigation.** When a suspicious paper's authors share an ORCID-claimed identity, the cross-platform footprint (or absence of one) is itself evidence.
- **Pseudonym deanonymisation.** A wildlife photographer posts naturalist observations under the handle `kueda`. iNaturalist's public API returns their ORCID, and one further request reveals their real name, employer (California Academy of Sciences), education (UC Berkeley), and 700+ citable observations.
- **Award / appointment due diligence.** Confirming a Turing-Award claim, a tenure status, or a society fellowship via the DBLP and ORCID activity timelines rather than a CV PDF.

1.15.2 Supported sites

Maigret currently checks the following ORCID-keyed platforms (use `--id-type orcid` when starting from a bare ORCID, or rely on the recursive chain when starting from a username):

Site	What it reveals	Notes
ORCID	Full name, biography, employment, education, researcher URLs (homepages and social), keywords, country, linked external IDs (Scopus, ResearcherID, Loop), publication summary, email-verified status	All disciplines — ORCID is a field-agnostic identifier issued to any researcher.
OpenAlex	Display name and name alternatives, works count, total citations, h-index, i10-index, last-known institutions (with country), topical areas, raw author-name variants from publications	All disciplines — indexes works across the entire scholarly literature, from humanities to medicine.
arXiv	Author preprint listing — paper titles, IDs, dates	Strongly biased toward physics, math, CS (and quantitative biology, statistics, EE, quantitative finance, economics).
DBLP	Full name, DBLP person ID, paper count, affiliation, awards (e.g. Turing Award), homepage URLs, links to Google Scholar / ResearchGate / Scopus	Computer science only — a biologist or pure economist will not be in the index.
Scholia	Wikidata QID for the author, linked publications, co-author graph, employer/affiliation timeline, topical clusters	All disciplines, but only if the author has been curated in Wikidata — coverage is broadest for prominent figures (award winners, senior faculty, deceased researchers).

1.15.3 Workflow: from username to ORCID

Step 1: Find a profile that publishes ORCID

The most reliable bridges from a username to an ORCID are platforms whose users *want* their academic identity discoverable. In practice:

- **iNaturalist** — biologists, naturalists, citizen scientists. Public API `api.inaturalist.org/v1/users/{username}` returns the ORCID directly in the `orcid` field. Maigret extracts it automatically.
- **GitHub** — scientists often paste their ORCID URL into the **bio** or **blog** field on their GitHub profile, or under a *generic* entry in `/users/{u}/social_accounts`. A pattern like `orcid\.org/(\d{4}-\d{4}-\d{4}-\d{3}[\dX])` matches both `http://orcid.org/...` and `orcid.org/...` variants.
- **Lab homepage / institutional bio.** Use `--parse <URL>` to scrape an arbitrary page for known identifiers — useful when the target's footprint lives at `faculty.example.edu/~jdoe`.

Step 2: Let the recursive search run

```
maigret <username> --tags science -v
```

Recursive search is enabled by default. As soon as Maigret extracts an `orcid` field from any found profile, it queues an `--id-type orcid` second wave automatically. No manual chaining needed.

Step 3: Start from a bare ORCID

If you already have the ORCID (e.g. from a paper's author footer, a grant database, or a Wikidata entry):

```
maigret 0000-0002-9322-3515 --id-type orcid -a
```

This bypasses the username-bridge step entirely.

1.15.4 Workflow: from ORCID to mainstream identity

The ORCID record itself contains the link back to ordinary social platforms:

1. **ORCID ``researcher-urls``** — a list of self-declared external links. Typical entries: lab homepage, Twitter/Bluesky/Mastodon profile, personal blog. These are entered by the researcher and therefore confirmed.
2. **ORCID ``external-identifiers``** — IDs from sibling academic systems (Scopus Author ID, ResearcherID/Publons, Loop profile). Each unlocks another extraction pipeline.
3. **OpenAlex ``last_known_institutions``** — current employer with country, ROR ID, and OpenAlex institution ID; useful for pivoting into the institution's directory.
4. **DBLP** `<url>` **tags** — DBLP records often embed direct links to Google Scholar, ResearchGate, and personal pages.
5. **Scholia ``wikidata_qid``** — once you have a Wikidata QID, the SPARQL endpoint unlocks the broadest external ID set in any single OSINT system: VIAF, LCCN, GND, Twitter handle, Mastodon handle, IMDb, GitHub, ORCID, and ~200 others.

1.15.5 Tips

- **Email comes from ORCID** — within the academic chain (ORCID, OpenAlex, arXiv, DBLP, Scholia) it is the only source that returns an email address, and only when the researcher has marked their primary email public. The same response also carries `history.verified-primary-email`, an ORCID-confirmed flag that means the address was email-validated at registration. Use this as a strong pivot into email-keyed lookups (Holehe, HIBP, Gravatar, etc.).
- **Outside the academic chain, GitLab is the dev platform most likely to leak an email** via its public `public_email` field — relevant for scientists who self-host code on a CERN/research-institute GitLab. GitHub's REST API does not expose email by default.
- **Compare OpenAlex** `raw_author_names` **against the ORCID** `other-names`. Discrepancies often expose pre-marriage names, transliterations from non-Latin scripts, or co-author misattributions worth flagging.
- **Anything fed into** `--id-type orcid` **must match** `^\d{4}-\d{4}-\d{4}-\d{3}[\dX]$`. The trailing character may legitimately be `X` (ISO checksum); strip `https://orcid.org/` prefixes before passing the bare ID.